Preface

This is the report for the bachelor's thesis for three of the students at Bachelor in IT-operations and Information Security, commonly abbreviated as BITSEC.

The task given is the development of a physical demo to demonstrate "how cool IT-operation actually is". This report documents the design and implementation process, and the reasoning behind choices we made along the way.

We would like to thank Erik Hjelmås at NTNU in Gjovik for assigning us this task, and for showing us a huge amount of support during the development process. We will also thank Ernst Gunnar Gran, also at NTNU in Gjovik, for being our supervisor and giving us valuable feedback during our weekly meetings discussing the bachelor's thesis.

Also a thanks to the Department of Information Security and Communications Technology (IIK) for providing us with the equipment we needed to develop the demo.

Contents

List of Figures

Listings

1 Introduction

1.1 Background

The Department of Information Security and Communication Technology (hereafter known as IIK) at NTNU in Gjøvik [2] is widely known for its information security. One of potentially several subject areas, IIK is especially known for its research in biometrics and its fingerprint sensor; a physical demo which have been used several times when companies and governmental entities visits the university.

The field of information security is about studies of e.g., risk management [3] [4], incident response handling [5] [6] and legal/management studies [7] [6]. In other words, these studies is about doing management and technical based work on already existing infrastructure.

However, the Department of Information Security and Communication Technology includes not only information security. There is also IT operations, a complementary field of study to information security.

The field of IT-operation studies includes a lot of technical aspects which also comprises information security. However, IT-operations is also about creating infrastructure from scratch, for instance.

IIK wanted a small, physical demo that is related to showcasing IT-operations. The demo should show how exciting and interesting the field of IT-operations is. With Erik Hjelmås as the contractor, it was suggested that such a demo should be developed as a bachelor's thesis. The project group became interested in this project and got it assigned.

1.2 Task Description

The task was to develop a physical object to showcase the application of IT-operations. IIK with Erik Hjelmås (hereafter known as the contractor) wants us to create something "cool", which can be showcased to the an audience. The idea is to demonstrate to the audience how IT-operations works and how it is used on a day-to-day basis.

The day-to-day basis applies to situations where a digital infrastructure grows or shrinks based on the amount of traffic. It will also show how the infrastructure tackles that one or several devices are removed.

1.2.1 Delimitation

The task is to focus on developing one system, prototype it and in the end release it to the contractor. The group will not develop several systems in parallel and decide which system to expand on. A lot of features can be demonstrated in the demo. In consultation with supervisor and contractor, it is decided to demonstrate scalability and resilience. Scalability and resilience are core topics in the study of IT-operations.

1.3 Purpose

This project is meant to give the Department of Information Security and Communication Technology a demo which can be used to showcase to an audience how IT-operation works in a way that anyone should be able to understand.

This accompanied project report is to give the reader an understanding of the project group's choices behind the development of the demo. It will give a technical insight into the demo, and it is a valuable read to those who finds the demo interesting. The project group chose this task because, first and foremost, the members wanted a technical task. Compared to the other tasks that were offered as a bachelor's thesis, this task stands out because the majority of the task is to develop something physical.

1.4 Target Group

The target group for the demo is twofold:

Firstly, it is an interactive demo for laymen; from kindergarten kids to company leaders, and even more. The demo is a simple "push on buttons and see what happens" demonstration, which is not a difficult task for the vast majority. However, for those who find it extra interesting, there is this report. This report will be publicly available online a while after it is handed in [8]. Those who want, in addition to trying out the physical demo, can read this report as well.

On the other side, there is the contractor, who must maintain the project after the project group is done. For the contractor, this report is important for understanding what is going on "under the hood". When the project is finished with the bachelor's thesis, the ownership of the project goes to IIK. IIK needs documentation in order to maintain the project. There is also a repository on GitHub associated with the project, which contain all the code for the project. IIK will be able to create a copy of that repository and continue working on it.

1.5 The Project Group

The project group consists of three students, all attending the Bachelor in IT-operations and Information Security (BITSEC).

Each of the students' come from different backgrounds: One has been working a full-time job for several years. One has already completed one bachelor's degree in music education and is currently working on the second one in IT-operations. One is coming straight from high school.

From the three years course, the members has studied together and acquired knowledge and skills in programming, software engineering, networking, operating systems, service architecture operations and infrastructure as code, to mention some. These are all subjects that has been necessary to apply in working on this project. The project group has used these subjects as a base for working on the project, which contains technologies that the project group had little to no knowledge or experience.

1.6 Structure

The project report consists of this introduction chapter and seven more chapters:

Chapter 2 - The Demo - Discussion of the different approached to how the demo should be, and an explanation of how the demo will work is given.

Chapter 3 - Design - Description of the physical and logical design of the project demo.

Chapter 4 - Selection of Tools - Description of what technical tools were used in the project, and the discussion and reasoning behind why it was decided to use those tools.

Chapter 5 - Development Process - Description of the process of developing the project demo, with the practical use of the technical tools described in the previous chapter.

Chapter 6 - Testing - Dealing with testing of the different function of the demo, with focus on scalability and resilience.

Chapter 7 - Results and Discussion - The project group writes about the final product; how it is fulfilling the requirements the project group were given, and challenges that occurred during the development process.

Chapter 8 - Ending - A conclusion chapter which wraps up the development of the physical demo. It also discusses further work and evaluations of the task and work process.

1.7 Roles

The contractor is the Department of Information Security and Communication Technology, represented by Erik Hjelmås. The supervisor is Ernst Gunnar Gran, also an employee in the same department. Internal in the project group, Per-Kristian Kongelf Buer is the project leader. Other roles have been distributed under its course of the project.

1.8 Terminology

This report contains words that are not self-explanatory to someone reading this report. Following is a list of terminologies which are used in the report, with a brief explanation of what they mean.

Access Point
Hardware that allows Wi-Fi devices to connect to a wired network. The access point usually connects to a router via a physical cable.

Ansible
Ansible is an open source configuration manager. It uses a declarative language to describe system configuration.

Ansible Role
A predetermined set of instructions to achieve a specific condition. E.g: a role to install Docker on the server.

Ansible PlayBook
A collection of Roles are laid out to get a machine into a desired state. These roles are combined in a PlayBook to bundle them, so that after you run a PlayBook the desired state is

achieved. E.g:Install several applications and configure them to be ready for use.

Ansible Galaxy
Ansible Galaxy is a community where other Ansible users can publish their roles

Apache
Open source HTTP server project whose goal is to develop and maintain HTTP services for modern operating systems [9].

Cluster
A cluster is a set of masters and nodes communicating together.

Containerization
Bundling an application together with all of its required files including configurations, libraries and dependencies to run in in a bug-free way regardless of the environment [10].

Docker
Docker is an Open-source software which does operating system level virtualization to run containers [11].

Domain-Specific Language (DSL)
A language that has a specific purpose for an application or a program. Examples of these are the more widely known web-languages, like HTML and CSS [12]. There are also domain-specific languages for configuration management, such as Ansible, which is used in the demo.

IEEE 802.11
A set of LAN protocols for implementing wireless communication on different frequencies, such as the 2.4GHz and the 5GHz radio frequency.

Kubectl
The command tool to use Kubernetes.

Kubelet
The service that reads the manifests and makes sure the containers are all running as defined.

Kubernetes
"*Kubernetes is a portable, extensible open-source platform for managing containerized workloads and services*" [13].

Master Node
The node that controls the cluster and all the worker nodes. This is where services are created and scheduled.

Worker Node
These are the workers in the cluster, they perform the service requested by the master.

Pod
One or more containers that gets deployed on a worker node. The containers in a pod share IP address and resources [14].

Raspbian
A free Debian based operating system developed and optimized for the Raspberry Pi. Raspbian comes with over 35000 packages and pre-compiled software [15].

Raspberry Pi
A small affordable single-board computer developed by The Raspberry Pi Foundation. Originally intended to promote computer science in third world countries, their popularity exceeded all expectations and became the third most selling general purpose computer [16].

ReplicaSet
The type of Kubernetes configuration which purpose is to maintain a specified number of pods, regardless of nodes [17].

SSH
Secure Shell, abbreviated SSH is a protocol for doing network services over an unsecured network. Usages include command line login and remote command execution [18].

2 The Demo

This chapter provides an explanation of *scalability* and *resilience*, the two keywords this demo is mainly about.

Then the chapter focuses on different approaches to the demo. It was not explicitly written in the task description what was to be developed. A brief explanation of the different approaches, with its advantages and disadvantages, will be explained here.

There will be given an explanation of what the requirements for the demo are. There will also be explained how the demo will work in general.

2.1 Scalability and Resilience

Scalability and *resilience* are two major keywords in this demo. Both scalability and resilience are an important aspect to the field of IT-operations, and is thus what this demo mainly should demonstrate. A brief definition of these two keywords follows in the subsections below.

2.1.1 Scalability

The quality of service must be maintained regardless of how much traffic the system is currently handling. Scalability is a technique that ensures this is achieved by deploying more servers when the traffic increases and remove server when traffic decreases [19].

2.1.2 Resilience

Failures in an IT-system or a digital infrastructure are normal. For the most part, 100% operational excellence is near impossible to achieve. Therefore, resilience is built into the infrastructure. These can be things like redundancy (several ways to the same destination, in case one way is cut off) and auto scaling (deploying or removing servers automatically based on the amount of traffic) [20].

2.2 Different approaches to the demo

The requirement specification in the task description states that there are to be built a "cool" demo showing elements of scalability and resilience. The IT-operations course can show off this demo to attract people towards studying IT-operations at NTNU in Gjovik.

The initial question the project group had to ask themselves was “What is cool?”. "Cool" is a very subjective term. It is almost impossible to define the definition of cool when speaking to a larger audience containing different gender, age and interests. Nevertheless, it had to be possible to create something most people would try out at and think *“Hey, that’s pretty neat!”*.

The project group, in accordance with the contractor and the supervisor, wanted to settle for something most people will find interesting: A visual cue, something one can look at and interact with. Although it may be "cool" for someone to look at containers scaling up and down based on traffic in a system, something simpler was needed.

Different approaches to how such a system would be developed follows.

2.2.1 Activities

It was discussed that the demo could be some kind of activity where the audience could participate. This could be a Kahoot-quiz [21] or a table-top hacking demo, according to the task description.

The positive side of an activity is that it includes the audience taking part in an interactive process, where the main keywords scalability and resilience are in focus. The participants could learn about this in a fun and challenging way.

The negative is that this approach is not technical at all. It fits poorly in a topic that is all about the technical stuff. This approach would have worked if there were technical aspects involved.

This approach, however, was ruled out, in favour of investing most of the time developing something physical.

2.2.2 Virtual machines, Containers and the Cloud

Virtual machines, containers and the cloud are technical aspects that are very relevant for the demo.

The positives is that virtual machines and containers can be used interactively by the audience to scale up and down the number of virtual machines and/or containers. The audience can also learn about scalability and resilience in the cloud with such kind of demo.

The negative is that such a demo will be very logic oriented, meaning that the whole demo will be outputted as command line text. This is very interesting for those in the audience who are into computers, networks and IT-operations and have some prior experience with the command line. For those uninterested, however, will not understand, and might actually become less interested. It is reasonable to believe that most of the audience are not interested in computers and networks on such a high level that something useful can be achieved out of it. Most of the audience will in all likelihood find the demo confusing and boring.

The technical aspects in this approach are good, but they need to be more physical oriented in order to embrace a wide audience.

2.2.3 A Raspberry Pi Cluster

A group of several Raspberry Pi joined in a cluster was considered strongly as the approach to develop a "cool" demo. A Raspberry Pi is a tiny, low powered computer that suits the needs for the demo perfectly. With such small devices, the audience is able to interact with something physical that is able to demonstrate scalability and resilience.

A set of requirements for the Raspberry Pi cluster is described in the following section.

2.3 Requirements for the Raspberry Pi Cluster

A Raspberry Pi cluster has several requirements which makes it ideal to develop such a demo:

- A Raspberry Pi cluster is a highly mobile setup, since the Raspberry Pi are tiny devices. This makes them easy to move to different places, and they can power on regardless of a nearby power plug, with power being fed from a power bank. A Raspberry Pi cluster is excellent for bringing to various exhibitions and other remote events.
- A Raspberry Pi cluster can be set up so it is not dependent on Internet connection when it has no use for such a connection,

- A set of Raspberry Pi and power banks are not very expensive, especially when an institute such as IIK is responsible for the procurement of these devices. Therefore, a larger quantity of Raspberry Pi and power banks can be purchased simultaneously.

2.4 How the demo will work

The demo consists of a cluster of eight Raspberry Pis. These devices get power from four power banks, which each can power two Raspberry Pis each. On each Raspberry Pi there is an attached small screen. The eight Raspberry Pis is joined together in a group, which is called a *cluster*. When the Raspberry Pi are joined in a cluster, each Raspberry Pi is referred to as a *node*. In the cluster, there is one *master node*, one *access point node* and six *worker nodes*. To see an explanation of these words, see terminology at [1.8].

The master node distributes jobs to each of the six worker nodes. The access point gives network connection to itself and all the other nodes in the cluster. The access point node also ensures that the configuration on all the nodes are up to date and won't change. The worker nodes executes the jobs they receive from the master node. When the worker nodes executes a job, a colour is shown on the screen. This colour represents how much the worker node is working at the moment. In other words, the colour represents how much the current central processing unit (CPU) [22] usage on the worker node is. A light colour indicates not much to do, while a darker colour indicates more work to do.

On the master node, there are five buttons on the screen that can be pressed. When pressing the different buttons, the master node sets the number of jobs sent to the worker nodes. The higher amount of jobs, the harder the worker nodes will work and the CPU usage will increase. A darker colour will be shown on the screen on the worker nodes. When sending lesser jobs, the worker nodes will not work so much, so the CPU usage will decrease. A lighter colour on the screen on the worker nodes is shown then.

The rightmost button on the screen on the worker nodes will stop colours from showing on that specific worker node's screen.

One or several worker nodes can be removed from the cluster, like removing the power cable so that a worker node will turn itself off, for instance. What happens next is that the master node will redirect the jobs from the now disabled worker node to the rest of the worker nodes that is still running. This can be seen as the colour of the remaining worker nodes will turn darker, since the CPU on those worker nodes will increase.

3 Design

This chapter will deal with the physical and logical design of the demo.

In the physical part, it will be explained how the physical design is set up and a brief explanation of the main components.

In the logical part, the non-visible components are explained. How the colour scale is made up and the choice of colours will also be explained.

3.1 Physical Design

The physical cluster consist of eight Raspberry Pi as shown in Figure 1. Each Raspberry Pi has a screen attached to them. The Raspberry Pis are all powered by four 20000 MWh power banks, with two outputs each. The project group, in collaboration with supervisor and project owner, decided to go for a wireless solution to make the product portable and easier to transport around to show off to a wide variety of audiences. A closed network also means the cluster works without needing an Internet connection, and therefore requires less cable management.

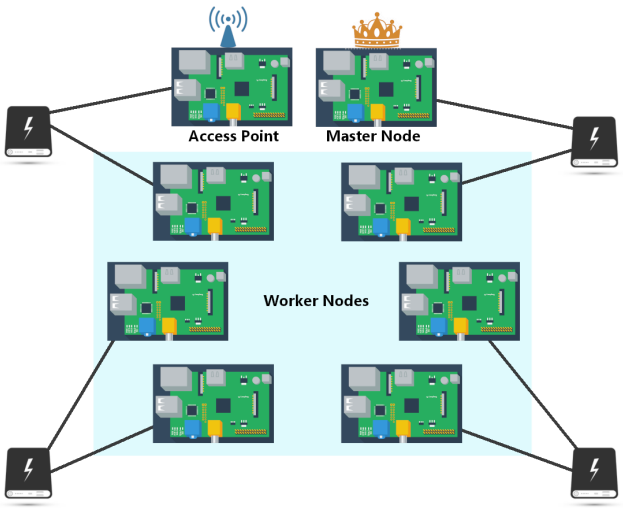


Figure 1: The physical design of the Raspberry Pi cluster, consisting of eight Raspberry Pi getting power from four power banks.

3.1.1 Raspberry Pi 3 Model B+

The Raspberry Pi is a small single-board computer. It has a variety of different models. It was decided that the Raspberry Pi 3 Model B+ is suitable for the requirements set for the Raspberry Pi cluster due to the following:

The Raspberry Pi 3 Model B+ has four USB slots, in which one can be connected to a power

bank and be fed power. It also has wireless network built-in, which makes the Raspberry Pi 3 Model B+ not dependent on cables and an external switch [23]. If wireless network was not a built-in feature, an external switch had to be used in order to let the network traffic from one Raspberry Pi to another.

Detailed information about the Raspberry Pi 3B+ can be found in the chapter that describes the hardware in [4.1.2].

3.1.2 Screen

The Raspberry Pi needs a component that can display colours visually. An attachable screen from 4D Systems called the *4DPi-24-HAT* [24] was chosen as the component that displays the visuals. The screen size is 2.4 inches and is compatible with the Raspberry Pi 3 Model B+. It gets power directly from the Raspberry Pi it is attached to, so no external power is needed for the screen.

3.1.3 Power Banks

The Raspberry Pi are powered by four 20000 mAh power banks. These power banks contain enough energy to maintain the Raspberry Pi over an extended period of time, in addition to making the demo portable [25].

3.2 Logical Design

Logical design describes where all the non-visible components are located and how they work together. Figure 2 shows a visual of how the components work together in the cluster.

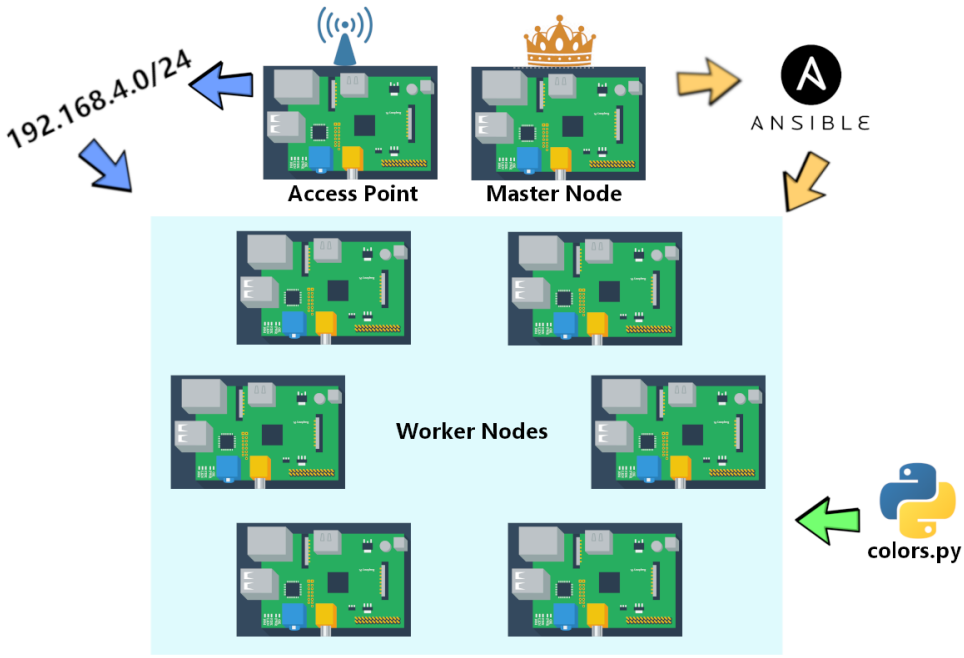


Figure 2: The logical design of the Raspberry Pi cluster.

As seen in figure 2, the master node and the access point node has their own defined jobs in the cluster. The worker nodes are in one group, where they do exactly the same type of job when running. The master node runs Ansible, a domain specific language (DSL) that controls how all the nodes should be configured and keep it that way. The master node also distributes jobs to the worker nodes. The access point node gives each node in the cluster an Internet Protocol (IP) address [26] in the address range of *192.168.4.1* to *192.168.4.254*. All the worker nodes runs a Python script named *colours.py*, which displays the colours on the screen on each worker node.

3.2.1 The network

The Raspberry Pies in the cluster communicates with each other through a closed local network. The closed network is provided by one of the Raspberry Pi acting as an access point. The other Raspberry Pi connects to the access point. They are all to be provided with static IP addresses.

3.2.2 Programmable buttons

Each Raspberry Pi has a screen attached. On the master node, there is five buttons that is programmed to do tasks. When pressing one of the buttons, a predetermined number of jobs is sent to the worker nodes, which in turn starts working. The leftmost buttons sends the minimum number of jobs. The number of jobs is increased for each button rightwards, up to the next to the last button to the right, which sends the maximum number of jobs.

On the worker nodes, the rightmost button is a panic button which kills all the current running jobs. If something has to be done on a worker node, it cannot be done while colours is being displayed, therefore the panic button is there.

The implementation of the programmable buttons can be read about in the chapter about the development process [5.6.5].

3.2.3 Access point

One Raspberry Pi is acting as a lightweight access point. The function on this Raspberry Pi is to keep the network running and to avoid conflict with other configurations. If the network stops working, the entire infrastructure is unable to communicate. This causes the system to fail. However, this Raspberry Pi is also the one responsible for running Ansible for the first-time setup of the cluster.

3.2.4 The cluster

The seven other Raspberry Pi are all part of a Kubernetes cluster, a platform for putting workloads and services into a container [13]. This cluster is automatically set up through Ansible, a configuration manager [1]. Ansible creates a Kubernetes master and six nodes. The cluster also creates a ReplicaSet, which is a keeps track of the number of replicated pods, regardless of nodes [17]. In a pod, there is located one or more containers, which shares an IP address and resources [14].

3.2.5 Master node

One Raspberry Pi is acting as the master node of the Kubernetes cluster. This one is responsible for the configuration of the cluster, as well as the one who scales the number of pods on each worker node. All the configuration files, the Docker image and the Python script to increase CPU load is located on the master node. From the master node, using the programmable buttons, a user can interactively scale up and down the number of pods on the worker nodes, effectively changing the colour displayed.

3.2.6 Worker nodes

The remaining Raspberry Pi are the worker nodes of the cluster. Each worker node runs a Python script named *colours.py*.

3.3 Colours

The use of colours is the most vital part in the demo. Colours is what the user of the demo will see when the buttons are pressed. A colour is displayed on the screen that represents the current CPU usage in percent. The colours go from low CPU usage with the colour blue to high CPU usage with the colour red. There are other colours in between to make the transition from low usage to high usage gradient. According to the illustration of the colour scale in figure 3, the colour will change hue for a set interval based on the increase or the decrease in the CPU usage in percent.

Figure 3: The colour scale used in the demo. The colour ranges from blue (low CPU usage) to red (maximum CPU usage).

The idea behind the colour scheme is to give the user a visual output of how much the Raspberry Pi's are working. The reasoning behind going from blue (cold) to red (warm) is based on to the Kelvin scale.

3.3.1 Kelvin

The Kelvin scale is a thermodynamic temperature scale [27], meaning that this scale measures temparature [28]. The symbol of measurement is K. The Kelvin scale is relevant for the demo because it is also a basis of measuring colour temperature.

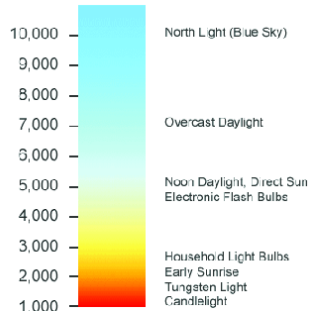


Figure 4: Representation of the Kelvin scale.

[29]

Note that, in accordance with figure 4, the Kelvin scale has similarities with the colour scale used in the demo, except that the colour green is absent from the Kelvin scale.

The Kelvin scale is basically three colours; blue, yellow and red. The colour scale is to represent CPU usage from 0% to 100%. Therefore, a fourth colour, the colour green, was

added to the colour scale in the demo. It gives a greater variety of colour hues. In addition, according to figure 3, the colours fits more naturally when the colour fades from blue into green, rather than directly into the yellow colour.

4 The Main Components of the Design

This chapter provides a description of different tools used as part of the design. It will elaborate on the Rapberry Pi, containerization and subsequent tools, Ansible and Python.

4.1 Raspberry Pi

The Raspberry Pis are the main hardware components of the cluster. A Raspberry Pi, as pictured in figure 5, is a small single-board computer [30]. Raspberry Pi originated in the United Kingdom and are developed by the Raspberry Pi Foundation [31]. Raspberry Pi is the third best-selling computer brand in the world, according to their own FAQ [32]. In 2018 Raspberry Pi sales had reached 19 million [33]. Raspberry Pi are, for the most part, produced either in Wales, China or Japan [34].

Figure 5: A Raspberry Pi 3 Model B+

4.1.1 History

The original Raspberry Pi were developed by the Raspberry Pi foundation as a way to promote computer science teaching at schools in developing countries [35]. However, the original idea exceeded expectations and became more popular than expected, causing it to sell more units than anticipated. Afterwards, the Raspberry Pi foundation split into two arms: *Raspberry Pi Foundation* and *Raspberry Pi Trading*. Raspberry Pi Trading is now responsible for technology and developing, while the Foundation handles the educational and charitable side. On 29 of February 2012, Raspberry Pi Trading released the third and latest model knows as the Raspberry Pi B+. The infrastructure in the demo is built using model 3B+.

4.1.2 Hardware

The Raspberry Pi 3B+ contains a Broadcom BCM2837B0 SoC with a 1.4 GHz 64-bit quad-core ARM Cortex-A53 processor with 512 kb cache [33]. By default, the first model operates at 700 MHz, which is equivalent to 0.041 GFLOPS. The model 3 is described having ten times the performance of the model 1. This is because the model 3 takes into use threading [36] and instruction sets [37]. The Raspberry Pi model 3B+ comes with 1 GB RAM.

The model 3B+ is equipped with a built in Ethernet port through a USB Ethernet adapter using the SMSC LAN9514 chip. In addition, the Raspberry Pi supports wireless on both the 2.4 GHz and 5 GHz Wi-Fi with IEEE 802.11b/g/n/ac [38] and Bluetooth up to 24 Mbit/s. The Raspberry Pi is equipped with four USB inputs, and can be operated with any generic USB I/O device. USB may also be used with storage.

The Raspberry Pi 3B+ can generate video up to a modern TV's resolutions, which means HD and full HD. The CPU is fast enough to allow decoding of H.265-encoded videos. The GPU runs at a clock frequency between 300 MHz and 400 MHz.

All models of the Raspberry Pi contains a heavy degree of thermal management. The Raspberry Pi are easily to overheat under heavy load. They all have a built-in temperature sensor which ensures the Raspberry Pi does not run at a temperature exceeding 85 degrees Celsius. If temperature reaches a critical threshold, the voltage is reduced on the CPU and GPU. This reduces generated heat to control temperature. On the model 3B+ the clock speed is reduced from 1.4GHz to 1.2GHz [39].

From September 2016 and later, a warning will be displayed on the Raspberry Pi if the core temperature runs between 80 and 85 degrees Celsius.

4.1.3 Software

The default operating system provided by The Raspberry Pi Foundation is *Raspbian* [15]. Raspbian is a Debian based Linux distribution free to download. Raspbian uses PIXEL, an abbreviation for *Pi Improved X-window Environment Lightweight* as its desktop environment. The desktop comes with a variety of pre-installed programs, including Mathematica, Chromium and Minecraft. For programming languages, Raspberry Pi promotes using either Python or Scratch. However, it does have support for other programming languages.

Etcher

In order to transfer the Raspbian operating system to the memory cards the Raspberry Pis runs, the program Etcher[40] is used. This program takes the zip file downloaded from the raspbian home page[15] and writes it to the SD card and makes it bootable.

4.2 Containerization

Containers are a way to put an application and all its dependencies and libraries into one package so the code can run quick and reliable in any environment, and not dependent on the operating system. Containers take less space and less resources than a virtual machine, which makes it perfect for a not so resource rich environment as the Raspberry Pi. However, having many containers may require some sort of container management system to keep control over them. The two most popular at the moment is Docker [11] and Kubernetes [13], both free open source software.

4.2.1 The advantage of containers

Before containers, the common way to deploy applications was to install it on a virtual machine (VM) using the operating systems' packet manager. This way the application, it's configuration and executables were entangled with the operating system of the VM. VMs are heavyweight.
A container however, is isolated from the host, with its own filesystem, making them portable across various infrastructures. In addition, container images can be created at build time instead of deployment, meaning a consistent can be carried from development to production.

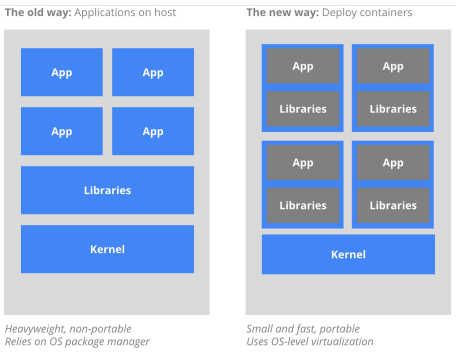


Figure 6: Comparison of running applications directly on the operating system, versus running applications in containers

[13]

4.2.2 Creating containers

The easy way to create a container is to use a Dockerfile. A Dockerfile is essentially a recipe for how to build and create an image of your application. The image can be uploaded to a cloud-based container image storage such as docker-hub, for easy access. The advantage of using a Dockerfile, is that the building process ensures your application always uses the last version available.

4.3 Docker

Docker was created by Solomon Hykes and initially released on March 13, 2013. Docker is described as a tool to create, deploy and maintain containers. By using docker one can avoid creating a whole virtual machine for your application, instead docker allows applications to use the same kernel as the machine they are running on. By using docker, developers can stop worrying about whether their code will run on a system, and instead focus on the code itself.

4.3.1 Dockerfile

A Dockerfile [41] is a file which contains all the necessary commands needed to build a docker image. Docker builds an image by running the commands found in the Dockerfile as command line instructions in succession.

4.3.2 Docker swarm

A Docker swarm is when a group of machines running Docker are joined into a cluster. The cluster is then managed by a swarm manager, which executes the docker commands given.

A swarm can consist of both virtual and physical machines, in the swarm they all become nodes. The swarm manager is the only machine with permission to allow other machines to join as workers. Workers does not have any authority, they are only there to perform the tasks given by the swarm manager.

4.4 Selecting an orchestration tool

In order to automate most of the installation and setup of the project required an orchestration tool to assist in setup. Because most of the cluster needed to be equivalent, except for the access point this would shorten the required first time setup, and ease administration and modification. To find an orchestration tool the following requirements would need to be met:

- Works on Raspbian.
- Not add unnecessary processes to the cluster.
- Be declarative.
- Be idempotent.
- Be able to install Kubernetes.

The first point is obvious, as the cluster will run on Raspberry Pis, the need for the orchestration tool to be compatible with it is vital. In addition, because Raspberry Pis are relatively low power machines (see hardware section at [4.1.2]) and are monitoring load on the cluster, the overhead on each machine should be as low as possible. Being declarative and idempotent was important for much of the same reasons; "Being declarative means you focus on the what instead of the how. An example would be you state that a service should be in a state of running. The imperative alternative is to simply start the service. Both paradigms have their positives and negatives. Imperative systems are often easier to get started with, able to tackle really complex problems, and more efficient. The big advantage of declarative systems is they are generally easier to make idempotent and resilient. In addition, it's perhaps easier to reason about adding additional logic to declarative systems without having to think as much about the big picture" [42]. Also, as the cluster will be running Kubernetes, the ability to install this was also considered important.

4.4.1 Contenders

In the selection process for an orchestration tool for this project, several tools were considered. Following the requirements outlined in Requirements the following orchestration tools were considered:

Chef

Chef is a configuration management tool built around cloud services like Amazon EC2, Google Cloud and Microsoft Azure [43]. Chef also requires an agent to be installed on all clients. Chef recipes are written in Ruby DSL, a language the group has no experience in. Chef is also written as an imperative system, as opposed to both Ansible and Puppet, being declarative systems.

Puppet

Puppet was strongly considered, as the project group was introduced to it during *IMT3005* [44]. Puppet follows a specified schedule for tasks, and does not have a push system for new

changes. This causes a slow iteration.

Puppet also requires an agent installed on all nodes in the cluster, which creates unnecessary overhead with an additional process running. The programming language, Puppet DSL, was also difficult to learn.

4.4.2 Ansible

After considering the contenders Ansible was elected as the orchestration tool to use. "Ansible is an opensource software provisioning, configuration management and application development tool. [...] It includes its own declarative language to describe system configuration" [1]. Ansible is also relatively new, so that the amount of legacy content is small, making it easier to understand and write. This made it the best configuration management tool for this project.

4.5 More about Ansible

4.5.1 History

Ansible was originally developed by Michael DeHaan [45]. Before creating Ansible he had worked, amongst others, for RedHat and Puppet. The word Ansible comes from the fictional work of the author Ursula K. Le Guin in the 1966 novel Rocannon's World. In this book it is a contraction of "answerable", a device that would allow its users to receive answers to messages over interstellar distances with a speed exceeding light speed [46]. Ansible was developed with a "batteries included" philosophy, meaning that everyone can contribute to the main modules, and should, for the most part, be enough for most configuration jobs [45]. Should additional configuration not existing in the main program be necessary there is a vibrant community of users creating their own *roles* (1.8) easily implemented into playbooks. These community made roles is available through the *Ansible Galaxy [47]*. These can be added to Ansible Playbooks (1.8) and immediately put to work.

4.5.2 Methodology

Ansible is agentless, meaning it does not require an installed agent on the machines it controls, it works by temporarily connecting to targets via SSH(for Unix systems) or remote PowerShell(for Windows systems) to do tasks written in YAML using Ansible modules built together in a Ansible Playbook. The only requirement is passwordless SSH access to the hosts in its inventory [48], easily done with public-key cryptography [49], and Python [50] installed on the nodes. The machine must also be specified in the inventory file for the Playbook, either with an IP address or a *hostname* (1.8).

4.5.3 Inventory

The Ansible inventory (1.8) is the list of hosts that Ansible will have access to either by the IP address or the hostname of the nodes. Different hosts can be grouped under several headings so that one group can be specified as a target. An example inventory file can look like this:

```
192.168.6.1

[webservers]
foo.example.com
bar.example.com
```

Listing 4.1: Sample inventory file [1]

This example contains three nodes. The first, specified by an IP address, the other two by hostname and grouped under the `webserver` group [1].

4.5.4 Ansible Galaxy

Ansible Galaxy is a community driven site with pre-packaged units of work known to Ansible as roles. Roles can easily be dropped into Ansible PlayBooks and immediately put to work. You will find roles for provisioning infrastructure, deploying application, and all the tasks you do everyday [47]. Installing a role from Ansible Galaxy is as easy as finding the role you want to install and, on your Ansible machine, writing:

```
ansible-galaxy install <user name>.<role name>
```

Where user name is the user name of the author, and role name is the name of the role. That role will then be available to use in PlayBooks just by specifying that role, for a set of hosts specified in the inventory.

4.6 Kubernetes

Kubernetes [13], often abbreviated as *k8s*, is an open-source container orchestration system developed by Google. Kubernetes can be used for deployment, management and scaling of applications and containers. With Kubernetes, you can cluster together groups of hosts with your containers, and Kubernetes helps you easily manage that cluster.

In a real production environment, application can span outward multiple containers on different hosts. Kubernetes helps by giving you an easy and efficient way to deploy containers, scale, and schedule those containers across a cluster and monitor the health of those containers over time.

4.6.1 What Kubernetes can do

A reason for using Kubernetes in an infrastructure, is the platform it provides for scheduling and running containers on both physical and virtual machines. In addition, Kubernetes is leaned towards automation, which means that a lot of operational tasks other management systems does, Kubernetes can do automatically on containers.
Some more specific examples of what Kubernetes is able to do:

- Manage containers across multiple hosts.
- Control and automate deployment and updates.
- Scale containers dynamically and on the go.
- Mount and add storage to your applications.
- Health monitoring and self-restoration of your apps using automatic placement, restarts, scaling and replication.

4.6.2 What Kubernetes cannot do

While Kubernetes may sound like an all-powerful solution, it has its limits. Kubernetes is not a fully-fledged PaaS (Platform as a Service). While it provides some of the same services as a PaaS, such as scaling, monitoring and load balancing, Kubernetes is not monolithic, and these solutions are optional, not mandatory. This way Kubernetes prioritizes flexibility and

the user's choice.

Hence, Kubernetes does not:

- Deploy your source code and build your application, no form of CI/CD, the images used for containers needs to be pre-built.
- Have built in services such as storage, cache or databases. These services can be run as an application.

4.6.3 Kubernetes Components

Something that's quite handy to know about is Kubernetes master components [51]. These master components are responsible for making global decisions for the cluster, and detecting events happening to the cluster. For example, starting new pods when one unexpectedly crashes; or number of pods don't satisfy the *replica* field.

Kube-apiserver

The kube-apiserver exposes the Kubernetes API, operates and provides the front tend to interact with the cluster.

etcd

Etcd is the place where Kubernetes stores all the data about the cluster.

kube-controller manager

The kube-controller-manager runs on the master and handles all controllers. A controller is a loop that watches the state of the cluster trough the API-server, and will always try to move the cluster's current state toward the desired state.

4.6.4 ReplicaSet

A *ReplicaSet* [17] is the type of Kubernetes configuration which purpose is to maintain a specified number of pods, regardless of nodes. By using a ReplicaSet, it is easy to scale the number of pods up or down by changing the *replicas* field in the configuration file. The ReplicaSet will also migrate pods from one worker node to another if the worker node goes offline. However, the default time for this in Kubernetes is five minutes.

4.7 K3s

K3s is the lightweight Kubernetes distribution developed by Rancher Labs [52]. K3s is resource constraining and easy to install. One of K3s' use cases are ARM processors, which is what the Raspberry Pi has. K3s started as a component of an already existing project by Rancher Labs named *Rio* [53]. Rio was en experiment focused on serverless applications and ran on top of Kubernetes. Rancher Labs were not happy about the memory footprint of running Rio and Kubernetes, so they started looking at what parts of Kubernetes could be removed to make it smaller. Rancher Labs ended up with a open-source, lightweight version of Kubernetes they called K3s.

K3s puts all the Kubernetes components; apiserver, kube-controller-manager and kubelet into a combined process, presented as a server and agent model. All these enhancements makes K3s into a small single binary, about 40 megabytes in size. Inside the binary is everything needed to run a fully fledged Kubernetes cluster. K3s is also really easy to set up and get started with.[54] More of this will be explained later in the process chapter, section 5.6.

4.8 Python

Python is an open-source programming language which is powerful and easy to pick up for both beginner programmers and experienced programmers. The Python language has detailed documentation, found at https://docs.python.org/3/.

Compared to other programming languages, Python code is easy to implement and maintain because this language is a high-level general purpose programming language [39]. This means that Python can be used for a variety of tasks like C, C#, Java, and so on. However, Python has a higher level of abstraction. For instance, in older general purpose languages, algorithms usually needs to be programmed by several lines of logic.

On the other hand, Python has many built-in functions and libraries that already has these algorithms ready to use. There is no need to write most of the algorithms from scratch, since these can just be called on through a library that is included in the script. For instance, listing 4.2 shows how the library *psutil* is used to get the current CPU usage.

```
import psutil
current_cpu_usage = psutil.cpu_percent(interval=1)
```

Listing 4.2: Using the psutil library to get the current CPU usage in Python

In addition to being a high-level programming language, Python supports a socket programming interface. Socket programming lets the Python process on one Rasbperry Pi communicate with other processes on other Raspberry Pi through a network [55].

Based on these functionalities, Python was decided to be the language for programming the colors that is shown on the screen, based on the current CPU usage.

The worker nodes uses the colors.py script, written in Python. Colors.py uses the *psutil* Python library [56] to fetch the current CPU usage in percent and displays a color on the screen.

5 Development Process

This chapter describes the technical process related to developing the demo. A description of the choice of the development model is given first. Then the concept of *infrastructure as code* is discussed. The rest of the chapter is then about the development process of a prototype at different stages.

5.1 Development model

When choosing a development process, there are two main models to choose from: The plan-driven methods and the agile methods [57]. Roughly speaking, plan-driven methods is methods where the product is delivered when the development is finished. In agile methods, the product is updated continuously and new builds arrives at a rapid speed.

The project group chose two main development models for the developing process: The *incremental-sequential model* and the *reuse oriented model*. The incremential-sequential is a general purpose model for the development process in overall. The reuse oriented model is based on that code that is being used in the demo is not necessarily made from scratch - it is no use to write code from the beginning that is already public and free of use. An example of this is the script *term_colours.py* [58] which forms the basis of colours.py used in the demo.

5.1.1 The Incremental-Sequential Model

The incremental-sequential is a mix of both plan-driven methods and agile methods. The demo will not be released to the contractor, but implementations of new features is communicated to the contractor and discussed further with.

5.1.2 The Reuse Oriented Model

The development of the demo is not the first time a cluster of Raspberry Pi that does one function or another is created. There are others who already have done it and may have put the code open for everyone to use as they wish, like the term_colours.py Python script. This relates to both the development of the front-end (colours on the screen, what the audience sees and interacts with) and the development of the back-end (mainly Kubernetes and Ansible, the technologies that automatically sets up the Raspberry Pi cluster).

5.2 Infrastructure as Code

Infrastructure as Code, often abbreviated as IaC, is the IT practice where developers instead of manually configuring hardware and operating systems, automatically manages a technology stack with software instead [59]. IaC can in some ways be compared to scripting, in which it automates a process. However, while scripts generally automate a series of static steps, IaC provides a more adaptive and versatile process. For example, for the process Ansible is used. Ansible as an IaC tool can install a MySQL server, create a user and set up a database on a set of new machines.

5.2.1 Declarative vs Imperative

Using a declarative approach means focusing on the what instead of the how. Instead of describing the commands the computer should run, it outlines the desired state of the finished infrastructure [59]. Both Ansible and Puppet are declarative languages. In contrast to declarative, imperative means defining and listing the steps and commands to reach the final state. The IaC tool Chef [60] can be used as an imperative language.

5.2.2 The usage of IaC

For the demo, when discussed with the contractor, it was mentioned that if the demo feels satisfying, IIK will buy another set of Raspberry Pi for the campus at Trondheim to use. For this reason, the setup would benefit from an automatic setup. It also makes the addition or removal of pods in the cluster easier. Instead of manually setting them up, a IaC tool does it instead automatically.

5.3 First Prototype

The idea behind the first prototype was to establish a connection between all the Raspberry Pies within a closed network. This was done in order to test communicating between the Pies. Since this was only a prototype test, the easy way of communicating were a simple HTTP web server using Apache.

5.3.1 Turning a Raspberry Pi into an access point for a closed network

The idea is to create a closed local network for all the Raspberry Pi to communicate with each other without external interference. The Raspberry pi 3 B+ supports dual-band wireless LAN, meaning both 2,4Gz and 5Gz. Utilizing this, one of them could act as an access point to broadcast an SSID for the others to connect to. In order to use the Pi as an access point, it will need a type of access point software and a DHCP server installed for providing IP addresses to connected devices. For our project the decision settled on Hostapd and dnsmasq.

Hostapd

Hostadp (Host access point daemon) is a software for turning a normal network interface card into an access point (AP) and authentication server. With hostapd the Pi can create an AP to act as a wireless switch. However, it cannot assign IPs to connected devices or itself, therefore we need a DHCP server listening to the AP, providing IP addresses [61].

Dnsmasq

Dnsmasq is a software which provides a DNS (Domain Name System) forwarder and DHCP (Dynamic Host Configuration Protocol) server for the access point. Dnsmasq is labeled as lightweight DNS server and is designed to be easy to configure and suitable for small-scale networks for routers without too much resources. This makes Dnsmasq a perfect option for the small local network, where the access point is a Raspberry Pi with limited CPU and random access memory (RAM). Further dnsmasq supports both dynamic and static IP-address leases, we plan on using the latter for reasons specified later on [62].

Setup

For the actual setup of the access point, The Pi needed to be connected to the internet via Ethernet cable, since it needed to download required files. In addition, setting up the Pi as

a lightweight access point requires interference with already existing Wi-Fi configuration, meaning Wi-Fi would not work normally.

5.3.2 Simple web server

After successfully connecting all the Raspberry Pi together via a closed network, the time had come to set up a very simple web server. Three of the Raspberry Pi with Apache web server was set up.The Raspberry Pi ran a simple PHP Hypertext Processor (PHP) [63] website which only displayed the IP given to the Pi in the closed network.

```
<?php echo "This is raspberry pi worker <IP-address of the Pi> "; ?>
```

After configuring one of the Raspberry Pi with a HAProxy [64] load balancer using all three nodes running apache as a backend. The idea behind this setup, was that when entering the website using the IP of the load balancer, each time would in theory return a different IP address from the previous. This was to see if all the Raspberry Pi were able to connect and communicate with each other within the closed network.
In addition to the load balanced webserver, which proved successful in wireless communication between Raspberry Pi, some sort of way to make the Raspberry Pi itself display some sort of visual cue when it's handling a http request would be required.
The Raspberry Pi would run a simple script written in Golang (Go) [65] as a background service. This script had a *ListenAndServe* function on port 8080, which is the same port used for the Apache web server earlier. A ListenAndServe function is a function which triggers every time traffic is handles trough a certain port, in this case port 8080. This way, whenever a HTTP request is received on port 8080, a separate function could trigger which did something visually. Since this was an early prototype and would most likely be scrapped, the script did not need to be advanced. The final Go script printed a random number with a random colour to the Raspberry Pi attachable screen each time it received a HTTP request.

5.4 Using Docker swarm

With the basic communication between nodes done, there was time to move onto something more advanced for bringing all the Raspberry Pi together into a cluster. The solution requires one of the Raspberry Pies acts as a controller node and the rest acts as workers. The workers would contain a script which constantly display a colour based on the CPU load of the raspberry Pi. This was done using a python script, which can fetch current CPU load from the Pi. The controller would then, in some way, hand out tasks to the worker nodes which increased their CPU load and in turn changed the colours they were displaying. In addition, the controller would have to map the task scheduling to one of the buttons on the attachable screen, making the product interactive for a possible future audience. Therefore, we needed a solution to create a cluster where all this was possible. The logical option was to use Docker Swarm.

Positives of implementing Docker Swarm

- A way to connect all the Raspberry Pis together in a cluster for easy monitoring and maintenance
- The ability to containerize the tasks to increase CPU load, making it easier to scale up

and down to change colours.
- An easy way to deploy services across multiple Raspberry Pis.
- A hierarchy based on a master-slave relationship where a single Raspberry Pi acting as the Docker Swarm master could control the rest of the Raspberry Pi, who would be workers.
- Relatively easy to set up

Before configuring the Docker Swarm, some prerequisites needed to be in order. Firstly, all the Raspberry Pis were given a static IP. This made setting up the swarm a lot easier when we did not have to worry about an IP lease expiring, and the node then getting a different IP address which in turn could result in configuration files being out of date.

Listing 5.1: The Raspberry Pis were given a static IP addresses by appending to the file /etc/dhcpcd.conf

```
interface wlan0
        static ip_address=192.168.4.XX/24
```

Second, the Raspberry Pis needed to set up RSA [66] SSH keys, so the nodes are able to connect and transfer files between each other without requiring user name and password authentication.

The actual setup of the Docker Swarm was simple. On the master the following command was run:

```
$ docker swarm init
```

This creates a Docker Swarm using the current node as manager. In addition, it generates a token for which to use to join worker nodes onto the cluster.

Then, on each of the nodes that is going to be a worker, ran the command:

```
docker swarm join -token <generated token> <ip-address of manager>:<port>
```

For security purposes, it was chosen not to disclose the generated token or IP-address in this report.

When the cluster finished initializing the command:

```
$ docker node ls
```

Showed a working Docker Swarm containing 1 master node and 3 worker nodes.

5.4.1 Moving on from Docker swarm towards Kubernetes

Although Docker offers a simple solution which is also quick to get started with, for our cluster we wanted a more complex solution. Docker worked, and probably would be enough for our simple product, we wanted to explore some different solutions. By using a more complex solution, the cluster could be further automated and easier manageable with regard to solubility and resilience. One of the options we had was Kubernetes. Already from the start we had talked about the possibility of using Kubernetes. We were briefly introduced to Kubernetes in our course Infrastructure as Code, where we learned about the powerhouse, open source, google developed container orchestration tool that is Kubernetes. The thing holding us back was the question, do we really need this much for our simple product? However, in the end after discussing both internally in our group and with our supervisor, we

decided to give Kubernetes a try, to see if it could fit better for our product than docker swarm. Kubernetes offered a wider variety and choice, including a better way to scale the number of jobs using the ReplicaSet (Explained later), a better way to monitor container health and a number of already existing implementation which we could take inspiration from. On the downside Kubernetes required more complex configuration, a lot more researching and a heavier resource load for the Pis.

5.5 Kubernetes

5.5.1 Setting up a Kubernetes cluster

Setting up a Kubernetes cluster is a bit more complicated than configuring a Docker swarm. Some requirements before starting:

- All the Raspberry Pi must be on the same network and must be able to communicate with each other.
- The Raspberry Pi must all have a static IP address to prevent the cluster from breaking because of IP changes. The Kubernetes master's certificate is bound to the IP address, so its best to avoid any troubles.

For the actual setup of the Kubernetes cluster, all nodes need to download Docker. Even though Kubernetes is its own container orchestration tool, its uses docker technologies to handle containers etc.

After installing Docker successfully, swap must be disabled on all the Raspberry Pies. If swap is enabled Kubernetes will give an error. The reason behind this is that with Kubernetes all deployments should be limited to the CPU/memory limits of that node. When the master schedules a pod onto a node, it should not use swap if it oversteps that nodes CPU or memory limits, then you need another node. The code in listing 5.1 describes how swap is disabled on a Raspbian.

```
$ sudo dphys-swapfile swapoff && \
  sudo dphys-swapfile uninstall && \
  sudo update-rc.d dphys-swapfile remove

  cgroup_enable=cpuset cgroup_memory=1
  cgroup_enable=memory >> /boot/cmdline.txt
```

Listing 5.2: Disabling swap on a Raspbian

After disabling swap the Pi could install Kubernetes, which gives access to two new commands; kubeadm and kubectl.

Kubeadm

The command used to create a cluster or join an existing cluster. Kubeadm helps bootstrap a Kubernetes cluster that follows best practice standards. Kubeadm is designed to be a way for beginners to try Kubernetes to stitch together and run a cluster easily.

Kubectl

Kubectl is the command line tool for Kubernetes. It's what's used to run commands against the cluster. Kubectl is used when deploying, inspecting and managing the cluster.

To initialize a cluster the command used is

```
sudo kubeadm init --token-ttl=0 \
--pod-network-cidr=192.168.4.0/24 \
--apiserver-advertise-address=192.168.4.1
```

Listing 5.3: Initializes a new Kubernetes cluster

The argument –token—ttl=0 is passed along to ensure the join token never expires. This is not good practice, however for our project Ansible would later be used for an automatic setup, meaning tokens must not change. In addition, it's somewhat complicated to get a join token after the initial token has expired.

In addition, with another parameter, –apiserver, whose job it is to validate and configure data for the api objects including pods and services. After waiting for the cluster initialization to be complete and, run the command.

```
kubectl get pods --namespace=kube-system
```

Listing 5.4: Shows all pods running the cluster config

To get an output showing pods running the default Kubernetes cluster services.

5.5.2 Kubernetes with Ansible

In order to automate setup of kubernetes, ansible roles were created. The nodes in the cluster were added to the inventory file and grouped as "workcluster" and "master". In the Playbook called "cluster.yml", and shown in listing 5.4, the roles given to each group can be seen.

```
- hosts: all
  roles:
    - common
    - kubeadm

- hosts: master
  roles:
    - master
#    - dashboard

- hosts: all:!master
  roles:
    - workers
```

Listing 5.5: Playbook for rak8s kubernetes roles

Here all hosts will take on the roles 'common' and 'kubeadm'. Master will, in addition to this, take on the role of 'master' and, if not commented out, 'dashboard'. All nodes except master will take on the role of 'worker'. To see what each of these roles do, see addendum B.

This playbook will download, install and setup a kubernetes cluster, with each of the nodes joining the cluster. There is additionally another playbook called "cleanup.yml" created to remove kubernetes from the cluster.

5.5.3 Issues with Kubernetes

Conflicting IP tables when running Kubernetes master on same Raspberry Pi as access point

Both the access point and the Kubernetes cluster required their own set of custom IP-table rules, which when running on the same Raspberry Pi, resulted in unwanted conflicts. As a solution to this problem separating the two services from one another, resulting in one Pi running exclusively as an access point, while another Pi became the Kubernetes cluster master node worked out.

Performance issues

One of the problems with running a Kubernetes cluster on Raspberry Pi is the amount of resources needed while working in such a resource limited environment as the Raspberry Pi provide. One of the main points of our projects was the visual output based on fetching average CPU load. However just by running a Kubernetes cluster, the Raspberry Pi fluctuated between using 60 and 70 percent of max CPU, giving us a very limited amount of space to work with.

5.6 K3s and final prototype

As mentioned our infrastructure had some performance problems, mainly because of the heavy resource requirement for running a Kubernetes cluster. A solution to that problem was an open source lightweight version of Kubernetes named K3s [67]. K3s i built to run on ARM processors, be as resource minimalistic as possible while still providing the necessary tools for a Kubernetes cluster and it is easy to set up and get started with. All of these things combined makes K3s a perfect tool for setting up a cluster on multiple Raspberry Pi.

5.6.1 Setting up K3s cluster

The cluster is supposed to set itself up automatically, with minimal human interaction. The way to achieve this, is to use Ansible as an orchestration tool combined with an open source Ansible playbook made by Github user *Vincent Rabah* named *k3s-ansible* [67]. This playbook sets up a complete k3s cluster with one master and x amount of children. Before running Ansible to set up the cluster all connected nodes must be within the same network, and must all have passwordless SSH access to each other. Reason for this is Ansible uses SSH to run all its commands on remote hosts.

Running the Ansible playbook, Ansible first checks if any version of k3s are currently present on the connected nodes. If yes, Ansible deletes the current version before downloading the latest version available. This is to make sure the when setting up, the cluster always runs the latest version of k3s available. When downloading Ansible checks the current systems CPU version, which on Raspberry Pi would always be armf. Therefore for our project, to both save time and resources, this check was removed and instead armf processor as a hard coded value was inserted.

Further, Ansible divides the configurations into two different roles; master and node. The master role handles the initialization of the cluster, generates and copies the access token and enables the k3s service. This is done by reading the k3s service file for master. The service file is a recipe for the clusters current configurations. Node role configuration is only responsible for copying the k3s service file, and enabling it on all the nodes. In the service file itself, the

nodes retrieves the access token and uses it to join the cluster.

After the playbook has successfully ran its course, the Raspberry Pi should be connected by a k3s cluster. On the master node all normal Kubernetes commands is available for usage, except the flag k3s has to be put in front. example: *kubectl get pods* now becomes *k3s kubectl get pods*.

5.6.2 Implementing the Python script

With a infrastructure now running a k3s cluster containing multiple Raspberry Pi, next step is implementing the python script to get it running on all worker nodes. For the specific cluster, it was decided that containerizing the script and deploying it using Kubernetes/k3s would result in unnecessary work for a process that could be easily solved by just using Ansible to copy and run the script via SSH.

The final solution ended with Ansible fetching the script from a remote Github repository, copying is over to nodes it may concern, and running, with

```
python script.py &
```

Which results in the script running as a background process. This means the node would constantly print colours to the console, but is still able to receive commands over SSH if necessary.

5.6.3 Increasing CPU load

An important part of the cluster is the ability to demonstrate scalability and robustness. Therefore a way to schedule and deploy jobs onto the nodes is a necessity. Math calculations are CPU intensive tasks, therefore a python script was made which increases the CPU workload of the node by checking if a random integer is a prime number, as shown in listing 5.5.

```
01 import time
02 while True:
03     num = 3453452
04     for i in range(2,num):
05         if (num % i) == 0:
06             print(num)
07         if (num % 20) == 0:
08             time.sleep(1)
```

Listing 5.6: Checks if a number is a prime number, runs indefinitely but sleeps 1 second

The script will run indefinitely, while working it uses 100 percent of the Raspberry Pi CPU. Therefore it is programmed to take short breaks in between work periods, resulting in an overall increase of the average CPU load of the Pi. This way colours.py, which fetches the average CPU load, will notice a change, which in turn results in a different colour displayed on the screen.

5.6.4 Containerizing and deploying

The amount of jobs are supposed to be easily scalable both up and down. In addition, if a node collapses, jobs running on that node are supposed to migrate over to a working node to keep the amount of jobs running the same. Kubernetes support both scaling and robustness. To deploy a script on Kubernetes, it need to be put into a container, also known as being containerized [68].

In addition to containerizing the script, it was also uploaded to a docker registry. A docker registry is a repository for Docker images, where users can connect to push and pull images. Docker Hub is the default registry where Docker looks for images, therefore the image containing the script was uploaded to Docker Hub.

Using ReplicaSet

To use the docker image to assign pods to nodes, Kubernetes requires a configuration file (yaml) which describes the metadata about the service. This file requires a field which describes what type of nodes the service is going to run on. Therefore all the nodes in the cluster should manually get assigned a label, describing what type of node it is (master, worker, etc.). The kind of service used for our cluster is a ReplicaSet. Shown in listing 5.6 is an example of a ReplicaSet.

```
01   apiVersion: apps/v1
02   kind: ReplicaSet
03   metadata:
04     name: increaseload
05   spec:
06     replicas: 1
07     selector:
08       matchLabels:
09         name: cpuincrease
10    template:
11     metadata:
12       labels:
13         name: cpuincrease
14     spec:
15      nodeSelector:
16        role: worker
17      containers:
18        - name: increaseload
19          image: celebrian/pingmachine:6
20          imagePullPolicy: IfNotPresent
```

Listing 5.7: A ReplicaSet which would maintain 1 pod using the image from DockerHub

A ReplicaSet has the main purpose to maintain a given number of pods at any time, regardless of number of nodes.

A ReplicaSet contains several fields which describes itself. One of these fields specifies how to identify the nodes who should receive pods based on given label.

The replicas field is the one who decides how many pods should be running at a given time. By changing this number, the ReplicaSet will either create or delete pods until it reaches the desired number.

The container field specifies the name of the containers which will be deployed, in addition to what image they will use.

Also the imagePullPolicy states how often an image will be updated. For this project, since it will be a closed network with no internet connection, the project group did not want Kubernetes to try and pull images except for the first time it is being set up.

5.6.5 Buttons and interaction

On the master node, the one responsible for setting up and scheduling jobs, the buttons were programmed to each scale the number of replicas in the ReplicaSet to either increase or decrease jobs. Further, they each scale the number of jobs up to a certain threshold where the Raspberry Pi start displaying a new colour. With five buttons and five different main colours, a potential user could press buttons to make the Raspberry Pis display a wide variety of colour gradients.

These buttons are all programmed with C, and runs as a background script on the master node. The script is started by the .bashrc file[69], which automatically starts the script when the Pi starts. Ansible is responsible for downloading, copying the script and editing .bashrc.

5.6.6 Security

Security is important, especially because our cluster requires Internet access to download certain configurations. The system will therefore be exposed to the World Wide Web for a short period of time. However, this short period is enough for the Raspberry Pi to get infected, if they do not have the required amount of security. All the Raspberry Pi are password protected. In addition, SSH access to the Raspberry Pi are available only with SSH-keys. Furthermore, to push or pull to the git repository from one of the Raspberry Pi, the user must authenticate using one of the authorized account.

The git repository have no files containing either passwords, keys or other important info which may be exploited. However, if a password is required, it should not be written in plain text, it should be encrypted. The git repository in itself is private. Access can only be given by one of the group members.

6 Testing

This chapter is about testing the different functions of our cluster. Testing is done internally by members of the group. The testing included the core functionality of the cluster specified in the design chapter 3, Scalability and Resilience.

6.1 Scalability

When testing scalability we had a terminal window showing us number of pods currently running, which updated each second. Using different versions of a ReplicaSet and binding them to the different buttons, so each button scaled number of pods exponentially starting at 2. Clicking the first button we saw the ReplicaSet created two pods and assigned them the nodes. Same thing happened for 4, 8 and 16 Pods, however when scaling to 8 and 16 pods, the nodes started to noticeably change colors. As a form of limit testing we tried to scale it to 60 pods, aka 10 pods on each worker node. This resulted in the Pies crashing and rebooting. After several test we came to the conclusion that 4 was the max number of pods which could run comfortably on each node.

6.2 Resilience

The idea was that when a node goes into NotReady status, all pods currently running on that node should migrate to a Ready node instead. To test this, we had two nodes both running 2 pods each. Then we cut power to one of the nodes, expecting the second node to now contain 4 pods instead of 2. The pods migrated over to the healthy node, however it took roughly 10 minutes. The long timer was because of a default variable already set in Kubernetes configuration called pod-eviction-timeout, this problem will be discussed in more detail later in the results chapter.

6.3 User interaction

The interactive part of our product is the 5 buttons on each Pi. However we only use the buttons on the manager node for interactability. Each of the buttons on the master node is linked to a different version of a ReplicaSet, which scales number of pods up or down. In addition, when pressing a button, an output is shown in the terminal window telling the user how many pods the cluster is currently scaled to. When pressing button 2, the terminal shows the message "Number of pods scaled to 6". The number of pods are increased to 6 and the color on the worker nodes slightly changes. This effect is the same for each of the button except the number changes from 6 to 12, 18 and 24. However one thing which disturbs the visual part, is that if a Raspberry Pi runs at high capacity it may display a message to the terminal saying "Low power input". This message overwrites the existing ones and makes it hard to see the message displayed when pressing a button.

7 Results and Discussion

This chapter presents the final result of our work process, the cluster both physically and logical. The chapter also contains discussion about how the result corresponded with initial requirements, challenges along the way and how things could have been done better. By using the process described in earlier chapters, the project group developed a final product which both the group member and supervisor were content with. There are some aspects the project group did not get enough time to finish, and some things that should have been done different. Overall, the project group is satisfied with how the final product turned out.

7.1 Final product

The final cluster consists of eight Raspberry Pi. Each of the Raspberry Pis has their own attachable hat with a 5' screen which displays a terminal window. The hat also has five programmable buttons. The Raspberry Pis are all powered with power banks. All communication between the Raspberry Pis are within a closed network, which is broadcast from one of them, making the solution closed of from the Internet and therefore portable. The Raspberry Pi are all a part of a Kubernetes cluster, using a lightweight version of Kubernetes called K3s. The cluster consists of one master node and six worker nodes. The K3s cluster sets itself up automatically using the deployment tool Ansible.

On each worker node, there is a Python script which every 0.5 seconds utilizes the script to fetch the current CPU load of the node. Based on this, the script prints out a series of colourized squares onto the terminal window. These squares change colours based on the current CPU usage. If the CPU load increases, the squares will turn to a warmer colour temperature. If the CPU load decreases, the squares will turn to a colder colour temperature. This is done to give a visual interpretation of the CPU changes its workload, and thus the traffic is scaling up or down.

To scale up and down the amount of workload on the CPU, another Python script is deployed onto containers and is scheduled with the Kubernetes' ReplicaSets method onto the worker nodes. Using the programmable buttons on the master node, the product is interactive. The user can press any of the buttons to either create or delete the number of containers, hence changing the colours displayed on the Pies. In addition, if a node either loses power, or is manually plugged out, any pods running on that current worker node will migrate over to a healthy worker node after a set time, which Kubernetes defaults to five minutes.

Figure 7 shows an image of the final Physical cluster including all 8 Raspberry Pies. 6 of the Pies are running the colours python script, and is therefore printing a blue colour onto the screen. The remaining two are the access point and the master node. All of the Pies are powered by powerbanks, each powerbank can supply two Pies with enough power.

Figure 7: The finished demo with the physical and the logical design implemented

7.1.1 Fulfilling requirements

The task requirements given were to develop a prototype which could demonstrate what students studying IT-Operations work with on a daily basis. In addition, it is supposed to be something cool which would attract attention when displayed at a public exhibition. The final product can demonstrate both scalability and resilience using a variety of infrastructure as code tools and scripting, which is something a student studying IT-Operations at NTNU eventually will become familiar with. The visual display of colours combined with the interactive part, makes it attract attention and perhaps steer someones interest towards studying IT-Operations.

7.2 Challenges

Even though the project group is satisfied with how the final product turned out, there are some challenges along the way which should have been handled differently. There are also some decisions which could have been made better to improve the overall quality of the final product.

7.2.1 Physical setup and Portability

The contractor had a wish, if we had enough time, that the final product could be contained as a portable setup within a briefcase or something similar. This way, it would be easy to travel with and it would also add to the cool-factor of the presentation aspect. Unfortunately, the required time nor financial resources were not enough to implement it. However, it could be a task for a future bachelor's thesis.

Overall, the physical setup of the product is rather a prototype than a finished product. During the development process, the project group instead prioritized the finalization of getting the logical system working, which resulted in minimal time to focus on the physical design. In addition, the attachable hats that were bought for the Raspberry Pi provided to be a challenge, since they have no official way of putting multiple Raspberry Pi together like a full sized Raspberry Pi case normally has. [70].

7.2.2 Pod-eviction-timeout

The *pod-eviction-timeout* was a problem the project group ran into when demonstrating the resilience of the product. In Kubernetes, the kube-controller-manager component [71] contains a variable which decides how long time it takes before a pod migrates from a dead worker node onto a new worker node. By default, this variable is set to five minutes. That is not ideal for a live demonstration.

Normally, one would be able to change this pod-eviction-timeout variable by editing the configuration for kube-controller-manager, since it is also a container running an image. However, in K3s, which is what the cluster is using, all Kubernetes cluster data are bundled into a sqlite databse instead of the normal etcd. This makes changing the specific variable significantly harder.

In K3s, it is possible to customize components by adding argument when initiating the cluster:

```
k3s server --kube-controller-arg \
--pod-eviction-timeout=10s
```

There were attempts to change the timeout to ten seconds instead of five minutes. However, the initializing of the cluster is done automatically using Ansible instead of doing it manually. This led to the initialization step failed because Ansible did not recognize the syntax as correct. A way of fixing this problem has not been found.

7.2.3 Network connectivity problems

One of the more serious problems encountered was the one where the cluster is reliant on the Internet connection to be able to communicate. The problem occurs because in the initial setup of the cluster, all the nodes, including the master node, is required Internet connection to download the needed files for the first time setup. All the nodes in the cluster is supposed to be assigned a 192.168.4.X internal IP address. However, when connected to the Internet,

the Raspberry Pi are assigned another IP address for the eth0 interface. This is in addition to the statically set IP address on the wlan0 interface, which is an IP address from the closed network. The problem then occurs when a Raspberry Pi tries to join the cluster as a node. The cluster sometimes uses the IP address from interface eth0 instead of wla0, resulting in the node getting a 10.10.0.X internal IP address in the cluster. When a node is restarted, it tries to communicate over the 10.10.0.X network, which no longer exists when the cluster is disconnected from the Internet. The node goes into *NotReady* mode. A way to fix it is to always have the Raspberry Pi connected to the Internet while it is booting up. This, of course, requires seven Ethernet ports available. Another solution is to have a switch which all the Raspberry Pi connects to. This would affect the overall portability of the product, because the switch also requires power and it would also need an Internet connection.

7.2.4 Refactoring

Both Ansible and K3s are open source software which is continuously updated. However, the system is built to be offline. While it is possible to update it to the latest version after a while, there cannot be any guarantees that everything will still work as it should. Should the cluster eventually be reconnected to the internet and the software becomes update, there is no guarantee the current implementation will work with future versions of both K3s nor Ansible. F

8 Ending

After working with the Raspberry Pi and trying different tools and technologies, the final product ended up with an implementation which satisfied both the group and the contractor. Working with Raspberry Pi is interesting, but also frustrating at times. The Raspberry Pi's limited resources makes them a bottleneck in certain situations.

Different configuration management systems were discussed, with the project group in accordance with the supervisor agreeing on using Ansible. Ansible is faster and requires less resources, which is already sparse on the Raspberry Pi.

Based on this, the final product is a mix of Ansible and Kubernetes used to set up a cluster which makes the Raspberry Pi change colors based on resource usage. The product has a simple to use interactive front-end which may attract attention from potential students interested in IT-operations at NTNU. In addition, for those already more knowledgeable in the art of IT-operations, the product contains an interesting back-end by mixing together different technologies to demonstrate scalability and resilience on a Raspberry Pi cluster.

8.1 Further work

With the prototype as a basis, several suggestions arose as to which changes to implement that could improve the overall quality of the product.

Network dependency

The infrastructure as a whole is supposed to be a closed network after first time configurations. This is to make the product portable and easy to bring with to show off at exhibitions. However, the nodes in the cluster requires and internet connection every time they are booted which makes the closed network redundant. Making the cluster fully independent from the Internet would greatly increase portability.

Containerize the color script

The Python script which displays colors on the screen is currently running as a background process on each worker node. The script is downloaded from a git repository and copied onto the nodes through Ansible. Containerizing the script and having it running as a container on each worker node would decrease resource usage, because Kubernetes when deploying containers limits the amount of resources available for the container. This would make it easier to scale further, should the number of worker nodes in the cluster increase over time.

Colorblind mode

Color blindness is a deficiency related to how the human eye perceives colors. With color blindness, the human eye cannot differentiate certain colors. The most common color blindness is the red-green color blindness, in which the human eye cannot see the difference between the colors red and green [72]. Color blindness affects about one in twelve men and one in two-hundred women worldwide [73]. When the demo is showcased to a variety of audiences, there is likely that there will be members of the audience who has color blindness.

A colorblind mode should be implemented, so that those who has color blindness also will enjoy the demo.

Further automate cluster setup

As from now, the cluster still requires a certain amount of manual configuration on each Raspberry Pi before running the Ansible playbook. These manual configurations, such as setting up the access point, connecting each pi to the network, setting static IP address and adding hostnames into the Ansible playbook, could be automated to save time.

Variable dependent number of jobs

For scaling up and down the amount of pods running on the nodes, the current setup has one ReplicaSet.yaml file linked to each button. The values in the files are hard coded to 6,12,18 and 24. Changing it to instead of having multiple files, it could be one file which takes a command line argument. For example: *(number of pods * button number)*. This would make button number 4 on a cluster with 6 nodes scale number of pods to $6 * 4 = 24$ pods. This solution is more scalable when increasing or decreasing the number of worker nodes available.

pod-eviction-timeout

The default timer of how long it takes a pod to migrate from a dead worker node onto a healthy one is five minutes. This variable is set by Kubernetes, but it can be changed. The tricky part is changing it while setting up K3s with Ansible, which does not use default commands for the cluster setup. The project group tried to find a way to change the timer, but to no luck. The five minutes is unnecessary long and not optimal for a live demonstration. Finding a way to change this times would improve the overall reception of the product.

Configure network settings while flashing Raspbian image

It would save significant amount of time and work if the Raspberry Pi would enable and connect to the Wi-Fi while flashing the Raspbian image. This is something that usually has to be done manually. However, it should be able to automate.

Increase scalability to the cloud

A great way to further demonstrate scalability would be to demonstrate that if, for example, the load on the cluster exceeded 90% and the cluster had an internet connection it could scale up to a cloud provider, so that in the event of several nodes going down, the work would be offloaded to the cloud. This would also demonstrate that even if the cluster ran out of battery, it would not completely fail.

8.1.1 Evaluation of the task

"The machine that goes ping" was the first task of choice for our group for several reasons. On one hand, the task offered creative freedom and an opportunity to work with something practical alongside the theoretical part of a bachelor's thesis. On the other hand, the freedom also meant the project group had to figure everything out on their own. There were no clear boundaries of what was supposed to work with, which made time management a critical factor. Overall the group is very happy with the task, it provided a fun and challenging way to work with exiting new technologies. In addition, it gave us a way to work with both a practical and a theoretical part, giving us a more real-life experience in comparison to working with a

strictly theoretical task.

In the task description given to the project group, it states that there is a need to showcase that digital infrastructure, cloud solutions and operative cybersecurity is exciting and cool.

To decide what is cool and not is a challenging task. "Cool" is very much a subjective term that can be interpreted differently from one person to another.

In hindsight, the project group is satisfied with the task, and by no means have any regrets choosing it. However, the project group underestimated the amount of time and the work it takes to learn a completely new technology.

Given that the only prior experience the project group had with Kubernetes was copying some commands from a PDF-document, a lot of time went into reading and researching new technologies. This resulted in less time for implementation, and the final product did not really end in the state we had hoped for when starting out. Using a mixture of new and already known technologies would have resulted in an easier learning curve and a more efficient work process.

8.1.2 Learning outcome

Generally working with a bachelors

Working with a bachelor's thesis provided us with firsthand experience with team work, time management, stress and dealing with an "employer" which gave us a task to full fill.

Raspberry Pis

Working with Raspberry Pies the group learned a lot about working on a platform which provides limited resources. There were times when the Pies would simply overheat or crash because of the heavy work load we performed on them. We learned about open-source software, since the Raspberry Pi community heavily consists of being open source [74]. Also working with a physical product instead of just virtualization taught us the importance of resource handling and overall being cautious. Unlike in OpenStack, [75] which the project group worked with before, a virtual machine that broke could just be deleted and then create a new one right away. If one of the Raspberry Pi will break, it will not be that easy to replace them instantly.

Ansible

Working with Ansible is great as an introduction to configuration management. Because Ansible uses python scripts hidden away under roles, it is easy to get started. The relative ease to find and install Ansible Roles from Ansible Galaxy, there is no real need to know python scripting yourself, as long as what you are trying to do has not been done before. The format of the inventory and playbook files, written in YML, is easy to understand and get started with. It is only when it is needed to write implementation specific tasks or roles that a deeper understanding is required. There are, naturally, some security concerns with using roles and playbooks taken from the internet without checking all tasks in them to see what they actually do. Overall, Ansible seems like an incredibly versatile and good tool to have experience in, and should be highly useful for all people who want a career in IT operations.

Kubernetes

While we had already some prior experience working with containers , and had already tried some of Kubernetes beforehand, the vast amount of options provided by Kubernetes were

at first overwhelming. Using Kubernetes to create and manage a cluster was a double-edged sword of sorts. Kubernetes provided us with a very simple and easy way to get started, a few simple commands, and we had a working cluster. However when something went wrong, and we had to troubleshoot, we quickly found out how comprehensive Kubernetes is. Overall working with Kubernetes was a pleasant experience which gave us a lot of knowledge about containerization and container orchestration. However because of the small time frame we feel like we did not utilize Kubernetes to its full extent.

Python

Python proved to be the best scripting language for colors.py in terms of simplicity. Colors.py is a mix of reused-code from an open-source project [76]. This leads the script to be 125 lines of code, which may, for some, seem overwhelming when the script is for displaying colors. It is for certain that the code could have been refactored into lesser, minimalist code. However, the code is thoroughly commented. The most important thing to think about, when tuning this code, is to change the *BASE* and *INTERVAL* variables to suit the CPU. One other thing is that Python lacks a traditional switch statement. This was instead solved as several lines of if statements. It may not be best practice, but it works certainly for the colors.py script.

8.1.3 Conclusion

If NTNU wants a physical product they can show off at exhibitions, the prototype accompanying this report, while relatively simple in usage, is both visually interesting and interactive. With some improvements the prototype could be further developed into a final product which NTNU could use as a method of attraction when attending public events.

Bibliography

[1] Wikipedia Contributors. *Ansible (software)*. [25.04.2019]. https://en.wikipedia.org/wiki/Ansible_(software).

[2] NTNU. *Department of Information Security and Communication Technology*. [06.05.2019]. https://www.ntnu.edu/iik.

[3] NTNU. *IMT2008 - ITSM, Security and Risk Management*. [24.04.2019]. https://www.ntnu.edu/studies/courses/IMT2008/2018/1#tab=omEmnet.

[4] NTNU. *IMT4129 - Risk Management for Information Security*. [24.04.2019]. https://www.ntnu.edu/studies/courses/IMT4129/2018/1#tab=omEmnet.

[5] NTNU. *IMT3004 - Incident Response, Ethical Hacking and Forensics*. [24.04.2019]. https://www.ntnu.edu/studies/courses/IMT3004/2018/1#tab=omEmnet.

[6] NTNU. *IMT4115 - Introduction to Information Security Management*. [24.04.2019]. https://www.ntnu.edu/studies/courses/IMT4115/2018/1#tab=omEmnet.

[7] NTNU. *IMT1003 - Introduction to IT-Operations and Information Security*. [24.04.2019]. https://www.ntnu.edu/studies/courses/IMT1003/2018/1#tab=omEmnet.

[8] NTNU. *NTNU Open*. [20.04.2019]. https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/223328.

[9] Apache. *Apache*. [15.5.2019]. https://httpd.apache.org/.

[10] Hackernoon. *Containerization*. [15.5.2019]. https://hackernoon.com/what-is-containerization-83ae53a709a6.

[11] Docker Inc. *Docker*. [15.5.2019]. https://www.docker.com/.

[12] Techopedia.com. *Domain-Specific Language (DSL)*. [15.05.2019]. https://www.techopedia.com/definition/18952/domain-specific-language-dsl.

[13] The Kubernetes Authors. *What is Kubernetes*. [23.04.2019]. https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/.

[14] The Kubernetes Authors. *Pods*. [16.05.2019]. https://kubernetes.io/docs/concepts/workloads/pods/pod/.

[15] raspbian.org. *Raspbian*. [05.05.2019]. https://www.raspbian.org/.

[16] Beau Hamilton. *Raspberry Pi Sales*. [15.5.2019]. https://hardware.slashdot.org/story/17/03/17/2032225/raspberry-pi-becomes-third-best-selling-general-purpose-computer-of-all-time-beati

[17] The Kubernetes Authors. *ReplicaSet*. [16.05.2019]. https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/.

[18] Wikipedia. *SSH*. [15.5.2019]. https://en.wikipedia.org/wiki/Secure_Shell.

[19] Antony Steed and Manuel Fradinho Oliveira. *Chapter 12 - Scalability*. [15.05.2019]. https://www.sciencedirect.com/science/article/pii/B9780123744234000124.

[20] Adrian Hornsby. *Patterns for Resilient Architecture - Part 1*. [15.05.2019]. https://medium.com/@adhorn/patterns-for-resilient-architecture-part-1-d3b60cd8d2b6.

[21] Kahoot. *Kahoot!*. [16.05.2019]. https://kahoot.com/.

[22] TechTerms.com. *CPU*. [15.05.2019]. https://techterms.com/definition/cpu.

[23] no.farnell.com. *RPI3-MODBP - Single Board Computer, Raspberry Pi 3 Model B+, BCM2837B0 SoC, IoT, PoE Enabled*. [15.05.2019]. https://no.farnell.com/raspberry-pi/rpi3-modbp/sbc-board-raspberry-pi-3-model/dp/2842228.

[24] 4D Systems. *2.4" Primary Display HAT for the Raspberry Pi*. [15.05.2019]. https://www.4dsystems.com.au/product/4DPi_24_HAT/.

[25] multicom.no. *Trust URBAN PRIMO - Strømbank - 20000*. [15.05.2019]. https://www.multicom.no/trust-urban-primo-strombank-20000/cat-p/c/p9560099.

[26] TechTerms.com. *IP Address*. [15.05.2019]. https://techterms.com/definition/ip_address.

[27] Wikipedia Contributors. *Kelvin*. [03.05.2019]. https://en.wikipedia.org/wiki/Kelvin.

[28] Wikipedia Contributors. *Thermodynamic temperature*. [03.05.2019]. https://en.wikipedia.org/wiki/Thermodynamic_temperature.

[29] Uploaded by Beata Stahre Wästberg. *Colour Temperatures in the Kelvin scale*. [03.05.2019]. https://www.researchgate.net/figure/Colour-Temperatures-in-the-Kelvin-scale-Image-courtesy-of-wwwmediacollegecom_fig4_236605608.

[30] Wikipedia Contributors. *Raspberry Pi*. [05.05.2019]. https://en.wikipedia.org/wiki/Raspberry_Pi.

[31] Raspberry Pi Foundation. *Raspberry Pi — Teach, Learn, and Make with Raspberry Pi*. [05.05.2019]. https://www.raspberrypi.org/.

[32] Raspberry Pi Foundation with community contributions. *FAQs - Raspberry Pi Documentation*. [05.05.2019]. https://www.raspberrypi.org/documentation/faqs/#introduction.

[33] Eben Upton. *Raspberry Pi 3 Model B+ on sale now at $35*. [05.05.2019]. https://www.raspberrypi.org/blog/raspberry-pi-3-model-bplus-sale-now-35/.

[34] Liam Tung. *Raspberry Pi: 14 million sold, 10 million made in the UK)* . [16.05.2019]. https://www.zdnet.com/article/14-million-raspberry-pis-sold-10-million-made-in-the-uk/.

[35] BBC UK. *Can a £15 computer solve the programming gap?* [05.05.2019]. http://news.bbc.co.uk/2/hi/programmes/click_online/9504208.stm.

[36] Wikipedia Contributors. *Thread (computing)*. [05.05.2019]. https://en.wikipedia.org/wiki/Thread_(computing).

[37] Wikipedia Contributors. *Instruction set architecture*. [05.05.2019]. https://en.wikipedia.org/wiki/Instruction_set_architecture.

[38] Wikipedia Contributors. *IEEE 802.11ac*. [05.05.2019]. https://en.wikipedia.org/wiki/IEEE_802.11ac.

[39] Raspberry Pi Foundation. *Frequency management and thermal control*. [05.05.2019]. https://www.raspberrypi.org/documentation/hardware/raspberrypi/frequency-management.md.

[40] Balena. *Etcher*. [18.5.2019]. https://www.balena.io/etcher/.

[41] Docker Imc. *Dockerfile*. [18.5.2019]. https://docs.docker.com/engine/reference/builder/.

[42] Dan Mcpherson. *Why openshift picked ansible*. [25.04.2019]. https://blog.openshift.com/why-openshift-picked-ansible/.

[43]

[44] NTNU. *IMT3005 - Infrastructure as Code*. [30.04.2019]. https://www.ntnu.edu/studies/courses/IMT3005#tab=omEmnet.

[45] Michael DeHaan. *The origins of Ansible*. [25.04.2019]. https://www.ansible.com/blog/2013/12/08/the-origins-of-ansible.

[46] Wikipedia Contributors. *Ansible, the word*. [26.04.2019]. https://en.wikipedia.org/wiki/Ansible.

[47] Ansible Team. *Ansible Galaxy*. [26.04.2019]. https://en.wikipedia.org/wiki/Ansible.

[48] Ansible Team. *How Ansible Works*. [26.04.2019]. https://www.ansible.com/overview/how-ansible-works.

[49] Wikipedia Contributors. *Public-key Cryptography*. [26.04.2019]. https://en.wikipedia.org/wiki/Public-key_cryptography.

[50] Wikipedia Contributors. *Python (programming language)*. [05.05.2019]. https://en.wikipedia.org/wiki/Python_(programming_language).

[51] The Kubernetes Authors. *Kubernetes Components*. [16.05.2019]. https://kubernetes.io/docs/concepts/overview/components/.

[52] Rancher Labs INC. *k3s - Lightweight Kubernetes*. [16.05.2019]. https://k3s.io/.

[53] Rancher Labs INC. *Rio*. [16.05.2019]. https://github.com/rancher/rio.

[54] Rancher Labs INC. *k3s - 5 less than k8s*. [16.05.2019]. https://github.com/rancher/k3s/blob/master/README.md.

[55] IBM Corporation. *Socket programming*. [05.05.2019]. https://www.ibm.com/support/knowledgecenter/en/ssw_ibm_i_72/rzab6/rzab6soxoverview.htm.

[56] giampaolo.rodola billiejoex. *psutil*. [25.04.2019]. https://pypi.org/project/psutil/.

[57] Ian Sommerville. *Software Engineering Tenth Edition*. Pearson, 2016.

[58] Christian Oudard. *Terminal output colors in python*. [15.05.2019]. https://gist.github.com/christian-oudard/220521.

[59] Margaret Rouse. *Infrastructure as code*. [16.05.2019]. https://searchitoperations.techtarget.com/definition/Infrastructure-as-Code-IAC.

[60] The Chef Authors. *Chef*. [16.05.2019]. https://www.chef.io/.

[61] Gentoo Linux Contributors. *Hostapd*. [24.04.2019]. https://wiki.gentoo.org/wiki/Hostapd.

[62] Archlinux Contributors. *dnsmasq*. [24.04.2019]. https://wiki.archlinux.org/index.php/Dnsmasq.

[63] The PHP Group. *What is PHP?*. [16.05.2019]. https://www.php.net/manual/en/intro-whatis.php.

[64] haproxy.org. *HAProxy - The Reliable, High Performance TCP/HTTP Load Balancer*. [16.05.2019]. http://www.haproxy.org.

[65] golang.org. *The Go Programming Language* . [16.05.2019]. https://golang.org/.

[66] Margaret Rouse. *RSA algorithm (Rivest-Shamir-Adleman)* . [16.05.2019]. https://searchsecurity.techtarget.com/definition/RSA.

[67] Vincent Rabah. *k3s-ansible*. [25.04.2019]. https://github.com/itwars/k3s-ansible.

[68] Runnable. *Dockerize your Python Application*. [25.04.2019]. https://runnable.com/docker/python/dockerize-your-python-application.

[69] Alexander Fox. *.bashrc*. [18.5.2019]. https://www.maketecheasier.com/what-is-bashrc/.

[70] geekworm.com. *Raspberry Pi Cluster Case with Fan Kit | 1- 5 layer Multi Layer Acrylic Case* . [16.05.2019]. https://geekworm.com/products/raspberry-pi-cluster-case-with-fan-kit-1-5-layer-multi-layer-acrylic-case.

[71] The Kubernetes Authors. *kube-controller-manager* . [16.05.2019]. https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/.

[72] The National Eye Institute. *Facts About Color Blindness* . [16.05.2019]. https://nei.nih.gov/health/color_blindness/facts_about.

[73] colourblindawareness.org. *Colour Blindness* . [16.05.2019]. http://www.colourblindawareness.org/colour-blindness/.

[74] opensource.com. *What is a Raspberry Pi?* . [16.05.2019]. https://opensource.com/resources/raspberry-pi.

[75] openstack.org. *Openstack* . [16.05.2019]. https://www.openstack.org/.

[76] Christian Oudard. *Terminal output colors in python*. [15.05.2019]. https://gist.github.com/christian-oudard/220521.

A Scripts

A.1 Scripts

```
#!/usr/bin/python

from __future__ import print_function
import SocketServer
import socket
import os
import psutil
from BaseHTTPServer import BaseHTTPRequestHandler


colors = ['\33[106m', '\33[102m', '\33[101m', '\33[106m']


LOW_LOAD = 35
MID_LOAD = 70

RESET  = '\x1b[0m'
PORT   = 8080
def rgb(red, green, blue):
    """
    Calculate the palette index of a color in the 6x6x6 color cube.
    The red, green and blue arguments may range from 0 to 5.
    """
    return 16 + (red * 36) + (green * 6) + blue

def print_color():
#  for color in colors:
#    print(color + "....................................." + RESET)
    print('RGB color cube, 6x6x6:')
    for green in range(6):
        for red in range(6):
            for blue in range(6):
                print_color('  ', bg=rgb(red, green, blue), end='')
            print(' ', end='')
        print()
    print()

class MyHandler(BaseHTTPRequestHandler):
  def do_GET(self):
    if self.path == '/test':
      print_color()

print("test")
print_color()
print("done")
```

```
#!/usr/bin/python

import socket
import SimpleHTTPServer
import SocketServer

PORT = 8080

def print_color():
  color = '\x1b[6;30;42m'
  reset = '\x1b[0m'
  print (color + "....................................." + reset)


class MyTCPServer(SocketServer.TCPServer):
  def server_bind(self):
    self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    self.socket.bind(self.server_address)
  def do_GET(self):
    if self.path == '/test':
      print_color()

Handler = SimpleHTTPServer.SimpleHTTPRequestHandler

httpd = MyTCPServer(("", PORT), Handler)

httpd.serve_forever()
```

```
#!/usr/bin/python

"""
colors.py is a python script that returns a gradient color-based output
based on the current CPU-usage.

Thanks to https://gist.github.com/christian-oudard/220521 for inspiration
and
for providing the base code.
"""

from __future__ import print_function
from BaseHTTPServer import BaseHTTPRequestHandler
import os
import psutil
import socket
import SocketServer
import time

PORT   = 8080

def rgb(red, green, blue):
    """
    Calculate the palette index of a color.
    The red, green and blue arguments may range from 0 to 5.
    """
    return 16 + (red * 36) + (green * 6) + blue

def set_color(fg=None, bg=None):
    """
    Print escape codes to set the terminal color.
    fg and bg are indices into the color palette for the foreground and
    background colors.
    """
    print(_set_color(fg, bg), end='')

def _set_color(fg=None, bg=None):
    result = ''
    if fg:
        result += '\x1b[38;5;%dm' % fg
    if bg:
        result += '\x1b[48;5;%dm' % bg
    return result

def reset_color():
    """
    Reset terminal color to default.
    """
    print(_reset_color(), end='')

def _reset_color():
    return '\x1b[0m'

def get_color(*args, **kwargs):
    """
    Print function, with extra arguments fg and bg to set colors.
    """
    fg = kwargs.pop('fg', None)
    bg = kwargs.pop('bg', None)
    set_color(fg, bg)
    print(*args, **kwargs)
```

```
    reset_color()

def get_cpu_usage():
   """
   Returns the CPU usage using the psutil library.
   Note that the psutil is not installed by default.
   Psutil can be installed using pip / apt.
   """
   return psutil.cpu_percent(interval=1)

def print_color():
  """
  Prints a color to screen based on CPU usage.
  The colors are gradient and will blend into one another when the CPU goes
    up and down.
  White = No load.
  Blue = Low load.
  Green = Some load.
  Yellow = Medium to High load.
  Red = Very high load.

  BASE is the percent at which the CPU is at its most idle (e.g., BASE can
be
  'idle' if the CPU is 50% when running k8s).

  INTERVAL is how many percent should increase / decrease before changing
color.

  """
  BASE = 5
  INTERVAL = 5
  CPU = get_cpu_usage()
  #CPU = 55
  if (CPU <= BASE): get_color('  '*1500, bg=rgb(5, 0, 35), end='')
  if (CPU > BASE and CPU <= (BASE+(INTERVAL*1))): get_color('  '*1500,
bg=rgb(4, 0, 35), end='')
  if (CPU > BASE+(INTERVAL*1) and CPU <= BASE+(INTERVAL*2)): get_color('
'*1500, bg=rgb(3, 0, 35), end='')
  if (CPU > BASE+(INTERVAL*2) and CPU <= BASE+(INTERVAL*3)): get_color('
'*1500, bg=rgb(2, 0, 35), end='')
  if (CPU > BASE+(INTERVAL*3) and CPU <= BASE+(INTERVAL*4)): get_color('
'*1500, bg=rgb(1, 0, 35), end='')
  if (CPU > BASE+(INTERVAL*4) and CPU <= BASE+(INTERVAL*5)): get_color('
'*1500, bg=rgb(0, 0, 35), end='')
  if (CPU > BASE+(INTERVAL*5) and CPU <= BASE+(INTERVAL*6)): get_color('
'*1500, bg=rgb(0, 6, 35), end='')
  if (CPU > BASE+(INTERVAL*6) and CPU <= BASE+(INTERVAL*7)): get_color('
'*1500, bg=rgb(0, 6, 34), end='')
  if (CPU > BASE+(INTERVAL*7) and CPU <= BASE+(INTERVAL*8)): get_color('
'*1500, bg=rgb(0, 6, 33), end='')
  if (CPU > BASE+(INTERVAL*8) and CPU <= BASE+(INTERVAL*9)): get_color('
'*1500, bg=rgb(0, 6, 32), end='')
  if (CPU > BASE+(INTERVAL*9) and CPU <= BASE+(INTERVAL*10)): get_color('
'*1500, bg=rgb(0, 6, 31), end='')
  if (CPU > BASE+(INTERVAL*10) and CPU <= BASE+(INTERVAL*11)): get_color('
'*1500, bg=rgb(0, 6, 30), end='')
  if (CPU > BASE+(INTERVAL*11) and CPU <= BASE+(INTERVAL*12)): get_color('
'*1500, bg=rgb(1, 6, 30), end='')
  if (CPU > BASE+(INTERVAL*12) and CPU <= BASE+(INTERVAL*13)): get_color('
'*1500, bg=rgb(2, 6, 30), end='')
```

```
  if (CPU > BASE+(INTERVAL*13) and CPU <= BASE+(INTERVAL*14)): get_color('
'*1500, bg=rgb(3, 6, 30), end='')
  if (CPU > BASE+(INTERVAL*14) and CPU <= BASE+(INTERVAL*15)): get_color('
'*1500, bg=rgb(4, 6, 30), end='')
  if (CPU > BASE+(INTERVAL*15) and CPU <= BASE+(INTERVAL*16)): get_color('
'*1500, bg=rgb(4, 5, 30), end='')
  if (CPU > BASE+(INTERVAL*16) and CPU <= BASE+(INTERVAL*17)): get_color('
'*1500, bg=rgb(4, 4, 30), end='')
  if (CPU > BASE+(INTERVAL*17) and CPU <= BASE+(INTERVAL*18)): get_color('
'*1500, bg=rgb(4, 3, 30), end='')
  if (CPU > BASE+(INTERVAL*18) and CPU <= BASE+((INTERVAL*19)-1)):
get_color('  '*1500, bg=rgb(4, 2, 30), end='')
  if (CPU == BASE+(INTERVAL*20)): get_color('  '*1500, bg=rgb(4, 1, 30),
end='')

class MyHandler(BaseHTTPRequestHandler):
  def do_GET(self):
    if self.path == '/test':
      print_color()

while True:
  print_color()
  time.sleep(1)
#try:
#  print_color()
#  #httpd = SocketServer.TCPServer(("", PORT), MyHandler)
#  #print ("Connection established!")
#  #httpd.serve_forever()
#except:
#  pass
```

```
#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>

#define LCD4DPI_GET_KEYS _IOR('K', 1, unsigned char *)

int get_keys(int fd, unsigned char *keys) {
  if (ioctl(fd, LCD4DPI_GET_KEYS, keys) == -1) {
    perror("_apps ioctl get");
    return 1;
  }
  *keys &= 0b11111;
  return 0;
}

int main(int argc, char *argv[]) {
  char *file_name = "/dev/fb1";
  int fd;
  unsigned char key_status;
  fd = open(file_name, O_RDWR);
  if (fd == -1) {
    perror("_apps open");
    return 1;
  }
 key_status = 0b11111;
//while (key_status & 0b00001) { // press key 1 to exit
  while (1) {
   if(get_keys(fd, &key_status) != 0) {
      break;
     // printf("key_status: %x\n", key_status);
    }
   if (!(key_status & 0b10000)) {     // key 1 (leftmost)
    system("echo Scaled to 0 pods");
     system("k3s kubectl apply -f
/home/pi/TheMachineThatGoesPing/ansible/k3s-
ansible/replicasets/replicaset.yaml");
     //break;
   }
  if (!(key_status & 0b01000)) {      // key 2
    system("echo Scaled to 2 pods");
     system("k3s kubectl apply -f
/home/pi/TheMachineThatGoesPing/ansible/k3s-
ansible/replicasets/replicaset2.yaml");
    }
  if (!(key_status & 0b00100)) {      // key 3
    system("echo Scaled to 4 pods");
     system("k3s kubectl apply -f
/home/pi/TheMachineThatGoesPing/ansible/k3s-
ansible/replicasets/replicaset4.yaml");
    //break;
    }
  if (!(key_status & 0b00010)) {      // key 4
    system("echo scaled to 6 pods");
     system("k3s kubectl apply -f
/home/pi/TheMachineThatGoesPing/ansible/k3s-
ansible/replicasets/replicaset6.yaml");
    //break;
    }
```

```
  if (!(key_status & 0b00001)) {       // key 5
    system("echo scaled to 8 pods!!");
     system("k3s kubectl apply -f
/home/pi/TheMachineThatGoesPing/ansible/k3s-
ansible/replicasets/replicaset8.yaml");
    //break;
    }
    sleep(0.1);
  }
  close(fd);
  return 0;
}
```

```
FROM python:2.7-slim
WORKDIR /app
COPY . /app
RUN apt-get update && apt-get upgrade -y && apt-get install -y \
        gcc
RUN pip install psutil
ENV PYTHONUNBUFFERED=0
CMD ["python","./incLoad.py"]
```

```
package
main

        import (
                "fmt"
                "log"
                "math/rand"
                "net/http"
                "os/exec"
        )

        func main() {

                http.HandleFunc("/test", handleRequest)
                if err := http.ListenAndServe(":8080", nil); err != nil {
                        fmt.Printf("Listen and serve failed %s", err)
                        log.Fatal(err)
                }
        }

        func handleRequest(w http.ResponseWriter, r *http.Request) {
                cmd, _ := exec.Command("/bin/bash", "./tesh.sh").Output()
                fmt.Printf("%s", cmd)
                fmt.Println(rand.Intn(100))
        }
```

```
#!/usr/bin/env python

from multiprocessing import Pool
from multiprocessing import cpu_count

def f(x):
    while True:
        x*x

if __name__ == '_main_':
    processes = cpu_count()
    print 'Running load on cpu'
    print 'Utilizing %d cores'% processes
    pool = Pool(processes)
    pool.map(f,range(processes))
```

B Rak8s configuration files

B.1 Rak8s

```
- hosts: all
  roles:
    - common
    - kubeadm

- hosts: master
  roles:
    - master
#    - dashboard

- hosts: all:!master
  roles:
    - workers
```

```
---
# tasks file for minions
- name: Reset Kubernetes
  shell: kubeadm reset -f
  register: kubeadm_reset

- name: Join Kubernetes Cluster
  shell: kubeadm join --ignore-preflight-errors=all --token {{ token }}
192.168.4.1:6443 --discovery-token-unsafe-skip-ca-verification
  when: kubeadm_reset is succeeded
  register: kubeadm_join

- name: Poke kubelet
  systemd:
    name: kubelet
    state: restarted
    daemon_reload: yes
    enabled: yes
  register: kubelet_poke
```

```
---
# tasks file for master
- name: Reset Kubernetes Master
  shell: kubeadm reset -f
  register: kubeadm_reset

#- debug:
#    msg: "Variable is {{ master_ip }}"

- name: "Initialize Master {{ kubernetes_version }}"
  shell: kubeadm init --apiserver-advertise-address={{ master_ip }} --
token={{ token }} --kubernetes-version={{ kubernetes_version }} --pod-
network-cidr={{ podnet }}
  register: kubeadm_init
  when: kubeadm_reset is succeeded

- name: Create Kubernetes config directory
  file:
    path: /root/.kube/
    state: directory
    owner: root
    group: root
    mode: 0755

- name: Copy admin.conf to config directory
  copy:
    src: /etc/kubernetes/admin.conf
    dest: /root/.kube/config
    owner: root
    group: root
    mode: 0755
    remote_src: yes
    backup: yes
  when: kubeadm_init

- name: Join Kubernetes Cluster
  shell: kubeadm join --ignore-preflight-errors=all --token {{ token }} {{
groups['master'][0] }}:6443 --discovery-token-unsafe-skip-ca-verification
  when: kubeadm_reset is succeeded
  register: kubeadm_join

- name: Install Flannel (Networking)
  shell: "curl -sSL https://rawgit.com/coreos/flannel/{{ flannel_version
}}/Documentation/kube-flannel.yml | sed 's/amd64/arm/g' | kubectl create -f
-"

- name: Poke kubelet
  systemd:
    name: kubelet
    state: restarted
    daemon_reload: yes
    enabled: yes
  register: kubelet_poke
```

```
---
# tasks file for kubeadm

# https://gist.github.com/alexellis/fdbc90de7691a1b9edb545c17da2d975
- name: Disable Swap
  shell: dphys-swapfile swapoff && dphys-swapfile uninstall && update-rc.d
dphys-swapfile remove
  ignore_errors: True

# Docker Convenience Script Can Only Be Run Once
- name: Determine if docker is installed
  stat:
    path: /usr/bin/docker
  register: docker_there
  ignore_errors: True

# https://docs.docker.com/engine/installation/linux/docker-
ce/debian/#install-using-the-convenience-script
- name: Run Docker {{ docker_ce_version }} Install Script
  script: "files/get-docker.sh {{ docker_ce_version }}"
  when: docker_there.stat.exists == False
  register: docker_installed

- name: Lock docker version to {{ docker_ce_version}}
  command: /usr/bin/apt-mark hold docker-ce
  when: docker_installed.changed


#- name: Pass bridged IPv4 traffic to iptables' chains
#  sysctl:
#    name: net.bridge.bridge-nf-call-iptables
#    value: 1
#    state: present

# https://kubernetes.io/docs/setup/independent/install-kubeadm/
- name: Install apt-transport-https
  apt:
    name: apt-transport-https
    state: latest

- name: Add Google Cloud Repo Key
  shell: curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |
apt-key add -
  args:
    warn: no

- name: Add Kubernetes to Available apt Sources
  template:
    src: kubernetes.list
    dest: /etc/apt/sources.list.d/kubernetes.list
    owner: root
    group: root
    mode: 0644

- name: apt-get update
  apt:
    update_cache: yes
    autoclean: yes
    autoremove: yes

- name: Install k8s {{ kubernetes_package_version }} Y'all
```

```
apt:
  name: "{{ item }}={{ kubernetes_package_version }}"
  state: present
  force: yes
with_items:
  - kubelet
  - kubeadm
  - kubectl
```

```
token: udy29x.ugyyk3tumg27atmr
podnet: 192.168.4.0/24
master_ip: 192.168.4.1

kubernetes_package_version: "1.11.4-00"
# Available versions:
# 1.10.3-00
# 1.10.2-00
# 1.10.1-00
# 1.10.0-00
# 1.9.8-00
# 1.9.7-00
# 1.9.6-00
# 1.9.5-00

kubernetes_version: "v1.11.4"
# Available versions:
# v1.10.3
# v1.10.2
# v1.10.1
# v1.10.1
# v1.9.8
# v1.9.7
# v1.9.6
# v1.9.5

#docker_ce_version: "18.03.0~ce~3~0~raspbian"
docker_ce_version: "18.06.0~ce~3-0~raspbian"
# Available versions:
# 18.05.0~ce~3-0~raspbian
# 18.04.0~ce~3-0~raspbian
# 18.03.1~ce-0~raspbian
# 18.03.0~ce-0~raspbian
# 18.02.0~ce-0~raspbian
# 18.01.0~ce-0~raspbian

flannel_version: "v0.10.0"
# v0.10.0
# v0.9.1
# v0.9.0
# v0.8.0
# v0.7.1
# v0.7.0
```

C k3s configuration files

C.1 k3s

```
[master]
192.168.4.5

[node]
192.168.4.40
#192.168.4.92
192.168.4.101
192.168.4.194
192.168.4.27
192.168.4.151

[k3s-cluster:children]
master
node
```

```
---

- name: Delete k3s if already present
  file:
    path: /usr/local/bin/k3s
    state: absent

    #- name: Download k3s binary x64
    #get_url:
    #url: https://github.com/rancher/k3s/releases/download/{{ k3s_version
}}/k3s
    #dest: /usr/local/bin/k3s
    #owner: root
    #group: root
    #mode: 755
    #when: ( ansible_facts.ansible_architecture == "x86_64" )

    #- name: Download k3s binary arm64
    #get_url:
    #url: https://github.com/rancher/k3s/releases/download/{{ k3s_version
}}/k3s-arm64
    #dest: /usr/local/bin/k3s
    #owner: root
    #group: root
    #mode: 755
    #when: ( ansible_facts.ansible_architecture is search "arm" and
    #        ansible_facts.ansible_userspace_bits == "64" )

- name: Download k3s binary armhf
  get_url:
    url: https://github.com/rancher/k3s/releases/download/{{ k3s_version
}}/k3s-armhf
    dest: /usr/local/bin/k3s
    owner: root
    group: root
    mode: 755
    #when: stat_result.stat.exists == False
    #when: ( ansible_facts.ansible_architecture is search "arm" and
    #        ansible_facts.ansible_userspace_bits == "32" )
```

```
---

- hosts: k3s-cluster
  gather_facts: True
  become: yes
  roles:
    - { role: download }


- hosts: master
  gather_facts: True
  become: yes
  roles:
    - { role: k3s/master }
      #   - { role: k3s/dashboard }

- hosts: node
  gather_facts: True
  become: yes
  roles:
    - { role: k3s/node }
```

```
---

- name: Copy K3s service file
  register: k3s_service
  template:
    src: "k3s.service.j2"
    dest: "{{ systemd_dir }}/k3s.service"
    owner: root
    group: root
    mode: 0755

- name: Enable and check K3s service
  systemd:
    name: k3s
    daemon_reload: yes
    state: restarted
    enabled: yes

- name: Register file access mode
  stat:
    path: /var/lib/rancher/k3s/server
  register: p

- name: Change file access node-token
  file:
    path: /var/lib/rancher/k3s/server
    mode: "g+rx,o+rx"

- name: Read Node Token from Master
  slurp:
    src: /var/lib/rancher/k3s/server/node-token
  register: node_token

- name: Store Master Token
  set_fact:
   token: "{{ node_token.content | b64decode | regex_replace('\n', '') }}"

- name: Restore file access
  file:
    path: /var/lib/rancher/k3s/server
    mode: "{{ p.stat.mode }}"


- name: Copy buttons script to master
  copy:
    src: /home/pi/TheMachineThatGoesPing/ansible/k3s-ansible/buttons.out
    dest: /home/pi
    owner: pi
    mode: 0775

- name: add buttons.out in bashrc
  lineinfile:
          path: /home/pi/.bashrc
          line: './buttons.out &'

- name: reboot
  reboot:


#- debug: msg="Node TOKEN {{ token }}"
```

```
[Unit]
Description=Lightweight Kubernetes
Documentation=https://k3s.io
After=network.target
[Service]
ExecStartPre=-/sbin/modprobe br_netfilter
ExecStartPre=-/sbin/modprobe overlay
ExecStart=/usr/local/bin/k3s server
# --kube-controller-arg --pod-eviction-timeout=10s
KillMode=process
Delegate=yes
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
TasksMax=infinity
[Install]
WantedBy=multi-user.target
```

```
[Unit]
Description=Lightweight Kubernetes
Documentation=https://k3s.io
After=network.target
[Service]
ExecStart=/usr/local/bin/k3s agent --server https://{{ master_ip }}:6443 --
token {{ hostvars[groups['master'][0]]['token'] }}
KillMode=process
Delegate=yes
LimitNOFILE=infinity
LimitNPROC=infinity
LimitCORE=infinity
TasksMax=infinity
[Install]
WantedBy=multi-user.target
```

```
---

- name: Copy K3s service file
  template:
    src: "k3s.service.j2"
    dest: "{{ systemd_dir }}/k3s.service"
    owner: root
    group: root
    mode: 0755

- name: Enable and check K3s service
  systemd:
    name: k3s
    daemon_reload: yes
    state: restarted
    enabled: yes

- name: Copy colors script
  copy:
    src: /home/pi/TheMachineThatGoesPing/ansible/k3s-ansible/colors.py
    dest: /home/pi
    owner: pi
    mode: 0755

- name: Install pip
  apt:
    name: python-pip


- name: Install psutil
  pip:
    name: psutil

    #- name: Execute colors script
    #command: python /home/pi/colors.py &

- name: Edit bashrc to start script at boot
  lineinfile:
          path: /home/pi/.bashrc
          line: 'python colors.py &'



          #- name: Edit bashrc to starts buttons.c at boot
          #lineinfile:
          #path: /home/pi/.bashrc
          #line: 'sudo /homr/pi/TheMachineThatGoesPing/4DPI-24-
HAT/buttons.out &'

- name: Rebooting
  reboot:
```

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: increaseload
spec:
  replicas: 1
  selector:
    matchLabels:
      name: cpuincrease
  template:
    metadata:
      labels:
        name: cpuincrease
    spec:
     nodeSelector:
       role: worker
     containers:
       - name: increaseload
         image: celebrian/pingmachine:6
         imagePullPolicy: IfNotPresent
```

```
#!/bin/sh
set -x
systemctl stop k3s
systemctl disable k3s
systemctl daemon-reload
rm -f /etc/systemd/system/k3s.service
rm -f /usr/local/bin/k3s
if [ -L /usr/local/bin/kubectl ]; then
        rm -f /usr/local/bin/kubectl
fi
if [ -L /usr/local/bin/crictl ]; then
        rm -f /usr/local/bin/crictl
fi
if [ -e /sys/fs/cgroup/systemd/system.slice/k3s.service/cgroup.procs ];
then
        kill -9 `cat
/sys/fs/cgroup/systemd/system.slice/k3s.service/cgroup.procs`
fi
umount `cat /proc/self/mounts | awk '{print $2}' | grep '^/run/k3s'`
umount `cat /proc/self/mounts | awk '{print $2}' | grep
'^/var/lib/rancher/k3s'`

#rm -rf /var/lib/rancher/k3s
#rm -rf /etc/rancher/k3s
```

D Møtereferater (Norwegian)

Møte 1 - 7. Januar 2019
A116, 1230-1330

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

I dag: Oppklare oppgaven.

(Forrapport inn ca 25 januar, greit å levere lenge før).

På onsdag: Intro med Tom Røise.
-> Hvordan planlegge prosjektet, rapportskisse, etc..

Om oppgaven:

- Prosjektet resulterer i en fysisk "duppeditt", en video, og en presentasjon/ demo.
  Demo: "slidesett", men et opplegg på noe som kan demonstreres.
- F.eks bygge et raspeberry pi cluster.

- Flere ting å tenke på:
  -> Hva er kult å vise fram?
  ->

- Raspberry pi: lav kostnad, lar seg raskt oppgradere
- kan klare demonstrere skalerbarhet via raspberry pi.

Første del av oppgaven:
- Om å gjøre å kartlegge hva som er det kuleste og mest fancy man kan
  bygge med raspberry innenfor kostnadsrammer, og som lar oss visualisere med både
  lyd, farge, etc (lukt?).

Budsjett:
10.000 NOK.

Equipment:
- 8 raspberry pi m/ skjerm og lyd.
- 4 powerbanks
- Billigst mulig fargeskjerm (kjøp fra Norge/ få noen til å kjøpe fra utlandet).
- Varmesensor

Lyd: Optional. fungerer ikke med lyd alle steder.
- Music Angel (høyttaler).

- Når nærmer seg noe ferdig design/oppsett: Prat med Jon Langseth.

-Inntil videre: Skriv ned tanker, gjør research.

Til neste uke:
- forrapport
(noe mer??)

Møte 2 - 14. Januar 2019
A116, 1230-1319

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

Til nå: Jobbet med prosjektplan og funnet ut av billige raspberry PIs som kan kjøpes

Erik skulle sjekke opp budsjett: Snakket med noen i Trdheim, men fikk ikke svar. Bedt om å kontakte sekretær i “Excited”, også snakket med Nils om institutt kan dekke hvis vi ikke får penger (inntill 20k), lar seg gjøre.
Si i fra så fort vi er klare til å bestille (send liste). Erik og Ernst kjøper inn.

Bør kjøpe inn 2 kopier. Sannsynligheten for å få penger øker hvis vi sier at dette er felles for Gjøvik og Trondheim. Kjøper 2 sett, slik at vi kan ha ett i Gjøvik og ett i Trondheim.

Får kanskje budsjett til 20k, men bør holde oss på rundt 10k. Bygg så billig som mulig, men lar seg realisere så lenge vi er innenfor 20k.

Bør bestille så tidlig som mulig, siden det kan ta opptil en måned før varene kommer.

Vi bør bruke wifi, pga kan bli trøbbel med mye kabler og vi trenger en switch i tillegg.
Kan ha usb-drevet switch, men det fører ikke til noe særlig god ytelse.

Clas Ohlson har powerbanks, opptil 6700mah.

I første omgang konsentrere oss om raspberry pi boks og skjerm som vi helst bestiller denne uka. (Fra asia? 7-25 dager levering) (den komponenten som er dyr og vanskelig å få tak i norge (billig))

Bestiller vi fra utlandet vil det sannsynligvis komme på moms og toll. Ca 300kr i toll + 158 i firmaet som sender gebyret, men bare ett gebyr hvis alt sendes i ett kolli.

Vi trenger en raspberry pi model 3B+ (har den på Power, men ikke i Gjøvik)

Skjermene brukes ikke HDMI, men noe annet som fungerer til å koble til skjerm. Skjermdriver ligger på github. Bare pass på at skjermen passer til de raspberry pi’ene vi kjøper. Skjermen må

også passe til OSet som kjører (rasbian). Se om vi finner erfaringer på om det virker eller ikke.

Raspberryene trenger ikke tilgang til nett på utsiden. (kan føre til sårbarheter/exploits, e.g., noen hijacker skjermen, etc).

Mål for uka: Få til en bestilling på de vi har snakket om i dette møtet. Sikter på å få gjort dette på fredag denne uken (skjerm og kasser fra asia + raspberry pi). Vær sikker på at dette er noe vi får til å bruke. Tenker på å bestille 9 (raspberry pi), i tilfelle en ikke virker. Bestiller bare ett sett i utgangspunktet. Erik bestiller og legger ut for det.

Mål til neste mandag:

- Ferdig med punkt 6 i prosjektplan (veldig nærme ferdig versjon)
- Ferdig med grupperegler og roller
- **Store deler av prosjektplanen skal være ferdig denne uken.**
- **Til fredag: Send liste til Erik med ting som skal bestilles, så bestiller han det**

Tips: Bruk prosjektplanen som en del av starten på prosjektrapporten (med noe omskriving så det passer inn med rapporten).

(Ekstra)Møte 3 - 18. Januar 2019
A117, 1100-1130

Tilstede:
Martin Brådalen
Trond Håvard Thune
Erik Hjelmås (Oppdragsgiver)
Kjell Are Refsvik (Raspberry Pi ekspert)

Ikke tilstede:
Per-Kristian Kongelf Buer
Ernst Gunnar Gran (Veileder)

I dag: Levere handleliste på utstyr som skal kjøpes inn.

Strømforsynging: PoE. Slipper separat strømforsyning.
Oppsett som vi lager i en koffert, som kan tas med rundt på stands.
Vi bruker wifi, pga mye styr med bruk av switch hvis ethernet.
Hvis vi skal utnytte PoE, kan vi få både strøm og enkelt nettverk, men må da ha en switch som gir PoE, som kun kan drives av en powerbank, og det er verre.
Skal være 100% mobilt, ikke nødvengvis, men foreløpig det vi tenkte.
Kan bli mye elektroarbeid, spesielt hvis mange raspberries.

Send link med powerbank, og banggood. Erik hører med jingjing hvor han skal kjøpe Pi i fra.

Kjell Are har noen Raspberry Pis (6 stk?) som vi muligens kan få eksperimentere med mens vi venter på at utstyret som bestilles kommer.

Møte 4 - 21. Januar 2019
A116, 1230-1300

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

Forprosjekt:

WBS: Viktisgte at aktivitetene er strukturert. Når man kommer til en aktivitet, mer innsikt i hvordan den bør deles inn i deler. Bør ta gjennomføringen da, etter hvert.

Gantt-diagram: Leg ved nøyaktig kopi, ikke revidert versjon.

- Bryt opp noen ting i mindre deler (f.eks "planleging/utføring av (...).

Senest fredag: Lever WBS og endelig Gantt-diagram til Ernst. Prater mer om den påfølgende mandag.

Fremover:
Tenke på hvordan produkt / video skal demonstreres.
Kan få låne raspberrys av Kjell Are ved behov.
Bruk Google/Youtube! (Med flest mulige innfallsvinkler). Garantert noen som har gjort dette før oss.
Lavterskel: Skjermene skal bare vise load-parameter som farge. Kan være et slags "første demo". Kan traffiken økes ved å tappe på skjermen?

- Eks: Lysegrønn ved lav last, så rødere for mer last.
- Eks: Hvis tap, skjerm svart, tap igjen, starter opp igjen.

Vi bør bruke Python. Første blir å sette opp et py-program som kan kommunisere med skjermen. Muligens lettest å jobbe mot et python library. Google it for å sjekke.

Tips: Ha lave ambisjoner fra starten. Bedre å ha noe å vise frem enn å lage noe kjempestort og ikke rekke å bli ferdig med det.

Greit å bruke ting man finner andre steder, istedet for å lage noe nytt fra scratch. Men husk å referer til hvor man har det fra!

Møte 5 - 28. Januar 2019
A116, 1230-1302

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

Ikke tilstede:
Trond Håvard Thune (syk)

I dag:

Tema for oppgaven er robusthet og skalerbarhet
Prosjektplan allerede lastet opp.
Ernst skal se over forprosjektrapport.
Glemte å kjøpe SD kort til Raspberry PI'ene, Erik redder dagen ved å ofre sitt eget.
Starte en med skjerm, få visualisere noe på skjermen og lyd.
Huske at kjernen i oppgaven er å vise noe kult/skalerbart.
Visualisere brute force passord knekking.

Til neste møte:

Ikke møte mandag 4. Feb, 2 uker til neste møte
Research Micro SD kort. Finne den som har best ytelse til best pris.
Sette opp en raspberry Pi som et access point.

Møte 6 - 11. Februar 2019
A116, 1230-1338

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

I dag:

- Om det er noen ting vi ikke får gjort enda, så gjør noe annet istedet som kan gjøres.
- Funnet noe kalt “Hypriot”, til raspbian os / cluster.
- Har ikke fått til kobling med skjerm enda, pga manglende minnekort.
- Når vi får til skjerm, må vi få til et script som viser enkle visualiseringer (warnings, etc).
- En bør gjøre teknisk, mens to andre tegner og forklarer “hva nå?”.
- Har vi funnet noen demoer enda som viser raspberry pi cluster på noen måter? (linker i google docs, bilder på nettet, etc)..
- Må vise/ gi tilbakemelding på last på clusteret.
- Hvordan kan man sørge for å dele jobber på antall noder tilgjengelig? Må ha en form for distribusjon, løsning på det.
- Kan klone minnekort så lenge det er linux noder med DHCP.
- Bør lage en tegning over hva vi skal gjøre/ hvordan vi skal lage løsninger.
- -> 1x controller (last, styrer helo demooppsettet med, generer last mot clusteret), 1x manager (dhcp,), 6x “vanlig” raspberries (skal se at de fylles opp og ned).
- **Hjelmås krever en demo den 25. Februar. (Også innafor i forhold til Gantt-skjemaet).**

Til neste møte:

- Få installert Raspbian OS på alle PI
- Få installert skjerm på alle PI og få dem til å kommunisere så de viser noe på skjermen.
- Begynn på et (enkelt) script for å øke og generere last (på en enkel måte).
- Finne ut hvordan schedule/ dele ut jobber fra controller/ manager node.

Møte 7 - 18. Februar 2019
A116, 1230-1338

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)

Ikke tilstede:
Erik Hjelmås (Oppdragsgiver)

I dag:
Viser frem praktisk minidemo for veileder.
Viktig at vi fokuserer på å lage løsninger som svarer på oppgaven, ikke henge seg så opp i tekniske ting som vi bruker for lang tid på.
Generere last - cpu kraft mot et svært primtall??
Visuell - lage noe som endrer skjermen, i farger (rødt, grønt, etc.. et eller annet som indikrerer trafikk/last).

Til neste møte:
Ha klar en demo som viser visuell output. Gjerne så mange Raspberrys som mulig (flere enn tre noder).

Møte 8 - 25. Februar 2019
A116, 1230-1322

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

I dag:
Demo 01 av The Machine That Goes Ping:

- 4 raspberry, der 1stk er controller (sender trafikk), og 3 er workers som mottar trafikk.
- Controller broadcaster privat nettverk med ssid PiTest som workerne kobler seg til.
- Load balancing med haproxy, docker swarm har vi tenkt å bruke, men er ikke tatt i bruk på.
- Golang er hittil brukt for workerne å lytte på trafikk.
- Har ikke fått til farget output enda på de små skjermene, men det fungerer på terminal.

Feedback fra oppdragsgiver og veileder:

- Q: Hvordan ser vi for oss at den endelige demoen blir, og hvordan fungerer den for brukeren og under panseret? A: Retningen vi går i er at man har en controller med resten noder 5-6 stk, og vi har ikke bestemt oss for at man kan ha flere muligheter slik at f.eks trykker på en knapp og på hver av dem kommer requestene, plugger ut og inn selv, load balancer seg selv. En annen knapp som sender tyngre arbeid over til de, f.eks hvis vi gir dem noe som f.eks sjekker at et tall er prime. Kan f.eks hvis for mye av CPU de bruker.
- Q: Mtp skalering, mer jobb å gi beskjed enn å svare på dem. Jobben man gir til noen må relativt være mer krevende enn trafikken man sender også, i verste fall klarer ikke controller å overbelaste workerne.
- Ikke bekymre for overoppheting. Kan hende det kuleste er å ha det liggende utover et bord, som i denne demoen. Viktig at oppsettet fungerer, gir blaffen i innpakningen foreløpig. Oppdragsgiver synes dette er kjempebra, føler vi er på trygg vei. :)
- Direksjon: For veien videre, IKKE sett opp alt manuelt med scripting. Dere bør demonstrere egenskaper ved et system, bruk et eller annet system (f.eks docker swarm).
- **Undersøk dette:** Hva er enkleste måten å kjøre k8s på Raspberry pi? K8s har sitt eget deployment-verktøy, det er observert at det er mer og mer modent.
- Hva hvis det er slik at man har controllern, og den f.eks er en progressbar / lastkontroll, trykker + eller - fra 0 til max last. Og så ligger 5-6-7 pier med minimumslast, så er det 2

som lyser grønt som betyr at det er 2 servere som håndterer lasten. Og så øker man laster, så starter det opp flere og øker kanskje lasten opp halveis, og da starter alle sammen, alle lyser grønt som betyr at det kjører en node på alle. Fortsetter man åt rykke da, så blir noen gule, med økt last, og så kobler man ut 2stykker, da blir alle gule og noen rød, så kobler du inn igjen en eller to, så er det noen som blir grønne og/ eller gule. Vise grad av parallelitet, last per node og at man kan koble til og fra.

- Ser hele tiden total last (f.eks i prosent) på controllern. Endrer fargen når lasten økes, ved et visst nivå (begeret renner over), starter det en ny Pi og alle fargene går ned til grønt for å reflektere load balancingen.
- Virkemåten er likt distribusjon av jobber i et system hvor man har varienrende grad av noder tilgjenglig.
- I k8s kan man gjøre alt det her.
- Hvilken oppgave er det som spiser 10% av en Pi?
- Tenk så enkelt som mulig. Hvis en node feiler, så restarter den.
- Lurt å gå for et Ansible k8s setup. Er en risikofaktor, men oppdragsgiver tror at det er mange som har kjørt k8s på raspberry pi. I sent 2017 ble k8s for raspbian lifta inn i official k8s.
- Se for dere at dette skal kunne gjenbrukes av andre, kjører en tidshorisont på 5 år. Dette er en demo som skal kunne brukes 4-6 ganger i året i en 5-års periode. Det er nye personer som skal ta over. Hvordan ta disse, koble dem sammen, deploye?? Ta høyde for at ny person skal rulle dette ut etter at vi er ferdige.
- Hvis det tar mye tid å implementere i en varig produkssjonssystem, er det viktig å huske på at detet skal kjøre i et lukka system. Er da ikke nødvendig å oppdatere ubuntu.
- Tenk at det er en student som skal ta over og vise det frem på stands, etc.

<u>Til neste møte:</u>
Researche k8s
Researche Ansible
Fortsette å finne ut hvordan vise farger på de små skjermene

Møte 9 - 11. Mars 2019
A116, 1230-1300

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

I dag:
Forteller om hva vi har drevet med. Spør om ting angående k8s. Bruk gjerne k8s-kommandoene fra IaC fra i fjor høst.

Angående neste mandag; prototype v2. Håper å få til k8s og ansible så vi kan demonstrere det, fokuserer på k8s og ansible så vi kan fra de to sette opp clusteret (installere k8s og sette opp clusteret på alle nodene).

Prototype v2: Ansible og kubernetes
Prototype v3: Ansible og kubernetes med load og farger, alt er sydd sammen.

Når vi jobber, husk på å dokumenter og skriv i tillegg, slik at ikke alt bøtter over oss mot slutten.

Skriver rapport etter påske, men husk på at veileder trenger litt på forhånd hvis skal leses igjennom. Kan hende ikke ferdig til påske. Kan glemme og notere og husker ikke ting riktig underveis. Ta stikkord!

Ang rapporten: Fortell en historie for noen som ikke kjenner til prosjektet fra før. Skriv på et slikt kunnskapsnivå som vi hadde før vi begynte på bacheloren. Se hvordan tidligere bacheloroppgaver har strukturert. Har en avtale med en annen gruppe om å lese hverandres oppgaver.

Spørsmål angående videoen: Hva skal den inneholde? Når vi kommer i havn, hva er den kuleste demoen vi kommer med? Produser en video med den.

Til neste møte:
Fra manager installere Ansible og legge inn de andre nodene, og få de inn i kubernetes. (?)
Forbered prototype v2 til fremvisning, som skal vise ansible og k8s i aksjon.

Møte 10 - 18. Mars 2019
A116, 1230-1300

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)

Ikke tilstede:
Erik Hjelmås (Oppdragsgiver)

I dag:
Skulle ifølge gantt-skjemaet presentere demo 2 i dag, men dette er blitt utsatt en uke pga vi sliter med forskjelige ting med kubernetes (...).

Har personer før som har gjort det samme som oss et problem med høy CPU load når k8s kjører?
Er det et problem at CPUen går på 50%? Kan bare endre scriptet ettersom. F.eks lav last kan være 50% og høy/max last kan være 100%.

Har problemer med at Ansible forsøker å laste ned/ oppdatere programvare fra Internet. Det fungerer dårlig når vi skal være på et lukket nettverk. Men man kan bare laste ned første gangen med nett, og deretter være på lukket nett, spiller ingen rolle om programvaren blir gammel og utdatert da.

Møte 11 - 25. Mars 2019
A116, 1230-1336

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

I dag:
Demo 2: Ansible og K3S (Lightweight Kubernetes)

Prøvde å sette opp et dashboard før demo men får ikke til å få til ssh desktop, siden vi mangler en ekstern skjerm å koble til. Bruker litt tid på å få til dette, med hjelp fra Hjelmås. Får til til slutt, via PKs pop-os.

Satt opp slik at Ansible setter opp k3s, med en master og to ndoer for denne demoen. Dashboard kjører som en service på clusteren. Skal også etterhvert kjøre et script som viser farger basert på CPU load.

Er vi avhengige av at en container skal vise noe? Planen er det, vi tenkte at det er en service i k3s som colors (en på hver), som sørger for at fargen på displayet tilsvarer loaden på noden. Hvis vi ikke får til, kan den kjøre ved siden av. Må sette opp slik at output på displayet er colors.py utenfor k3s. Skal se mer på det denne uken.

Ansible playbook feiler underveis; “file not found”. Fungerer etter fjerde kjøring.
Har nå fått til at Ansible setter opp k3s, med noen småbugs. Setter opp en cluster, krever nett første gang fordi ansible laster ned k3s. Lager en master, joiner noder, og starter en dashboard service. Trenger ikke nett når clusteret skal settes opp neste gang.

Det som står igjen er å få colors-scriptet som vises på skjermen, og at jobber distribueres mellom nodene. Skal ha noe som skrur laster opp og ned, og kunne plugge inn og ut noder.

Få noen andre studenter til å sette opp clusteret basert på vår oppskrift.

Prøver å viser dashboard, men returnerer HTTP ERROR 502 (internal server error). Det fungerer plutselig rett etter oppdragsgiver har gått. Selvfølgelig.

Hvis vi plugger ut strømmen og inn igjen, vil PIene sette seg opp igjen selv? Sletter alt først, setter så alt på nytt.

Oppdragsgiver er svært fornøyd med demoen. Til å være et første forsøk på et komplisert oppsett er det veldig bra.

Til neste møte:

- Finne ut med ready/ not ready i k3s.
- Finne ut med å justere load opp og ned.
- Distribuere load, slik at hvis en node faller ned, går loaden på de gjenværende nodene opp.

Møte 12 - 1. April 2019
A116, 1230-1308

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)

Ikke tilstede:
Erik Hjelmås (Oppdragsgiver)

I dag:

Fått til at en container starter en jobb, men varer kun ca 1 milisekund. Det er en while True løkke i koden, så den burde i teorien gå evig. Går evig via ssh, men ikke via k3s.

Det ene vi har å gjøre er at jobben kjører så lenge man ønsker.
Det andre handler om å endre lasten. Lettere å håndtere mange småjobber enn få store jobber.
Må finne ut en måte slik at de små prosessene kjører evig i containerne. Se litt på hvordan vi kan gjøre det med delay (sleep()) i colors-scriptet.

Til neste møte:

Demo 3 mandag neste møte.
Finne ut hvordan lasten kan skaleres opp og ned.
Få k3s til å handtere skalering.
Prøve å få til slik at hvis en node plugges inn/ut, redistribueres lasten.
Lage mange små jobber for enklere håndtering av skalering.

En fremgangsmåte fra Ernst: La knappene avgjøre hvor mange jobber som skal sendes.
Venstre knapp: 20 jobber.
Nest venstre knapp: 30 jobber.
Midterste knapp: 40 jobber.
Nest høyre knapp: 50 jobber.
(Høyre knapp: Killswitch som stopper alle python prosesser).

Møte 13 - 08. April 2019
A116, 1230-1315

Tilstede:
Martin Brådalen
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

Ikke tilstede:
Per-Kristian Kongelf Buer

I dag:

Demo 3:

Problem vi har støtt på: K3s sin time_eviction er 5 minutter, så det vil si at å skalere ned tar 5 minutter, som fungerer dårlig for en live-demo. Skal kunne endres i kube-controller-manager, men den finnes ikke i k3s? Kan gå tilbake til k8s, men da får nodene veldig mye CPU load.
Hvor mye jobb er det å gå tilbake til k8s? Kan bare endre på verdier i python-scriptet.
Kan eviction_time finnes i kildekoden til k3s? Etcd? Det skal finnes noe config i etcd, siden det er en delt ressurs som alle komponentene i k3s henter configen sin fra. Egentig ikke, den bruker sqlite(3) istedet for etcd.

I denne demoen kan man trykke på knappene på skjermene for å skalere. Knappene starter n antall pods. Regnejobber som krever CPU kjørerer på podene. En jobb krever ikke så mye regnekraft, men vi skalerer med flere antall pods som krever litt CPU, det blir tilsammen mye CPU, og det er enklere å håndtere og skalere opp/ ned.

I rapporten, skriv gjerne litt bruk om farger, henvis til en best practice menneke-maskin interaksjon av hvordan fargen oppfattes. (F.eks rød oppfattes forskjellig rundt om i verden).
Skriv også om utfordringer og ting vi gjør.

Stort sett er det kun å finne ut av eviction_time vi har igjen å gjøre.
Avtaler fjerde og siste demo den 6. Mai. Da sendes også et utkast på rapporten til veileder.

Møte 14 - 29. April 2019
A116, 1230-1313

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

I dag:
Snakker om lengde på rapport. Bør være heller bra enn altfor mange sider. Ca. 50-80 sider uten vedlegg. Vær nøye når dere skriver, tenk over hva dere skriver, les over det andre skriver.

Angående tid det skal skrives i (presens, preteritum) er det ikke noe universalsvar, men vær konsekvent og hold dere til én tid. Pass på så det ikke blir for mye "historiefortelling". Må forklare de valgene man har gjort, men ikke la rapporten være kronologisk fortelling over arbeidet. Når man skal forklare teknologier skriver man gjerne i presens, siden det er noe som er nå og ikke som var.

Få frem når vi har gjort noe og når noe er gjort som er lånt fra andre. Kan bruke vi som om det er forfatteren og leseren. I hovedsak unngå "jeg". Generelt styr unna personlig pronomonen, med mindre man sier noe om gruppa eller vil ta med leseren på noe. Vi er ikke viktig, men det vi skriver om er det.

Når går det fra å ha kode naturlig i koden til å ha det som vedlegg? Hvis det f.eks er en stor config fil og det endres to linjer, kan man bare ha med de to linjene som en snippet i teksten. Hovedsaklig ikke noe poeng i å legge ved kode i teksten. Hvor mye kode man har i teksten kommer an på hvor mye tekst det er og leseflyt. Kan ha en kodebit til å bryte opp hvis det er fem sider med tettpakket tekst.

Når det gjelder språk, er det viktig å tenke over "det/disse/dette, …" uten at det står noe subjektiv der. Har en ide hva "dette" er for noe, men refererer ikke til den tingen det dreier seg om.

Når det refereres til figurer, så legg en ref link i teksten i stedet for å putte figuren rett under.

Til neste gang
Skrive rapport som leveres til veileder og klargjøring av siste demo.

Møte 15 - 06. Mai 2019
A116, 1230-1313

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Ernst Gunnar Gran (Veileder)
Erik Hjelmås (Oppdragsgiver)

Ikke tilstede:
Trond Håvard Thune

I dag:
Demo 4: Siste demo, dette bli etter alt om dømme det endelige produktet. *(hvordan ble det med eviction time?)*

Et utkast til rapport leveres til veilederen. Gruppen har en skriveprosess mens denne leses over av veileder, og det vil komme feedback på ting som bør endres og tilføyes. Gruppen trenger ca. 20 sider til for at det skal være en typisk lengde på en bacheloroppgave.

Til neste gang
Neste møte blir trolig siste møte, siden oppgaven skal leveres mandag den 20. mai.

Møte 16 - 13. Mai 2019
A132, 1230-1300

Tilstede:
Martin Brådalen
Per-Kristian Kongelf Buer
Trond Håvard Thune
Ernst Gunnar Gran (Veileder)

Ikke tilstede:
Erik Hjelmås (Oppdragsgiver)

I dag:
Siste møte før innlevering av bacheloroppgave. Møtet dreier seg om rapporten.

Manglet det “mye” noe steder i rapporten? Det største var at det plutselig går fra hva oppgaven går ut på til design. Det manglet noe imellom der der man “fant mer ut av oppgaven”. Mye av dette blir rettet opp i skrivende stund. Kanskje også ha noe med vannkjøling, hvorfor det ikke ble valgt etc …

In-document referanser bør man merke tydelig med hva man referer til (section, figure, …).

Ved direkte sitering må det være markert tydelig i teksten. Bruk hermetegn og forklar hvor det kommer fra, men bruk så lite tekst som mulig på det.

Ved introduksjon til nye begreper, ha en kort forklaring (en-to setninger) som forklarer hva det er og legg ved en referanse.

E Oppgavebeskrivelse (Norwegian)

Forslag til bacheloroppgave NTNU:

The machine that goes Ping

Oppdragsgiver: IIK v/Erik Hjelmås, erik.hjelmas@ntnu.no

Om IIK

Institutt for informasjonssikkerhet og kommunikasjonsteknologi (IIK) holder til både på campus Trondheim og campus Gjøvik. IIK har stor forskningsaktivitet og får mye oppmerksomhet fra eksterne aktører.

Oppgaven

Vi har et stort behov for å vise frem at digital infrastruktur, skyløsninger og operativ cybersikkerhet er spennende og kult. Hver gang vi får besøk, om det være seg statsråder, bedrifter, foreninger eller om det er åpen dag for elever fra videregående, så må vi kunne visualisere og demonstrere vår aktivitet. Denne oppgaven dreier seg om å designe en løsning, både fysisk og virtuelt som kan benyttes i slike anledninger:

- Hva kan vi fysisk vise frem? en stack av raspberry PIs? vannkjøling og neonlys? trafikklast med LED-indikatorer? “pimp my server rack”? osv osv
- Hvilke aktiviteter kan vi ha som engasjerer? spesielle kahoot? sync telling i fellesskap som protokolldemo? table top hacking demo? osv osv
- Hva kan demonstreres mtp sikkerhet, ytelse og skalebarhet mtp VMer, containere og sky? autoskalering? evn med lysende raspberry PIs? osv osv

Det forventes at gruppen leverer et design og prototyp av minst en fungererende løsning. Oppgaven innebærer mye undersøkelse av hva andre gjør, mye kreativitet og mye implementasjon.

F Prosjektplan (Norwegian)

F.1 Prosjektplan (Norwegian)

Prosjektrapport for “The Machine That Goes Ping”

Martin Brådalen (473145)
Per-Kristian Kongelf Buer (480237)
Trond Håvard Thune (473147)

Bakgrunn

IT Drift og Informasjonssikkerhet er en linje ved NTNU Gjøvik. Dette er linjen for de som er interessert i å lære om hvordan datamaskiner og nettverk fungerer og driftes, i tillegg til hvordan gjøre den teknologiske hverdagen sikrere.

Ved NTNU i Gjøvik har mange av studieretningene noe å vise frem til verdenen utenfor campus. Sykepleiestudiene har paramedic-simulatoren å vise frem[1], spillprogrammering har VR-laben[2] og informasjonssikkerhet har biometri og fingeravtrykksensorene[3], for å nevne noen.

Vår oppdragsgiver ønsker seg at også studiene ved IT-drift har noe å vise frem, spesielt med tanke på å demonstrere skalerbarhet i tillegg til ytelse og sikkerhet.

Vårt oppdrag er å lage et resultat sammensatt av en fysisk “duppeditt” og en video som kan brukes som reklame for å trekke flere personer til å få en interesse for drift og sikkehetsfag. Målet vårt er å lage noe som er kult og som trekker blikkfang når det vises frem. Med begrepet “kult” mener vi noe som for eksempel blinker, beveger seg, lager lyd, og som er interaktivt. Ved å lage noe som inneholder en eller flere av disse elementene, mener vi at vi har noe som tiltrekker seg et publikum, og som er med på oppnå det som oppdragsgiver ønsker seg; noe “kult” å vise frem når Institutt for informasjonssikkerhet og kommunikasjonsteknologi får besøk.

Om oss

Vi er tre studenter ved IT-Drift og Informasjonssikkerhet, som alle har en viss interesse for drift. I Løpet av studiet har vi fått kjennskap til mange forskjellige driftingsteknologier deriblant, Puppet, Docker, OpenStack, og det var derfor naturlig for oss å velge en oppgave innenfor retningen drift.

---

[1] "Simulation at NTNU i Gjøvik - YouTube." 21 Jan. 2016, https://www.youtube.com/playlist?list=PLfjNXXnOnB_YWJLRMuAGWjANDdSdAYeqG. Åpnet 11 Feb. 2019.
[2] "VRLab - NTNU." https://www.ntnu.no/laeringsarealer/vrlab. Åpnet 11 Feb. 2019.
[3] "NTNU på Gjøvik er blant de fremste på datasikkerhet i Skandinavia ...." 28 Dec. 2017, https://www.tu.no/artikler/ntnu-pa-gjovik-er-blant-de-fremste-pa-datasikkerhet-i-skandinavia-na-vil-de-bidra-til-a-avslore-pedofile/415052. Åpnet 11 Feb. 2019.

Oppgavebeskrivelse/Problemstilling

Oppdragsgiver vil at produktet skal være noe som er “kult” å vise frem, som kan brukes av mange forskjellige målgrupper, alt fra barnehage/grunnskolenivå til næringslivstopper og ledere. Foreløpig er arbeidstittelen på dette produktet “The Machine That Goes Ping”, en referanse til Monty Python’s the Meaning of Life[4].

Mål og Rammer

Prosjektmål

Vi skal utvikle et helhetlig produkt/ prototype som består av tre deler: Et fysisk produkt, en video, og en presentasjon/demo. Presentasjonen kan være et slidesett som medfølger bruk av produktet, men det viktigste er å lage et opplegg på noe som kan demonstreres.

Produktet vil bli et cluster med åtte stykk Raspberry Pi med skjermer som vil vise noe som demonstrerer skalerbarhet og ytelse, gjerne med tanke på å demonstrere skalerbarhet overfor lekfolk. Det skal lages på en modulær måte og skal enkelt kunne replikeres, siden planen er at NTNUs campuser i Gjøvik og Trondheim skal ha hvert sitt kluster å vise frem.

Effektmål

Vår mål er å lage et produkt som linjen IT-Drift og Informasjonssikkerhet kan ta med seg og vise frem på for eksempel karrieredager eller besøk på videregående skoler. Målet er at vårt produkt skal fungere som et blikkfang og reklame som forhåpentligvis vekker interessen for IT drift i flere ungdommer.
Langsiktig mål er å trekke flere studenter til å søke IT studier på NTNU Gjøvik og etterhvert gjøre drifting av nettverk- og datasystemer mer populært blant yngre.

Resultatmål

Etter utførelsen av denne bacheloroppgaven skal gruppen ha laget en Raspberry Pi cluster som driftes som en docker swarm. Dette skal være en fysisk gjenstand som NTNU skal få.

---

[4] "Monty Python's The Meaning of Life - Wikipedia." https://en.wikipedia.org/wiki/Monty_Python%27s_The_Meaning_of_Life. Åpnet 11 feb.. 2019.

Rammer

Økonomiske rammer:
Produktet må utvikles innenfor en gitt kostnadsramme på rundt 20.000 NOK.

Fysiske rammer:
Produktet skal være mobilt; altså det skal være mulig å demonstrere uten tilgang på en stikkontakt, og kunne bli drevet av batterikraft en periode. Det burde også være lagret på en portabel måte som gjør clusteret lett å transportere og sette opp.
Logiske rammer:
Raspberry PI er en god del mindre enn en vanlig PC; dette må vi ta høyde for ytelsesmessig når vi utvikler for Raspberry PI. Raspberry PI kjører også sannsynligvis et eget Raspberry PI-basert OS (Raspbian) som vi må forholde oss til, i motsetning til om det var Ubuntu eller Windows operativsystem. En bonus her er at både Raspbian og Ubuntu er basert på Debian, så det burde ikke være veldig langt unna det vi bruker vanligvis.

Omfang

Fagområde/Problemområde
Vårt fagområde hører til drift av IT- og nettverkssystemer, som er en gren av IIK (Institutt for informasjonssikkerhet og kommunikasjonsteknologi) ved NTNU.

Avgrensning/Problemavgrensning
Vi avgrenser oss til å utvikle de tre hoveddelene som oppdragsgiver har bedt oss om å gjøre: Utvikle et fysisk produkt, en video, og en presentasjon/demo. Vårt produkt/prototype vil kunne demonstrere Skalerbarhet, robusthet og sikkerhet, fordelt ut over åtte stykk Raspberry Pi.

Avgrensning

Fysiske avgrensninger

Av budsjettmessige årsaker velger vi å begrense oss til å bruke åtte Raspberry Pi versjon B+ med skjerm og protective case (en liten eske man legger selve raspberryen inni) per cluster. Sluttproduktet vårt skal være portabelt å må derfor være innenfor rammene til hva en person klarer å bære med seg. Produktet er avhengig av strøm; vi har enn så lenge løst dette ved å bruke powerbanks.

Generelle avgrensninger

Ønskelig vil vi at produktet vår skal fungere på et lukket nettverk, og ikke være avhengig av internett. Vi holder oss til Raspian OS, som er standard operativsystem på Raspberry Pi. Alt som skal vises, vises bare på skjermene tilkoblet til Raspberry Piene.

Prosjektorganisering

Ansvarsforhold og roller

Referatansvarlig: Trond Håvard Thune
Prosjektleder: Per-Kristian Kongelf Buer
Innkjøpsansvarlig: Martin Brådalen

Rutiner og regler i gruppa

Møter med veileder og oppdragsgiver:
Det gjennomføres 30-60 minutters veiledningstimer med veileder og oppdragsgiver hver mandag klokken 12:30, unntatt i påsken.

Gruppearbeid:
Felles arbeidstid på campus bestemmes fra uke til uke, siden gruppens medlemmer også har andre emner og gjøremål som gjør at vi ikke kan legge statiske føringer på hvordan ukene våre ser ut.

På ulike tidspunkt i uken vil en eller flere av gruppens medlemmer være opptatt med andre emner, eller emner der gruppens medlemmer er læringsassistent i. Under disse tidspunktene vil de resterende av gruppens medlemmer gjøre separate oppgaver. Hver mandag før veiledningsmøte vil gruppen gå igjennom det alle har gjort hver for seg og sy dette sammen til en helhet.

Gruppen jobber med bacheloroppgave sammen i gruppe
hovedsakelig på mandager, onsdager og torsdager. En eller
flere av gruppens medlemmer er opptatt med andre emner tirsdager og fredager. Det forventes at hvert gruppemedlem bruker mellom 20-30 timer i uken på bacheloroppgaven. Antall timer dokumenteres på egenhånd ved hjelp av toggl[5]. Antall timer vil i utgangspunktet ikke bli dokumentert pga gruppemedlemmer er enig om at det å dokumentere arbeid i stedet for timer er en bedre måte å presentere arbeid. Derfor er en ukentlig logg over hva hvert enkelt har gjort forventes.

Fravær:
Ved sykdom gis det beskjed til de andre hva den syke hadde tenkt å gjøre, slik at de gjenværende kan diskutere seg imellom om det er nødvendig å fordele de arbeidsoppgavene mellom seg, eller om det kan vente til den syke er tilbake.

Oppbevaring og sikring av fysisk utstyr:
Fysiske deler av prosjektet vil bli splittet mellom medlemmer for å minske risiko for at ting blir stjålet.

Sikring (backup) av prosjektrapporten:
Prosjektrapporten skrives i ShareLaTeX/Overleaf (et online latex-miljø). Prosjektrapporten er linket mot Per-Kristians fork av GitHub-repoet som malen til prosjektrapporten er basert på. Innholdet pushes periodisk fra ShareLaTeX/Overleaf til dette GitHub-repoet. Derfra kan hver av medlemmene laste ned en kopi med git clone / git pull.

Loggføring
Alle individuelle og/ eller gruppebaserte beslutninger og oppgaver som er utført skal det føres logg av. Det loggføres i en dedikert mappe i gruppens Google Docs[6].
Det trenger ikke å være en formell tekst som skrives, men loggen skal inneholde den individuelles/ gruppens tanker og refleksjon mot de valgene som er gjort i utførelsen av oppgavene. Her skal det blant annet opplyses om begrunnelser for valg (hvorfor vi til slutt bestemte oss for å gjøre A i stedet for B og C, etc). Loggene danner grunnlag for skriving av endelig sluttrapport som leveres den 20. mai 2019.

[5] "Toggl." https://toggl.com/. Åpnet 11 Feb. 2019.
[6] "Google Docs: Free Online Documents for Personal Use." https://www.google.com/docs/about/. Åpnet 11 Feb. 2019.

Kvalitetssjekk på arbeid:
Alt individuelt arbeid som vil tilføyes til sluttprodukt må først gjennomgås og godkjennes av de andre gruppemedlemmer før det tilføyes til produktet.

Sanksjoner mot brudd på rutiner og regler:
Ved kontinuerlig brudd av grupperegler av en eller flere medlemmer kan prosjektleder innkalle til møte hvor brudd og eventuelle sanksjoner kan diskuteres mellom den tiltalte og de resterende medlemmer. Hvis interne tvister ikke kan løses innad i gruppen, er det mulighet for å tilkalle veileder til å ta avgjørelser som en ekstern, upartisk stemme.

Brudd på regler fører til en advarsel. Det vil bli gitt opptil og inkludert tre advarsel. Etter dette kan videre brudd føre til alvorligere konsekvenser, i verste fall utestengelse fra gruppen.

Planlegging, Oppfølging og Rapportering

Hovedinndeling av prosjektet: Valg av Systemutviklingsmodell og valg av metode og tilnærming
Vårt produkt vil kunne bygges i moduler, men vi ser ikke for oss at produktet kan taes i bruk før det er ferdig/ begynner å nærme seg sammen. Forskjellige komponenter skal kunne snakke sammen, og vi må ha disse ferdige for at de skal kunne gjør det.

Vi har kommet frem til at vi velger følgende bruk av systemutviklingsmodeller:

Sekvensiell-inkrementell:
Vi er en gruppe på 3 personer som ikke nødvendigvis alltid befinner oss på samme sted, derfor er vi avhengig at prosjektet kan brytes ned i flere forskjellige deler eller sekvenser som hver person kan jobbe med separat. Etter at individuelt arbeid er utført og godkjent av de andre vil det bli innlemmet inn i produktet. Ved å jobbe på denne måten støtter vi også en gjenbruksorientert modell, fordi vi har muligheten til å gå tilbake til en bestemt del av koden å gjøre forbedringer senere. Denne modellen gir også mulighet til å fokusere på en ting om gangen, slik at vi ikke nødvendigvis må lære alt på en gang, og kan komme i gang med prosjektet mye fortere.

Gjenbruksorientert systemutviklingsmodell:
Vi skal sy sammen de forskjellige Raspberry Pi'ene, og trenger derfor ikke utvikle all kode fra bunnen av. Det finnes mye åpen kildekode som vi kommer til å ta i bruk, og deretter kombinere dette sammen på en ny måte slik at det endelige produktet oppfører seg på en vi ønsker, i henhold til oppgaven.

Plan for statusmøter og beslutningspunkter i perioden
Det gjennomføres 30-60 minutters veiledningstimer med veileder og oppdragsgiver hver mandag klokken 12:30. Her vil vi gå igjennom det som er gjort til nå/ siden forrige møte og det er her vi vil ta beslutninger for veien videre fremover. Av disse vil 30 minutter være med både arbeidsgiver og veileder og opp til 30 minutter med bare veileder.

Organisering av kvalitetssikring

Dokumentasjon, standardbruk og kildekode
Kildekode og annet lignende vil lagres på GitHub[7]. Dette lagres i et privat repository som kun gruppens medlemmer har tilgang til.

Konfigurasjonsstyring
Konfigurasjonsstyring og versjonskontroll utføres ved hjelp av git. Selve rapporten skrives i Overleaf[8] som er knyttet til et repository der gruppens medlemmer har tilgang og kan clone ned til egne maskiner. På denne måten vil vi ha en kopi i Overleaf, en versjon på Github og en versjon på hver av medlemene sine maskiner.

[7] "GitHub." https://github.com/. Åpnet 11 feb.. 2019.
[8] "Overleaf, Online LaTeX-redigeringsprogram." https://no.overleaf.com/. Åpnet 11 feb.. 2019.

Risikoanalyse på Teknologi, Foretningsmessing, Prosjektgruppemessig (identifisere, analysere, tiltak, oppfølging)

Type Risiko:	Foretning.
Hva er risikoen:	Totalprisen sprenger budsjettet.
Sannsynlighet:	Middels.
Konsekvens:	Lav.
Tiltak:	IIK er fleksibel på budsjett, så i følge oppdragsgiver er det ingen krise hvis budsjettet sprenges.

Type Risiko:	Prosjekt.
Hva er risikoen:	Utstyr blir stjålet/ borte.
Sannsynlighet:	Lav.
Konsekvens:	Høy.
Tiltak:	Lås inn utstyr på et trygt sted når det ikke er i bruk, eller spre risiko ved at alle på gruppa har med seg noen Raspberry PI hver seg hjem.

Type Risiko:	Teknologi.
Hva er risikoen:	Komponenter på Raspberry PI snakker ikke sammen.
Sannsynlighet:	Lav/ Middels.
Konsekvens:	Middels.
Tiltak:	Gjør research på komponentene og vær sikker på at de snakker sammen før vi kjøper utstyr. Bygge en først som proof-of-concept for å bevise at hva vi prøver å få til faktisk er mulig.

Type Risiko:	Prosjekt.
Hva er risikoen:	Overholder ikke deadlines/ endelig deadline 20. mai.
Sannsynlighet:	Lav.
Konsekvens:	Høy.
Tiltak:	Ha en strukturert plan fremover (f.eks Gantt-diagram), og ha denne prosjektplanen/ gruppereglene tilgjengelig som et hjelpeverktøy til å unngå at frister overskrides.

Type Risiko:	Foretning.
Hva er risikoen:	Gruppen blir svindlet ved kjøp av utstyr fra utlandet (f.eks varen sendes aldri, murstein i pakka i stedet for Raspberry Pi, etc).
Sannsynlighet:	Lav/Middels.
Konsekvens:	Høy.
Tiltak:	Denne risikoen er det ikke så mye vi får gjort noe med. Vi må stort sett stole på at NTNU har gode nok bestillingsrutiner som kan avdekke om et nettsted fra utlandet som baserer seg på salg er en legitim og troverdig selger. Også se på tilbakemeldinger på sider vi bestiller fra etter misfornøyde kunder, ved mange av disse se etter et annet sted å kjøpe utstyr fra.

Type Risiko:	Prosjekt.
Hva er risikoen:	Et gruppemedlem blir syk over lengre tid.
Sannsynlighet:	Lav.
Konsekvens:	Middels.
Tiltak:	Det er ikke mulig å forutse om et gruppemedlem vil bli langtidssyk. Ingen av gruppens medlemmer kjenner til noe spesifikk historikk som indikerer at en av gruppemedlemmene må holde seg borte over lengre tid på grunn av sykdom.

Type Risiko:	Prosjekt.
Hva er risikoen:	Veileder blir syk.
Sannsynlighet:	Lav.
Konsekvens:	Lav.
Tiltak:	Hvis veileder er syk en dag, spesielt der det er veiledningsmøte, har gruppen andre kanaler man kan benytte seg av for å holde kontakt med veileder. Vi anser ikke dette som en alvorlig konsekvens, siden gruppen kan snakke med veileder senere i uka eller på neste veiledningsmøte når veileder er tilbake igjen. Vi kjenner ikke til at veileder har en historikk som tilsier at vedkommende kommer til å bli borte over lengre tid.

Type Risiko:	Prosjekt.
Hva er risikoen:	Oppdragsgiver blir borte og/ eller mister interessen for

	oppgaven.
Sannsynlighet:	Lav.
Konsekvens:	Høy.
Tiltak:	Det er ingen indikasjoner på at at oppdragsgiver vil forlate vårt arbeid midt i prosessen. Oppdragsgiver har vært en del av fagmiljøet på Gjøvik i over 20 år. Vi anser at denne ikke risikoen vil slå til, men konsekvensene ville ha vært høye hvis det skulle skje.
Type Risiko:	Prosjekt.
Hva er risikoen:	En eller flere av gruppemedlemmene bidrar ikke i arbeidet.
Sannsynlighet:	Lav.
Konsekvens:	Høy.
Tiltak:	Medlemmene har gruppereglene å forholde seg til, som også inneholder konsekvenser for brudd på regler. Denne vil bli benyttet hvis en eller flere av gruppemedlemmene ikke bidrar, sluntrer unna, etc.

PLAN FOR GJENNOMFØRING

Gantt-skjema

Se vedlegg.

Liste over aktiviteter (Work Breakdown Structure)

Se vedlegg.

Milepæler og beslutningspunkter

18. Januar: Levert handleliste til instituttet, som gjør bestilling av utstyr.

21. Januar: Levert prosjektplan til veileder.

Starten av mars: Første prototype/ design av produktet vises for oppdragsgiver og veileder.

Ca midten/slutten av mars: Andre prototype/ design av produktet vises for oppdragsgiver og veileder.

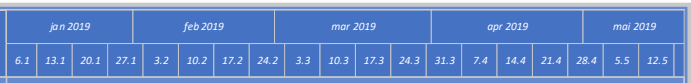
Innen påskeferien starter: Sluttversjon av produktet ferdig/ så og si ferdig, i tillegg til et utkast av rapport.

15. Mai: Sluttrapport innleveres (siste frist).

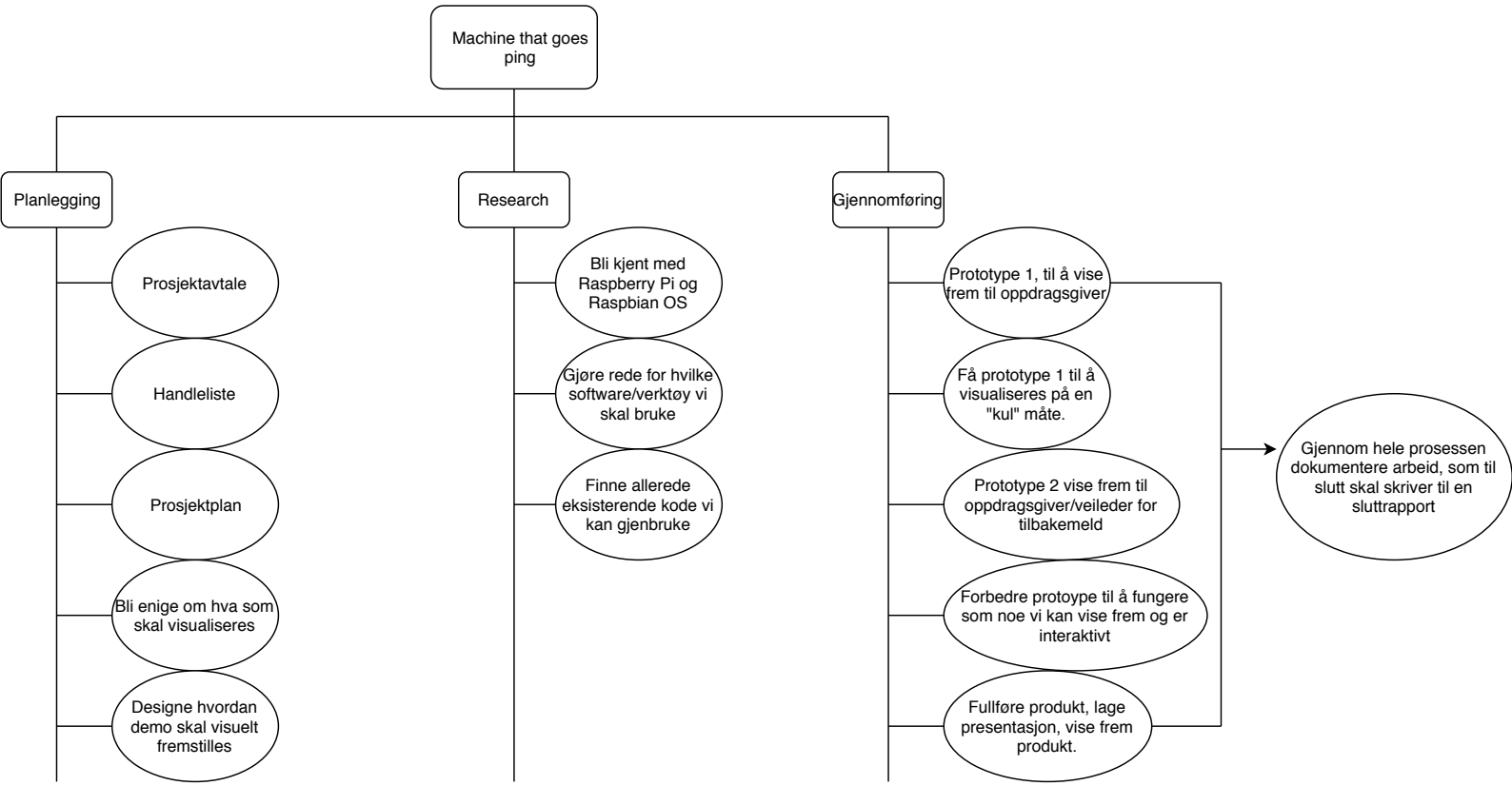
F.2 Gantt-skjema (Norwegian)

Vedlegg: Gantt-skjema
(revidert 28.01.2019)

ID	Task Name	Start	Finish	Duration
1	Bacheloroppgave	07.01.2019	20.05.2019	96d
2	Prosjektavtale	07.01.2019	07.01.2019	1d
3	Handleliste	14.01.2019	18.01.2019	5d
4	Prosjektplan	09.01.2019	21.01.2019	9d
5	Bli kjent med Raspbian OS (GUI/CLI, programmering, etc..) (Enten på VM (virtualbox, etc), eller native på Raspberry Pi	21.01.2019	15.02.2019	20d
6	Prototype 1: Installer Raspbian OS og få det til å fungere	28.01.2019	01.02.2019	5d
7	Prototype 1: Koble sammen skjermer, få dem til å snakke med Raspberry PI og vise visuell output	04.02.2019	08.02.2019	5d
8	Prototype 1: Logikk på Raspberry PI: f.eks visualisere brute force passordknekking, skalerbarhet, robusthtet, ...	11.02.2019	22.02.2019	10d
9	Prototype 1: Koble til eksterne høyttalere, få dem til å snakke med Raspberry PI og generer ut lyd fra Raspberry PI	11.02.2019	15.02.2019	5d
10	Prototype 1 (vis for oppdragsgiver og veileder påfølgende veiledningsmøte etter denne oppgavens slutt)	28.01.2019	25.02.2019	21d
11	Prototype 2 (vis revidert versjon for oppdragsgiver og veileder påfølgende veiledningsmøte etter denne oppgavens slutt) (mer spesifikke oppgaver for prototype 2 kommer etter hvert)	25.02.2019	18.03.2019	16d
12	Prototype Sluttversjon (2stk Ping-machine skal leveres oppdragsgiver. En skal være i Gjøvik, den andre skal til Trondheim) (mer spesifikke oppgaver for sluttversjon kommer etter hvert)	18.03.2019	12.04.2019	20d
13	Planlegge, designe, og utføring av video (opptak + redigering) (mer spesifikke oppgaver kommer etter hvert)	18.02.2019	12.04.2019	40d
14	Planlegge, designe, og utvikling av presentasjon/demo (mer spesifikke oppgaver kommer etter hvert)	18.02.2019	12.04.2019	40d
15	PÅSKEFERIE	15.04.2019	23.04.2019	7d
16	Ekstra tid på produktet hvis behov	23.04.2019	06.05.2019	10d
17	Skrive sluttrapport (Grunnlag for karakter)	01.04.2019	20.05.2019	36d



F.3 Work Breakdown Schedule (Norwegian)


Machine that goes ping
Planlegging
Prosjektavtale
Handleliste
Prosjektplan
Bli enige om hva som skal visualiseres
Designe hvordan demo skal visuelt fremstilles
Research
Bli kjent med Raspberry Pi og Raspbian OS
Gjøre rede for hvilke software/verktøy vi skal bruke
Finne allerede eksisterende kode vi kan gjenbruke
Gjennomføring
Prototype 1, til å vise frem til oppdragsgiver
Få prototype 1 til å visualiseres på en "kul" måte.
Prototype 2 vise frem til oppdragsgiver/veileder for tilbakemeld
Forbedre protoype til å fungere som noe vi kan vise frem og er interaktivt
Fullføre produkt, lage presentasjon, vise frem produkt.
Gjennom hele prosessen dokumentere arbeid, som til slutt skal skriver til en sluttrapport

F.4 Grupperegler (Norwegian)

Prosjektorganisering - Ansvarsforhold, roller, rutiner og regler.

Ansvarsforhold og roller

Referatansvarlig: Trond Håvard Thune
Prosjektleder: Per-Kristian Kongelf Buer
Innkjøpsansvarlig: Martin Brådalen

Rutiner og regler i gruppa

Møter med veileder og oppdragsgiver:
Det gjennomføres 30-60 minutters veiledningstimer med veileder og oppdragsgiver hver mandag klokken 12:30, unntatt i påsken.

Gruppearbeid:
Felles arbeidstid på campus bestemmes fra uke til uke, siden gruppens medlemmer også har andre emner og gjøremål som gjør at vi ikke kan legge statiske føringer på hvordan ukene våre ser ut.

På ulike tidspunkt i uken vil en eller flere av gruppens medlemmer være opptatt med andre emner, eller emner der gruppens medlemmer er læringsassistent i. Under disse tidspunktene vil de resterende av gruppens medlemmer gjøre separate oppgaver. Hver mandag før veiledningsmøte vil gruppen gå igjennom det alle har gjort hver for seg og sy dette sammen til en helhet.

Gruppen jobber med bacheloroppgave sammen i gruppe
hovedsakelig på mandager, onsdager og torsdager. En eller
flere av gruppens medlemmer er opptatt med andre emner tirsdager og fredager.
Det forventes at hvert gruppemedlem bruker mellom 20-30 timer i uken på bacheloroppgaven. Antall timer dokumenteres på egenhånd ved hjelp av toggl (referanse her). Antall timer vil i utgangspunktet ikke bli dokumentert pga personvern, men en ukentlig logg over hva hvert enkelt har gjort forventes.

Fravær:
Ved sykdom gis det beskjed til de andre hva den syke hadde tenkt å gjøre, slik at de gjenværende kan diskutere seg imellom om det er nødvendig å fordele de arbeidsoppgavene mellom seg, eller om det kan vente til den syke er tilbake.

Oppbevaring og sikring av fysisk utstyr:
Fysiske deler av prosjektet vil bli splittet mellom medlemmer for å minske risiko for at ting blir stjålet
Sikring (backup) av prosjektrapporten:

Prosjektrapporten skrives i ShareLaTeX/Overleaf (et online latex-miljø). Prosjektrapporten er linket mot Per-Kristians fork av GitHub-repoet som malen til prosjektrapporten er basert på. Innholdet pushes periodisk fra ShareLaTeX/Overleaf til dette GitHub-repoet. Derfra kan hver av medlemmene laste ned en kopi med git clone / git pull.

Loggføring
Alle individuelle og/ eller gruppebaserte beslutninger og oppgaver som er utført skal det føres logg av. Det loggføres i en dedikert mappe i gruppens Google Docs.
Det trenger ikke å være en formell tekst som skrives, men loggen skal inneholde den individuelles/ gruppens tanker og refleksjon mot de valgene som er gjort i utførelsen av oppgavene. Her skal det blant annet opplyses om begrunnelser for valg (hvorfor vi til slutt bestemte oss for å gjøre A i stedet for B og C, etc). Loggene danner grunnlag for skriving av endelig sluttrapport som leveres den 20. mai 2019.

Kvalitetssjekk på arbeid
Alt individuelt arbeid som vil tilføyes til sluttprodukt må først gjennomgås og godkjennes av de andre gruppemedlemmer før det tilføyes til produktet.

Sanksjoner mot brudd på rutiner og regler:
Ved kontinuerlig brudd av grupperegler av en eller flere medlemmer kan prosjektleder innkalle til møte hvor brudd og eventuelle sanksjoner kan diskuteres mellom den tiltalte og de resterende medlemmer. Hvis interne tvister ikke kan løses innad i gruppen, er det mulighet for å tilkalle veileder til å ta avgjørelser som en ekstern, upartisk stemme.

Brudd på regler fører til en advarsel. Det vil bli gitt opptil og inkludert tre advarsel. Etter dette kan videre brudd føre til alvorligere konsekvenser, i verste fall utestengelse fra gruppen.

Jeg samtykker i å ha lest og forstått gruppens regler og rutiner for utførelse av bacheloroppgave, og erkjenner at brudd på reglementet vil gjøre det mulig for de resterende i gruppen å utføre sanksjoner mot meg.

Sted, Dato, Signatur:

Sted: Gjøvik Dato: 11/2 Signatur: Martin Brådalen

Martin Brådalen

Sted: GJØVIK Dato: 11/02-19 Signatur: Per-Kristian K Buer

Per-Kristian Kongelf Buer

Sted: Gjøvik Dato: 11/2-19 Signatur: Trond Håvard Thune

Trond Håvard Thune