Bachelor's project

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

Gaute Hiis-Hauge
Magnus Walmsnæss Refsnes
Nick Zakharov
Raymond Aardalsbakke

Out-of-Band Management with Redfish and Ansible

Bachelor's project in IT-drift og informasjonssikkerhet
Supervisor: Ernst Gunnar Gran

May 2021

NTNU
Kunnskap for en bedre verden

Gaute Hiis-Hauge
Magnus Walmsnæss Refsnes
Nick Zakharov
Raymond Aardalsbakke

Out-of-Band Management with Redfish and Ansible

Bachelor’s project in IT-drift og informasjonssikkerhet
Supervisor: Ernst Gunnar Gran
May 2021

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

NTNU
Kunnskap for en bedre verden

Abstract

The Idun high performance computing group at NTNU is in charge of an ever growing cluster of servers which forms a high performance computing environment. These servers are managed through a small piece of hardware named the baseboard management controller (BMC). The BMC allows the Idun group to manage and monitor the servers hardware and BIOS remotely, independent from any operation system (OS). To interface and manage the BMC a specification known as Intelligent Platform Management Interface (IPMI) and its related tools are used. This specification is rather dated, and with limited functionality across vendors. The Idun group has started to look for new tools and specifications which scale better in a growing environment, and they have become curious about a new specification named Redfish. The Idun group commissioned a study into the possibilities of Redfish alongside Ansible to solve several remote server management tasks.

This thesis is a dive into the Redfish specification, and the functionality which is available to it. It presents a Proof of Concept (PoC) on the functionality and ability of Redfish in the remote management of BMCs, it also incorporates the orchestration tool Ansible to show how this specification can be used in automating server management. The PoC and later demonstration illustrates how the tasks presented by the Idun group can be solved using Redfish and Ansible. Essentially the PoC provides a reference for the functionalities of Redfish both alone and when used with Ansible, and how it can be implemented to manage a more diverse server environment.

Sammendrag

Iduns operatørgruppe ved NTNU styrer et voksende samling av høy-ytelses med servere som former et 'high performing computing' (HPC) miljø. Disse serverene styres gjennom en baseboard management controller (BMC) - en maskinvare som er innebygd i de fleste serverne. BMC'en tillater Idun-gruppen å fjernstyre og monitorere serverens maskinvare og BIOS uavhengig av hvilket operativsystem (OS) som er installert. Som grensesnitt for håndtering av BMC'en, blir Intelligent Platform Management Interface (IPMI) brukt. Spesifikasjonen er ganske utdatert, og med begrenset funksjonalitet på tvers av leverandører. Idun-gruppen har startet å se på andre verktøy og spesifikasjoner som skalerer bedre i et voksende miljø, og har dermed begynt å bli nysjerrige på en ny spesifikasjon som heter Redfish. Idun-gruppen kommisjonerte en studie om mulighetene til Redfish sammen med orkestrerings-verktøyet Ansible for å gjennomføre flere arbeidsoppgaver som innebærer fjernstyring av servere.

Denne rapporten er en fordypelse inn i Redfish spesifikasjonen, og dens funksjonalitet. Den presenterer et 'Proof of Concept' (PoC) på funksjonaliteten og evnene til Redfish ved fjernstyring av BMC, og knytter inn orkestrerings verktøyet Ansible for å vise hvordan denne funksjonaliteten kan bli automatisert. PoC'et og andre demonstrasjoner illustrerer hvordan oppgavene presentert av Idun gruppen kan bli løst ved hjelp av Redfish og Ansible. PoC'et brukes for fremstille en referanse av funksjonalitetene til Redfish både alene, ved hjelp av Ansible, og hvordan det kan bli implementert for å håndtere et server miljø med flere produsenter.

preface

We would like to acknowledge and extend our deepest gratitude to our supervisor Ernst Gunnar Gran, who helped guide us on the right track throughout the project and provided thorough and valuable input on reviews of the thesis drafts. We would also like to thank our employer, the Idun HPC group at NTNU, for the assignment, and for providing and setting up the equipment needed to build a testing-environment for this thesis. Thanks to Einar Næss Jensen for being our contact person throughout this project, and meeting with us regularly, being available for technical question, and together with Pavlo Khmel running to the data center to bring up connectivity again after we brought it down due to (unintentional) misconfigurations.

Contents

Acronyms

API	Application Programming Interface.
BIOS	Basic Input/Output System.
BMC	Baseboard Management Controller.
CA	Certificate Authority.
CI	Continuous Integration.
CIFS	Common Internet File System.
CLI	Command Line Interface.
CM	Configuration Management.
CSDL	Common Schema Definition Language.
DHCP	Dynamic Host Configuration Protocol.
DMTF	Distributed Management Task Force.
GPU	Graphics Processing Unit.
GUI	Graphical User Interface.
HPC	High Performance Computing.
HPE	Hewlett Packard Enterprise.
HTTPS	Hypertext Transfer Protocol Secure.
IaC	Infrastructure as Code.
ICMB	Intelligent Chassis Management Bus.
iDRAC	Integrated Dell Remote Access Controller.
IETF	Internet Engineering Task Force.

IPMI Intelligent Platform Management Interface.

LAN Local Area Network.

LDAP Lightweight Directory Access Protocol.

NEC Nippon Electric Company.

NFS Network File System.

NIC Network Interface Controller.

NTNU Norwegian University of Science and Technology.

OData Open Data Protocol.

OEM Original Equipment Manufacturer.

OMSDK OpenManage Python Software Development Kit.

OOB Out-of-Band.

OS Operating System.

PCI Peripheral Component Interconnect.

PoC Proof of Concept.

RAKP RMCP+ Authenticated Key-Exchange Protocol.

REST Representational State Transfer.

RFC Request for Comments.

RMCP Remote Management Control Protocol.

SCM Software Configuration Management.

SCP Server Config Profile.

SoL Serial over LAN.

TLS Transport Layer Security.

UDP User Datagram Protocol.

URI Uniform Resource Identifier.

VPN Virtual Private Network.

Figures

List of Listings

Chapter 1

Introduction

1.1 Background

NTNU hosts the Idun cluster, which is a combination of computing resources to provide a testing environment for high performance computing (HPC) [1] software for faculties at NTNU [2]. A HPC cluster is essentially several servers working as one, complex, very fast computer, which is capable of processing computing tasks like no ordinary computer can. The Idun cluster itself is built up from 100 separate servers, while in the rest of NTNU there are about 600 other servers which are also used for HPC purposes. Additionally the HPC group manages several other HPC environments at the national level. Those consist of about 3000 to 4000 servers in total [A].

All of the mentioned HPC environments above are managed by the IT Department's HPC group using the Intelligent Platform Management Interface (IPMI) [3] - which provides an interface to Baseboard Management Controller (BMC). The BMC is a piece of hardware which is most often integrated into the server, which allows to interact with other hardware components to monitor the server status, such as temperature, voltage, power state, etc.

The HPC group regularly receives new servers which are to be added to the Idun cluster. These come with a very basic configuration from the manufacturer. The IT department has to update the firmware of the servers, and set predefined or custom profiles of configurations to enable and optimize the performance needed for high performance computing [4]. After the server is added to the cluster, it also needs to be monitored for performance and maintenance.

1.2 Problem area

Up until now the process of setting-up, configuring, and adding a server to the cluster is handled manually by the administrators of the group. They have to locally configure these new server additions for the HPC cluster, though subsequent updates can be done remotely. The standard utilized for configuration is IPMI 2.0 [5], which dates back to 2004, not counting revisions. This makes it an almost two decades old standard, which was designed for a very different period of computing.

To ensure cross-vendor functionality, the IPMI specification commands are limited to the least common denominator such as power off, power on, and temperature. More complex BMC interactions are unique, and does not function across vendor types. Essentially this makes it more difficult to manage diverse server environments made up of different vendor servers.

Setting up and managing servers is therefore a time-consuming exercise. The HPC group receive new servers each month, and configurations are done in an ad-hoc manner. It steadily becomes more difficult to maintain a consistent configuration across servers, resulting in configuration drift.

This is where technology such as Redfish [6] comes in. Redfish is a standard which offers greater functionality than IPMI and can be paired with the preferred automation tool of the IT department, Ansible [7]. Though this presented the department with another problem, as they were unfamiliar with the capabilities of Redfish and its compatibility with their environment. They wanted a greater understanding of the Redfish standard and to uncover the compatibility before implementing any changes in the infrastructure management.

To address these challenges of the IT department, this thesis explores the usage of the functionalities of Redfish and its usage in combination with Ansible.

1.3 Project goals

The goals of this project is divided into effect goals and result goals.

1.3.1 Effect goals

The effect goals describes the desired long term effects on the HPC environment.

- Reduce the amount of time spent on setting up newly arrived servers both to the HPC environment and also possibly to the IT Department as a whole.
- Reduce configuration drift in the production environment.

1.3.2 Result goals

The result goals describes the achievements which the project and thesis are to achieve by the end of the project timeline.

- A study of the viability of using Redfish and Ansible in an HPC environment for the setup and configuration of new server additions to the cluster.
- A Proof of Concept of secure configuration of servers for an HPC environment using Redfish and automated with Ansible.

1.4 Target audience

The main target audience of this thesis is the Idun HPC group at NTNU, as they are the employers of this project. Though it is also of interest to the other parts of the IT department [A], as the findings could be used for their benefit as well. Other system administrators that manage IPMI-based servers may also be interested in this thesis for the theory surrounding Redfish; and to see a Proof of Concept before testing the transition to Redfish and automating management with Ansible themselves.

1.5 Scope

The scope for this project is to provide the NTNU Idun group with a Proof of Concept (PoC) of Redfish and Ansible for configuration management. This PoC is intended to demonstrate the capabilities and functions of the Redfish specification, and also how it can be used alongside Ansible in the management of servers. This thesis is to present the ability of Redfish alongside Ansible to do the following tasks as requested by the employer:

- Read and save Integrated Dell Remote Access Controller[1] (iDRAC) settings from one server to another using Redfish
- Write settings to iDRAC and BIOS with Redfish
- Collect support data with Redfish
- Deploy a default iDRAC based on previously gathered settings using Redfish
- Simulate a deployment of iDRAC settings from "out of the box" server
- Investigate and if possible show how Redfish and Ansible can be used together to automate the iDRAC setup
- Change the settings of multiple servers simultaneously using Redfish and Ansible
- Demonstrate the “Read, Write and Deploy” settings with Ansible and Redfish

Should Redfish and Ansible be proven to be incapable or incompatible with these types of tasks, the thesis should document the findings, and propose alternatives to achieve the aforementioned tasks.

[1]The iDRAC is Dell’s proprietary technology BMC integration to their servers. It is further introduced in chapter 3 subsection 3.2.1

1.5.1 Delimitation

The project is tackling multiple technologies, with the purpose of achieving the set goals. It is necessary to clarify the different restrictions that have been decided upon, which are set to limit the project scope to a reasonable degree.

- The use of Redfish has been tailored to the needs of the Idun group at NTNU, and does not cover all aspects and uses of Redfish.
- The PoC has been created and tested on the provided Dell Poweredge servers using iDRAC. Some Redfish functions are Original Equipment Manufacturer (OEM) implementations that will not be usable on all firmware versions or on other vendor equipment, further details are specified where it is applicable.
- Specific iDRAC/BIOS configurations for a production environment (e.g HPC optimizations, LDAP authentication), is out of scope for this project. This is due to the project being a Proof of Concept, and is meant to show the possibilities, not apply specific configurations.
- Open-source Redfish-modules in Ansible are to be used to automate the processes.
- This project and thesis has focused solely on the server functionality of Redfish, instead of touching its other networking equipment functionality. Additionally network and IP address management is also considered out of scope for this project.
- The PoC source code is to be open-source.

1.5.2 Limitations

Due to the ongoing Covid-19 pandemic the project group has been unable to travel to Trondheim to meet with the employer. This also extends to physically interacting with any of the hardware, as it is stationed in Trondheim. This has also meant that all interaction with the hardware which has required physical access has been done by the employer.

1.6 The Project group

The project group consists of four third year students at the BITSEC (IT-Operations and Information Security) course at NTNU Gjøvik. They have all taken the same elective subjects, the most relevant for this project being ‘Infrastructure as Code’, where together, they worked with the automation of infrastructure deployment with Openstack HEAT [8] and Puppet [9] to make an automatically scaling service based on set metrics. They also have general knowledge about networking, operating systems, Linux, programming, and IT operations/methodologies.

Magnus Walmsnæss Refsnes - Project Leader
Nick Zakharov - Deputy Project Leader
Gaute Hiis-Hauge - LaTeX Expert
Raymond Aardalsbakke - Secretary

1.6.1 Other parties

The two other related parties are the employer, which is an employee at the NTNU IT department and is our liaison to the Idun group. The project supervisor designated by NTNU helps guide the student group through the thesis and project.

Einar Næss Jensen - Employer
Ernst Gunnar Gran - Project Supervisor

1.6.2 Thesis structure

A short explanation of each chapters contents:

1. Introduction - An introduction to the project explaining the case, goals, scope and involved parties.
2. Background - A chapter detailing relevant theory required for understanding subsequent chapters.
3. Technical Design - An overview of the development environment, its components, and how they interact.
4. Implementation - A chapter demonstrating how Redfish and Ansible playbooks can accomplish out-of-band management tasks.
5. Security - A security chapter detailing best practices for BMC management and a review of the security of IPMI and Redfish.
6. Deployment - Building on the findings of chapter 4 and 5, this chapter simulates a deployment of an out of the box server using Ansible and Redfish.
7. Testing - A look into the theory behind testing and its relevance to this project. This chapter also details the testing methods used, and the reasoning behind them.
8. Discussion - Reflections on the development process, encountered challenges and possible future work.
9. Conclusion - Final thoughts on the project
10. Bibliography - A typical bibliography
11. Appendix - An appendix which contains minutes of meetings with the employer and supervisor, code that is too long for the main document and other miscellaneous files referenced in the thesis.

Chapter 2

Background

This chapter explains the theory, terms and technologies which are needed to understand when reading this thesis. At the centre of these terms and technologies is out-of-band (OOB) management. OOB management refers to the remote management and/or monitoring of servers and other network equipment on a separate management interface and network.

2.1 Configuration Management

Generally, when the thesis writes about configuration in this report, it means a specific or a set of properties in the server BMC or BIOS settings. This thesis uses the term ’configuration management’ to describe that this report is going to relate to the management of server settings and other information exposed by the BMC and its application programming interface (API). Software Configuration Management (SCM) [10] is a different, but closely related topic to this project.

In his book, “Software Engineering (10th edition)“ [11], Sommerville describes configuration management (CM) as the policies, processes and tools for managing change in software systems. He argues that as software systems are becoming more complex, keeping track of changes is crucial to not waste efforts trying to modify the wrong version, delivering the wrong version of a system to customers, or even forgetting where the source code of a particular version is stored. The term ’configuration’ in ’configuration management’ refers to all data or information which describes the functional characteristics of a software system, i.e., source code, build data, compiler version, design and testing requirements [12].

Sommerville describes 4 activities that are closely related in configuration management of a software system [11]:

1. Version control – The process of keeping track of changes to a system by different versions, and ensuring any changes made by another developer does not cause any conflicts.
2. Change management – Involves keeping track of the requests for changes by customers and developers and deciding whether they should be implemented or not.
3. System building - The process of assembling program components, data and libraries into an executable system
4. Release management – Preparing software for release and keeping track of which versions have been released for public and/or customer use.

These activities are a more abstract way to describe CM best practices as they are defined in standards such as ISO10007 – “Quality Management - Guidelines for configuration management” [13] and IEEE 828-2012 [12].

Essentially the Idun HPC is a growing cluster of servers with new additions every month. There is a need for time efficient, practical, and secure configuration management. This thesis revolves around the technologies and tools which can make this possible, specifically the Redfish standard and the orchestration tool Ansible.

2.2 Infrastructure as Code

Software configuration management becomes relevant in IT operations because of an approach in infrastructure management called Infrastructure as Code (IaC) [14]. IaC is an approach to infrastructure automation and provisioning with the use of code, and it is enabled by modern tools and services that treats infrastructure as if it was software and data.

An example of this is cloud computing, where you can define and provision system resources (compute, storage, network) by declaring them in text-files with a standard format such as JSON, YAML or XML [14] which is treated like the project source code. Their deployment is often automated by well-known orchestration tools such as Ansible and Puppet [9].

The principles of IaC revolves around how systems should be easily reproducible, disposable, and consistent [14], and becomes relevant to this project as they can be applied when automating server management with Ansible and the Redfish standard.

2.3 Intelligent Platform Management Interface

It would be impossible for system administrators in datacenters with hundreds or thousands of servers to personally manage each and every server if they had to be physically present on each of them. Instead, a standardized sub-system is embedded in the servers to enable OOB management. The most common system is called Intelligent Platform Management Interface (IPMI) [15]. IPMI is led by Intel and was released in 1998, it has been supported and developed in collaboration with a number of vendors such as Dell [16], Hewlett Packard Enterprise[17] and NEC [18][19]. The main component is the baseboard management controller (BMC) - a specialized service processor responsible for monitoring and controlling all the manageable components in the system [20].

IPMI solved the problem system administrators had at the time; there was no common model for system management, and OOB management was proprietary implementations from vendors (e.g. Dell Remote Access Card) [19].

In practice, IPMI provides an API to hardware components, using a message-based system. The following interfaces are supported for communication:

- System interfaces (local)
- Serial/modem
- Local Area Network (LAN)
- Intelligent Chassis Management Bus (ICMB) [21] and PCI Management Bus [22].

Each interface uses different protocols for communication, requiring specific message-formats, and is therefore categorized in individual channel numbers for configurations allowing the direct communications between the BMC and the interface. For example, the LAN interface will often (specific to implementation) be on channel 1, specifying how IPMI messages can be transmitted in the Remote Management Control Protocol (RMCP/RMCP+) UDP datagrams [20]. A subsystem with a separate power connection and network interface card (NIC), will allow remote management through the LAN channel even if the host system is powered off or does not have an operating system installed yet.

System administrators can use tools like ipmitool, ipmiutil, freeipmi, or openipmi [23] to query information from the IPMI-based system over a command-line interface on a master node. The most popular tool for this purpose is ‘ipmitool’ (ref app. C.1), which is provided as an independent package in a number of Linux-distributions [23].

After installing the package through a distributions package-provider, a standard command will follow the format as shown in listing 1.

```
ipmitool -I interface -H ipaddress -U username -P password command
```

Listing 1: Example of a command using ipmi tool

An example of this type of command is shown in listing 2 which shows the usage of the 'chassis status' command. This command provides information about the chassis, which includes information about the system power, the state of the different buttons, and if there is an issue with the power.

```
[redfish@bachelor ~]$ ipmitool -I lanplus -H 192.168.0.123 -U root -P redfish
 ↪  chassis status
System Power         : on
Power Overload       : false
Power Interlock      : inactive
Main Power Fault     : false
Power Control Fault  : false
```

Listing 2: Example of IPMI extracting chassis status (Shortened)

IPMI is becoming an old specification, and because it follows the least common denominator its functionality is very limited, meaning it only has a few sets of commands such as setting a LAN IP, power on/off/reboot, and temperature checks. OEM extensions are not interoperable, for example, if you wish to change the NIC settings of an iDRAC to 'shared', you must issue the raw command in listing 3 to access that type of functionality [24].

```
ipmitool raw 0x30 0x24 0
```

Listing 3: Example of raw input using ipmitool on iDRAC

While on a Lenovo server one would have to instead utilize the raw command in listing 4 for the same result. This would set the NIC to shared for both IPv4 and IPv6. Replacing the last hex-argument from 0x03 to either 0x01 or 0x02 would respectively change it to shared only for IPv4 or IPv6 [25].

```
ipmitool raw 0x32 0x71 0x00 0x00 0x03
```

Listing 4: Example of raw input using ipmitool on Lenovo

This archaic, non-human readable bit-mapped architecture requires data centres with a diverse, multi-vendor inventory to either develop their own tools (often relying on in-band management software), or using third-party solutions to manage all the vendor-specific extensions [26].

2.4 Redfish

As a solution to the problems and difficulties with the continued usage of IPMI, the group DMTF [27] (formerly known as Distributed Management Task Force) began to set the basis for what would be its successor, Redfish [6]. The idea was to specify models needed for a model-driven architecture as opposed to IPMI's bit-map oriented architecture [28]. The purpose was to reduce the complexity of systems through layers of abstractions.

Redfish is an open source and open standard specification for hardware management. It provides an API that can be used to obtain information and manage servers through an OOB controller. Redfish uses HTTPS for communication, which is both generally well understood and also a secure way of transferring data. The Redfish schema is defined in three formats, those being JSON schema[29], Open Data Protocol (Odata) Common Schema Definition Language (CSDL) [30], and in YAML [31][32].

The JSON representation, makes it easier for Redfish to integrate with other programming environments such as Python scripts, JavaScript code, and visualizations. While the OData CSDL, is adopted for naming conventions for descriptions, URL conventions, and definitions to provide a comparability for APIs to work together. Finally it is provided in Yaml as specified by OpenAPI. OpenAPI is an open specification on API services, and provides with a plethora of tools for users [33].

2.4.1 The Redfish API

Redfish uses a Representational state transfer application programming interface (REST API) which is an architectural style for distributed hypermedia systems [34]. It is a client to server architecture where the client uses HTTP methods (such as POST, GET, PATCH) to make requests towards the server which then answers with a response. REST is also stateless which means that each request must contain all the information that the server requires to understand and respond meaningfully.

“The key abstraction of information in REST is a resource. Any information that can be named can be a resource: a document or image, a temporal service (e.g. "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g. a person), and so on.” [35].

For Redfish these resources are what comes at the end of the requested Uniform Resource Identifier(URI), e.g "https://192.168.0.20/redfish/v1/**Systems**". In this request the resource that is requested is "Systems", which contains information about the logical system view of the computer system as seen by the operating system (processors, storage, BIOS, etc.).

In a RESTful API the resource is transferred using HTTP(s), and it can be in several formats like JSON, XLT, Python, PHP, and plain text [36]. The most commonly used format is JSON which is readable by both humans and machines. The Redfish specification uses JSON for transferring data.
A Universal Resource Identifier is a unique sequence of characters that is used to identify a logical or physical resource. The Redfish API follows a specific hierarchy for simple viewing and navigation in the following format [37]:
`https://{host IP}/redfish/v1/{Resource Path}`
Figure 2.1 gives a more practical example and explains how to read the URI:
`https://192.168.0.123/redfish/v1/Systems/System.Embedded.1`

Component part	Example
The scheme for transfer, in this case https	https//:
Where to delegate the URI to, in this case an IP address	192.168.0.123/
The service root and version	redfish/v1/
The "Systems" resource path	Systems/
Unique id of an instance of the resource	System.Embedded.1

Figure 2.1: Redfish URI

In Redfish, a URI can also represent a collection of similar resources. Redfish has something called a 'resource collection' which can describe a group of Systems, Managers and Chassis, amongst others. This is essentially an array of its members and if the array is empty, the returned JSON object will be empty.

2.4.2 Redfish examples

Following are some simple examples showcasing the simplicity of OOB management with Redfish, and giving an insight into its possibilities. Both of these examples are tested on a Dell Poweredge R720 server with iDRAC version 2.65.65.65, as well as the Poweredge C6420 server with iDRAC version 4.32.10.00.

Querying the server for information

Listing 5 contains an example that shows a GET HTTP(s) request on the iDRAC Redfish interface requesting information about the system state. The response is filtered using the command-line JSON processor tool ‘jq’ [38] with the filter ‘.Status’. This returns information about the system status like health and state.

```
[root@bachelor]# curl -sX GET -u root:calvin
 ↪   https://192.168.0.124/redfish/v1/Systems/System.Embedded.1 -k | jq
 ↪   '.Status'
{
  "Health": "OK",
  "HealthRollup": "OK",
  "State": "Enabled"
}
```

Listing 5: Example of redfish extracting health status

Running command against server

Action on a resource instance of a system is done using the HTTP POST method. For example in listing 6 there is first a GET request for the powerstate of the system, this returns that the system is currently ’On’. Then a POST request on the resource /ComputerSystem/ with the ResetType ’GracefulShutdown’ is sent. Repeating the first GET request now reveals that the powerstate is ’Off’.

```
[root@bachelor]# curl -sX GET -u root:calvin
 ↪   https://192.168.0.124/redfish/v1/Systems/System.Embedded.1 -k | jq
 ↪   '.PowerState'
"On"

[root@bachelor]# curl -k -u root:calvin -X POST https://192.168.0.124/red⌋
 ↪   fish/v1/Systems/System.Embedded.1/Actions/ComputerSystem.Reset
 ↪   -H "Content-type: application/json" -d '{"ResetType":
 ↪   "GracefulShutdown"}'

[root@bachelor]# curl -sX GET -u root:calvin
 ↪   https://192.168.0.124/redfish/v1/Systems/System.Embedded.1 -k | jq
 ↪   '.PowerState'
"Off"
```

Listing 6: Example of turning off a system with Redfish

HTTP requests like in the examples above, can be written in script-languages like Python [39] to perform more advanced series of tasks. Automation tools like Ansible uses modules written in Python which accept parameters defined in text-files to perform such managerial tasks. These modules can run on all defined hosts, bringing reliable, consistent, and automated OOB management to IT operations.

2.5 Automated OOB management with Ansible

Ansible is an IT automation engine meant for managing and automating tasks in IT infrastructures. It does not require any installation of agents on the managed nodes, only the master node is required to have a version of Ansible.

Ansible was chosen on the request of the employer as part of the initial project description, this is due to Ansible being the primary automation tool utilized by the department. In addition, Ansible has several working Redfish based modules [40][41] which can be used to solve the different tasks presented by the employer.

2.5.1 Modules

Ansible [7] manages its nodes through a concept known as modules, which are units of code executed through the command line or through a playbook. Modules support arguments and are also referred to as ‘task plugins’ or ‘library plugins’. A module is essentially a small program usually pushed and/or executed through SSH by Ansible which deletes itself after running. Redfish modules instead uses the local connection of the master node to make HTTP(s) requests on the target hosts REST API. Ansible does not require a dedicated database or server, as its library of modules can reside on any machine within the network infrastructure [42].

2.5.2 Inventory files

Inventory files exist to keep track of all your known hosts while also grouping and assigning variables to said hosts. Inventory files also allow for groups to define values by inheritance from a parent group to a child. Multiple inventory files can be used at the same time, which is useful when dealing with a fluctuating environment. The most used file formats are INI and YAML [43]. Listing 7 is an example that shows an inventory file in the INI file format. The first line declares a group, named ’myhosts’, followed by two hosts with a declared URI. After the group is defined, host-variables are set, in this case `user_name` is set to ’root’ and `user_password` is set to ’calvin’.

```
[myhosts]
host1  baseuri="192.168.0.123"
host2  baseuri="192.168.0.124"

[myhosts:var]
username="root"
password="calvin"

[all:children]
myhosts
```

Listing 7: Example of an INI inventory file

2.5.3 Playbooks and plays

The modules are utilized by an Ansible playbook, which is a configuration file written in YAML. This file has the sequence of instructions needed to bring the target host configuration to the desired state. Ideally playbooks are supposed to be short and readable as most of the work is handled by the modules, however they can be rather complex with variables and conditions. Playbooks are applicable to multiple machines, repeatable, and reusable [44].

Playbooks consist of plays which in turn consist of tasks. A play has a name, followed by applicable hosts and values. After the play related values are set, tasks are defined. Tasks are given appropriate names and one or several modules with the required parameters. In the case of listing 8 the task 'Shutdown system power gracefully' is using the redfish_command module which is a part of the community.general collection, more on this in subsection 2.5.5. The indented lines after redfish_command: are parameters used by the redfish_command module. Variables in a YAML file are used with a double curly brackets as such {{ variable }}, and in this case the variables URI, username and password which was defined in the inventory file and passed to the module.

Listing 8 is an example of a play detailing a shutdown through Redfish. A YAML file always starts with '---', which is a document separator. In this specific example the play and task has the same name: 'Shutdown system power gracefully', names does not need a specific value, but will be printed when running a play. Hosts are set to 'all', meaning **all** hosts in an inventory file will run said play. 'Connection: local' is used to run the playbook locally instead of connecting over SSH. If 'gathering_facts' is set to true it will gather various information about the remote host. Some playbooks depend on up-to-date information, in this case it is set to 'false' as the extra information is not required.

```
---
- name:  Shutdown system power gracefully
  hosts: all
  connection: local
  gather_facts: False

  tasks:
  - name: Shutdown system power gracefully
    community.general.redfish_command:
      category: Systems
      command: PowerGracefulRestart
      baseuri:  "{{ baseuri }}"
      username: "{{ username }}"
      password: "{{ password }}"
```

Listing 8: Example of a YAML play file

Listing 9 shows the output of running the play in listing 8. It shows that the task in the play has been successful on all hosts. `[root@bachelor]` is the master node that runs the ’`ansible-playbook`’-command, ’playbooks/shutdown_system_power.yml’ is path to the play being executed and ’-i inventory’ defines the inventory that is being used. Below the command which executed the play is the output. The output shows the name of the play that is being executed, as well as the task(s). The module returns ’changed’ because the power state of the system has been changed. The play recap shows a summary per host, returning ’ok=1’ to signal its success and ’changed=1’ to signal that a change on the system has been applied.

The ’`ansible-playbook`’-command is run from the same directory as the Ansible configuration file (ansible.cfg), which should be located at the root directory of the Ansible project. It defines project-specific variables like relative paths to the roles-directory.

```
[root@bachelor redfish-ansible]# ansible-playbook -i inventory
 ↪  playbooks/shutdown_system_power.yml

PLAY [Shutdown system power gracefully]
****************************************************************************

TASK [Shutdown system power gracefully]
****************************************************************************
changed: [host2]
changed: [host1]

PLAY RECAP
****************************************************************************
host1                    : ok=1    changed=1    unreachable=0    failed=0    skipped=0
 ↪  rescued=0    ignored=0
host2                    : ok=1    changed=1    unreachable=0    failed=0    skipped=0
 ↪  rescued=0    ignored=0
```

Listing 9: Example of a playbook being run

2.5.4 Roles

To help manage complex playbooks, the concept of roles can be used. Roles is an Ansible functionality which groups a set of tasks, variables, and other Ansible artifacts based on a known file structure [45] to perform a specific, independent, reusable function. A role is created by creating the "roles/{rolename}" directories in the same directory as the Ansible configuration file. The relative path to the 'roles'-directory should be added to the configuration file. The minimum requirement of a role is a tasks directory with a main.yml file containing the list of tasks suitable for the role. In subsection 4.5.6 of this thesis, the complexity of a playbook is anticipated to grow when BIOS and BMC configuration has to be executed on multiple vendor servers. To help reduce that complexity, the concept of roles is used to split up the tasks based on vendor and component configurations. Another use case for roles is to group together common tasks which can then be called from other plays without having to use relative paths to include arbitrary tasks in the project-directory.

2.5.5 Ansible Galaxy

Ansible Galaxy [46] is a website for sharing collections and roles. These prepacked roles and collections are community driven and cover a wide variety of modules. Adding roles or collections from the galaxy hub is simple and easily done with a single command. Installation is performed by either specifying one or more roles(`ansible-galaxy role install {username}.{role}`), alternatively one or more collections(`ansible-galaxy collection install {username}.{collection}`). Once installed, the role/collection can be used in any playbook by specifying either {username}.{role}.{module} or {username}.{collection}.{module}.

Ansible community general

Ansible community general is a collection of modules, which contains a wide variety of the most used modules in Ansible Galaxy. Below is a full list of Redfish related modules within the general collection [40]:

- redfish_command
- redfish_config
- redfish_info
- idrac_redfish_command
- idrac_redfish_config
- idrac_redfish_info
- xcc_redfish_command

The modules which starts with Redfish are general modules which are usable on any Redfish supported systems, while those starting with iDRAC are Dell [47] specific modules, and those starting with xcc are Lenovo [48] specific.

Dell EMC OpenManage

The OpenManage module [49] is written and managed by Dell EMC. With 'OpenManage' being Dells brand name for system management applications. The OpenManage module has a few prerequisites, and a limited amount of platforms which are supported, as seen in figure 2.2 and 2.3. OMSDK is a Python library written by Dell for automation of Dell servers which is required to run OpenManage.

Software	**Version**
Ansible	>= 2.10.0
Python	>=2.7.17 or >=3.6.5
OMSDK	any

Figure 2.2: Dell EMC OpenManage pre-requsities

Platform	**Version**
iDRAC 7/8	2.70.70.70 and above
iDRAC 9	4.32.10.00 and above
Dell EMC OpenManage Enterprise versions	3.4 and above
Dell EMC OpenManage Enterprise-Modular versions	1.20.00 and above

Figure 2.3: Dell EMC OpenManage Supported Platforms

2.5.6 Ansible Vault

Ansible vault serves the purpose of encrypting sensitive content, for example files, variables, passwords, keys, etc [50]. An encrypted file or variable would only be editable with the correct password and vault. Encryption is fairly easy and can be done with a single command such as in listing 10. Ansible vault can also be used for encrypting strings which can replace plaintext variables in an Ansible-file. Such encrypted strings are detected by Ansible at playbook run-time, and can be decrypted by either entering a password or passing a file containing the decryption password.

```
[root@bachelor]# ansible-vault create foo.yml
```

Listing 10: Example of encrypting a file with Ansible Vault

2.6 Alternative technology

The goal of this project, as required by the employer, is to utilize Redfish and Ansible for the setup and configuration of servers. However, alternative technology exists to both of these solutions, such as IPMI and Puppet. IPMI was never relevant to the project due to Redfish being the technology which the employer requested a thesis on.

Puppet on the other hand is a management and automation tool, though it differs from Ansible in using agents in their management. This essentially means that the managed nodes needs software in the form of a Puppet 'agent' to function. This differs to the Ansible approach which is agentless. Though this too was excluded due to Ansible being both the preferred and required automation tool by the employer.

2.7 Security

When implementing any new service or software it is important to take a look at the security of the new software, and if applicable compare it to the one in use already. IPMI is an old standard and has been revealed to have several security issues and challenges. It is therefore important to take a look at Redfish, and the security features which it utilizes and compare it to the features of IPMI.

2.7.1 IPMI security

As previously stated, IPMI is a rather old standard and though it has been revised several times, there have been multiple security issues related to it through the years. Specifically issues discovered by security researcher Dan Farmer [51], and the metasploit module and pentesting guide written by the Rapid7 Group [52]. Though both of these write-ups were written in 2013, Farmer took another, albeit short, look at IPMI in 2021 [53]. In this writeup he addressed that not much had changed with the standard in the last 8 years except from IPMI 2.0 adding support for SHA256 in their Remote Management Control Protocol (RMCP) Authenticated Key-Exchange Protocol (RAKP).

Some of the issues with the standard have been listed in the US Cybersecurity and Infrastructure Security Agency's Alert (TA13-207A) [54], which is largely based on the findings of Farmer. Amongst these security issues are:

1. IPMI passwords being saved in plaintext, and knowledge of one password giving access to all computers in a managed group.
2. Root access on IPMI BMC granting complete control over the system.
3. Certain types of traffic from and to the BMC are not encrypted.
4. Sanitizing passwords documentation is unclear.
5. Options which are enabled by default have large security issues, such as cipher 0 which allows authentication to be bypassed if the attacker knows the username and user id 1 allowing anonymous login [54].
6. Due to the way IPMI 2.0 negotiates a secure connection it allows an anonymous user to remotely get the password hash from the BMC.
7. Information leaking in the form of revealing information about the system and users to anonymous users.

With the IPMI standard having several glaring security issues, it has been up to the vendors to implement the standard differently and more securely. An example of this is the Dell's iDRAC, where they have by default disabled cipher 0, disabled the ability to login anonymously with user id 1 and with no option to enable it, and removed the support for the use of null passwords [55].

Finally a joint message [56] was published by Intel alongside Dell, Hewlett Packard Enterprise, and NEC. This message specifies that the 2013 2.0 specification is the last update to the specification, and that there is not planned another update nor should one be expected. Finally they encourage to move towards other specifications, mentioning Redfish as an alternative.

2.7.2 Redfish security

Redfish supports the use of TLS v1.1 and later versions, which lets the clients and servers send these requests encrypted instead of in plaintext. TLS is used by the specification in the form of HTTPS, and this adds a layer of protection for traveling packets in the network. Using HTTPS for transfering traffic also has another benefit which is that the traffic can be inspected by the firewall or an administrator by decrypting the traffic and then recrypting it [57]. This could significantly increase the chance of discovering an infected device, though it could also impact security if implemented poorly [58].

Redfish requires all write requests to Redfish objects to be authenticated except from the initial post operation on the service root. There are two ways of authenticating using Redfish, the first being HTTP Basic Authentication following the practices of IETF's RFC 7235 [59], in which the username and password is added to the request itself.

The second way of authentication is through the Redfish Session Login Authentication. This type of authentication lets the user send a post request to the "SessionService/Sessions"-resource using basic authentication. The response will contain an X-Auth-Token header which has a session authentication token. This token can then be used by the client to authenticate their subsequent requests [37].

Redfish also supports using LDAP and Active Directory [60] as external account providers.

Redfish also has an inbuilt privilege model for authorization, and this system uses roles with assigned privileges to control the access of the user. Essentially a role contains several privileges e.g login, ConfigureSelf, ConfigureUsers, and so on, these privileges control which resources the role is able to access and write to. In Redfish there are three predefined roles; Administrator, Operator, and ReadOnly. In addition to this one may also create custom roles and assign privileges to them. When a new user is created it must be assigned a role from either the three predefined or a custom role.

2.7.3 Comparison

IPMI was created and developed in a time period in which cyber security was less of a consideration. Through many iterations the security has been improved by both new releases of the specification, and the alterations made by the different vendors. This has not changed the fact that there are several issues with the standard. In addition to this the specification will most likely never be updated again, which means the security issues will remain. This will continue to be a debt on either the vendors or the IT administrators in disabling and repairing these functionalities and vulnerabilities.

Redfish on the other hand is a new specification which was developed with modern security practices in mind. It also does not have the same vulnerability debt which IPMI carries, and with it being actively developed one can expect vulnerabilities which are discovered to be patched. There are some issues with a new specification, for instance, finding information on both troubleshooting and best practices for implementation is difficult. Vendors such as Dell have their own best practices for implementation of their servers, and DMTF has mockups for developers.

Chapter 3

Development Environment

The following chapter details the physical lab environment as well as the Ansible project directory structure and content.This is where the reader should get an impression of what type of environment the PoC has been developed in, and detailed information about the files in the Ansible directory to tie up the theory from chapter 2.7 with the Proof of Concept demonstration in the coming chapter (4).

3.1 High Level Overview

The development environment, as shown in figure 3.1, consists of a Dell N3048 switch, two Dell R720 running iDRAC version 7, and lastly Dell C6420 servers with iDRAC version 9. These are reachable through a login server, 'bachelor-server', which is reachable using a VPN to connect to the NTNU campus network.

As shown in figure 3.1, the login node labeled "bachelor-server" works as the master node. The master node locally stores a version of the project repository with all the produced Ansible files. Logging into the master node requires a client on to the NTNU network. A VPN connection can be used for remote connection. The master node has direct access to the iDRAC NIC-interface through a switch which connects all the nodes to be managed. This allows the use of IPMI or the Redfish API for OOB management. Ansible playbooks which automate OOB management tasks is run on the master node. Figure 3.2 shows a high-level view of the interactions between a master node running Ansible Redfish modules and the nodes to be managed.

3.2 Dell Poweredge servers

The servers which have been provided are part of Dell's Poweredge server line [61], and the lab environment is composed of these types of servers. The employer

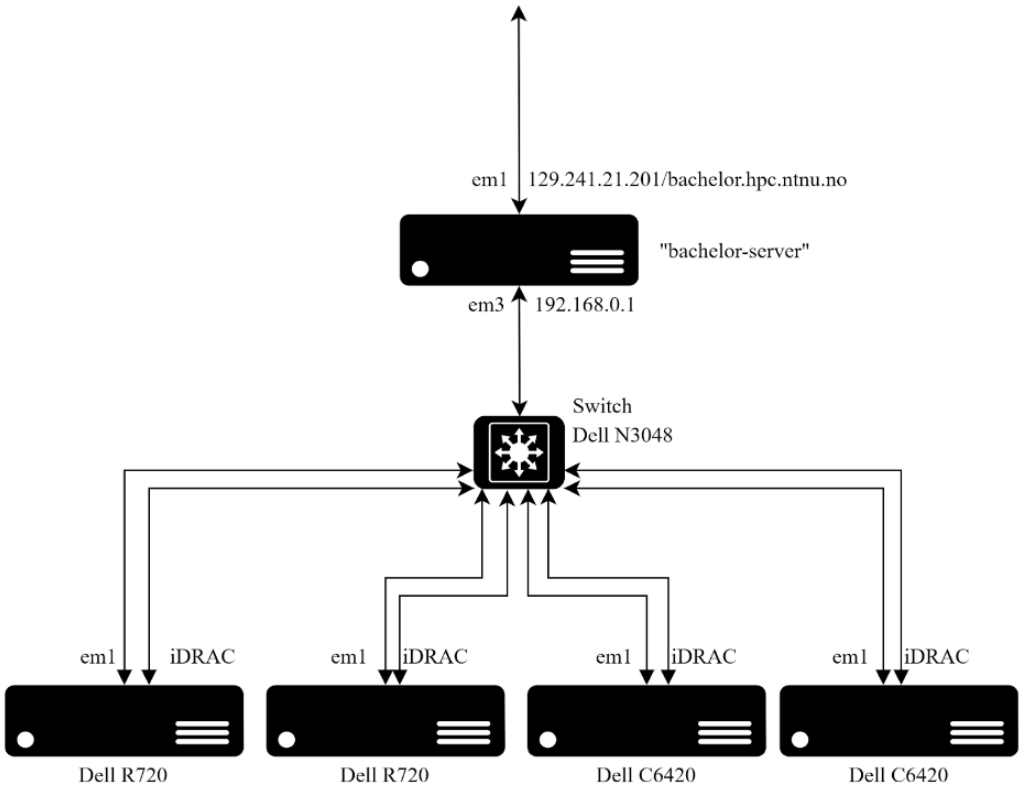


Figure 3.1: Lab environment network topology

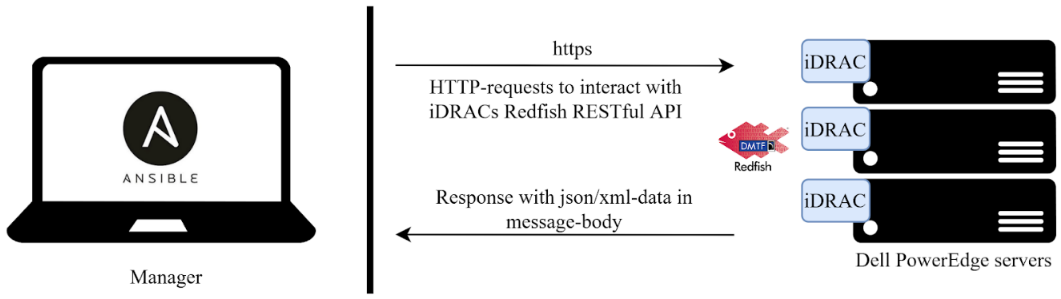

ANSIBLE
Manager
https
HTTP-requests to interact with
iDRACs Redfish RESTful API
Response with json/xml-data in
message-body
DMTF
Redfish
iDRAC
iDRAC
iDRAC
Dell PowerEdge servers

Figure 3.2: showcasing a high-level view of OOB management with Redfish

has provided the project with four Poweredge servers, two of which are of the type R720, and two of which are of the newer C6420 server line.

3.2.1 iDRAC

The Integrated Dell Remote Access Controller (iDRAC) is Dell's proprietary implementation of the BMC - a controller card which is embedded into the motherboard [62] of the Poweredge R720 and C6420 servers. This piece of hardware lets an administrator deploy, update and monitor the servers remotely [6]. The iDRAC has a web-gui which can be accessed by an administrator for management, but later versions such as the iDRAC 7-9 also utilizes the Redfish standard. This essentially means that the administrator can utilize automation software such as Ansible to make Redfish HTTPS calls, and automate processes such as deployment, updating and monitoring.

Although new releases of iDRACs come with Redfish pre-enabled, older firmware needs to be update to a firmware supporting Redfish. The firmware requirement can be seen in figure 3.3. If the appropriate firmware version is installed, the only thing that needs to be done is to enable Redfish. Redfish is enabled through either iDRACs web interface, iDRAC RACADM [63] or WSMAN [64]. However the focus of this thesis is Redfish, which is enabled by default on new hardware, which is the reason the process of enabling Redfish is not further explained.

iDRAC Version	Firmware Requirement
iDRAC7/8	2.40.40.40 or newer
iDRAC9	3.0.0.0 or newer

Figure 3.3: iDRAC firmware requirement for Redfish usage

3.2.2 SCP files

A feature specific to iDRAC is the Server Config Profile, which is a file containing parts of or a complete set of iDRAC, BIOS, NIC and RAID settings. The iDRAC settings shown in figure 3.4 are formatted as such in listing 11 when in an XML file format, additionally an SCP file can also have the JSON file format. An SCP file can be either exported or imported to an iDRAC, through web GUI or through Redfish. Model, ServiceTag and TimeStamp are values recorded at the time of export.

Component Name	Attribute	Value
iDRAC.Embedded.1	Users.2.UserName	root
iDRAC.Embedded.1	Users.2.Password	calvin

Figure 3.4: Example of system attributes

```xml
<SystemConfiguration Model="PowerEdge C6420" ServiceTag="BWX5WC3"
↪  TimeStamp="Thu Mar 18 21:04:09 2021">
<Component FQDD="iDRAC.Embedded.1">
 <Attribute Name="Users.2#UserName">root</Attribute>
 <Attribute Name="Users.2#Password">calvin</Attribute>
</Component>
</SystemConfiguration>
```

Listing 11: Example of system attributes in an XML SCP-file

3.3 Ansible

Ansible is the primary orchestration tool of the project, and it allows for interactions with one or multiple servers simultaneously. Ansible substitutes the manual construction of API calls by automating the process by listing a sequence of tasks in YAML-files. The tasks usually calls modules, which are either locally developed or downloaded through Ansible Galaxy [46]. Most modules require a specific set of parameters, Redfish modules usually require a host, credentials to said host and a module-specific commands with corresponding required variables.

3.4 Ansible directory structure

The PoC produced is an Ansible project directory stored in a maintained git-repository which lays the groundwork for future work with Redfish and Ansible by showcasing their OOB management possibilities. It is organized following Ansible best practises [65], but tailored to suit the needs of the project. For example, the Ansible documentation suggest a group-vars directory to store inventory group variables. Because only host-specific variables are needed for this project, this directory was not included. The directories and files can be modified and expanded as seen fit when adding new OOB management functionality.

Listing 12 shows a tree-graph of the current Ansible project directory named 'redfish-ansible'. The following subsections describe elements of this tree-graph to get a better understanding for the rest of the thesis.

3.4.1 ansible.cfg

Default Ansible settings can be overridden by adding an entry in the 'ansible.cfg' file. A list of settings which can be changed can be found in the official Ansible documentation [66]. In this project, the most notable entries include a path to the Ansible vault password file so that the vault-password does not have to be entered by the user on each test-run. A relative path to the roles directory is also defined so that Ansible can locate roles called from a playbook. This is the reason that '`ansible-playbook`'-commands must be run from the root 'redfish-ansible' directory. The '`ansible-playbook`'-command will look for a configuration file in the current working directory before executing playbooks, and if it is not found it will run with the default Ansible settings. As specified in the documentation, the default roles paths are '`/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles`. Unless roles are copied over to one of those locations or the relative '.roles/' path is defined, playbooks calling roles stored in the 'redfish-ansible/roles'-directory will fail.

3.4.2 host_vars

Host variables are defined in files in this directory in the following format: '{inventory_hostname}.yml'. The reason for this format is so that host-specific variables can be dynamically loaded during playbook-runtime. 'inventory_hostname' is a unique global variable defined on a per host basis in the inventory file.

Currently the only host-variables needed is the username and password for the BMC. The password-variable is encrypted using Ansible vault so that it is secure to keep in a public git-repository.

3.4.3 inventory

The inventory directory contains one file named 'static_inventory.yml' which contains all hosts in the lab-environment. The hosts gets assigned a hostname which can be referenced in ansible-playbooks, as well as the host-variable 'ansible_host' which contains the IP address of the host. Several testing-groups were created to test playbooks on specific hosts instead of running the playbooks on all hosts every time.

3.4.4 playbooks

This folder contains all playbooks referenced in this thesis.

3.4.5 plugins

Locally developed modules have to be stored in 'plugins/modules' so that Ansible can find them when they are referenced in a playbook.

3.4.6 roles

Directory containing all roles used in this project. The 'bios_idrac_settings' and 'idrac_settings' roles work to seperate functionality limited to only iDRAC to improve the readability of playbooks which will apply BIOS and BMC configurations on an inventory of multiple vendors. The 'common' role groups together common tasks such as 'include_host_vars.yml' which includes host-specific variables during playbook run-time.

3.4.7 tests

The tests directory is used when running automated tests in the CI-tool travisCI. The 'test-requirements.txt' file contains tools needed to be downloaded in the CI-system, and the inventory file defines a 'localhost' host which the playbooks are ran against in the testing-environment.

3.4.8 .travis.yml

This is the configuration file for the travisCI tool. It defines what type of system the tool shall build to test the playbooks.

3.4.9 .ansible-lint

This is the configuration file for the 'ansible-lint' tool which checks the playbooks for code quality and bugs. In this project some errors are ignored because they are intended functionality of the playbooks.

```
redfish-ansible
|-- ansible.cfg
|-- host_vars                              |--tests
|   |-- idrac1.yml                         |  |--inventory
|   |-- idrac2.yml                         |  |--test-requirements.txt
|   |-- idrac3.yml                         |--.travis.yml
|   |-- idrac4.yml                         |--.ansible-lint
|-- inventory
|   |-- static_inventory.yml
|-- LICENSE
|-- playbooks
|   |-- bios_settings.yml
|   |-- bmc_settings.yml
|   |-- export_server_config_profile.yml
|   |-- import_scp_preview.yml
|   |-- import_server_config_profile.yml
|   |-- server_health_check.yml
|   |-- server_setup.yml
|   |-- test_playbook.yml
|-- plugins
|   |-- modules
|       |-- get_job_details_redfish.py
|       |-- import_idrac_scp_preview.py
|-- README.md
|-- roles
|   |-- bios_idrac_settings
|   |   |-- tasks
|   |       |-- main.yml
|   |-- common
|   |   |-- tasks
|   |       |-- create_output_file.yml
|   |       |-- get_redfish_info.yml
|   |       |-- include_host_vars.yml
|   |-- idrac_settings
|   |   |-- tasks
|   |       |-- main.yml
|   | -- restart_idrac
```

Listing 12: Tree graph of the Ansible project directory structure

3.5 File storage

Files are an integral part of several IPMI and Redfish commands. Unless a BMC is connected to the Internet, a file is either accessed locally or from a network share. For instance, one of the core features of a BMC is to install an OS remotely which requires an installation file which can be stored in the network share. Server configuration profiles can also be stored here. iDRACs are compatible with both CIFS [67] and NFS [68] type file shares. In this project, the share type used is NFS because this is Linux-only environment. This has been located on the master node.

3.5.1 Git

Git is an open source tool which streamlines version control. Features such as branches make it possible for multiple users to work on the same project without affecting others, repositories can be publicly shared and versions can easily be rolled back if something breaks.

All files relevant to the PoC are kept within a public Git repository created by the project group [69]. Included amongst these files are playbooks, scripts, modules, etc.

Chapter 4

Implementation

This chapter is separated into several parts, which shows how Redfish can be used to solve the tasks presented in section 1.5 - "Scope" in chapter 1. The first part of the demonstration sticks to simple demonstrations in how the Redfish API can be used from the command line or together with scripting languages such as Python. The second part shows how Ansible Redfish modules and playbooks can be used to execute these tasks on a larger scale. The PoC consists of several created playbooks, plays and tasks in addition to two custom modules.

These demonstrations have been developed for and tested on Dells BMC implementation the iDRACs in accordance with the Dell Poweredge based lab environment provided by the Idun HPC group. However, only a few iDRAC specific implementations have been used, an example being Server Configuration Profiles (SCP), which uses a Dell-specific, OEM Redfish resource extension to import, export, or preview configuration settings as an XML file or JSON file. Though, with some modifications and research into other vendor implementations, the same principles as seen in this chapter can be applied to expand the functionality and practicality of this PoC. Otherwise, basic configuration or functionality shown should, unless otherwise noted, work on any BMC that supports and has implemented the Redfish specification version 1.6.0 with OpenAPI 3.0 support, which is the basis for our PoC.

4.1 High-level overview

The figure 4.1 is a high-level overview on how the different parts and technologies in the project environment interact when using Ansible together with Redfish. When Ansible is ran from the master node using the command line tool ’`ansible-playbook`’, two parameters are included. The first being the playbook, and the second being the inventory file. It then fetches the modules required - either from a previously downloaded Ansible community module, or using modules locally developed and added to the ’plugins/modules’-directory. In this PoC the host-groups are hard coded into the playbooks for testing purposes, and during playbook execution it looks up the group in the inventory file, fetches the correct host details, and then retrieves the host credentials from the relevant host-variable file. Then using the parameters included in the modules it crafts Redfish URIs and forms the destination of the HTTP(s) requests to the iDRAC hosts specified in the inventory file. These requests are received by the RESTful API which either performs actions on the BIOS and/or iDRAC configurations or collects information from it, and the result of the play is returned to the master node in the form of a JSON response.

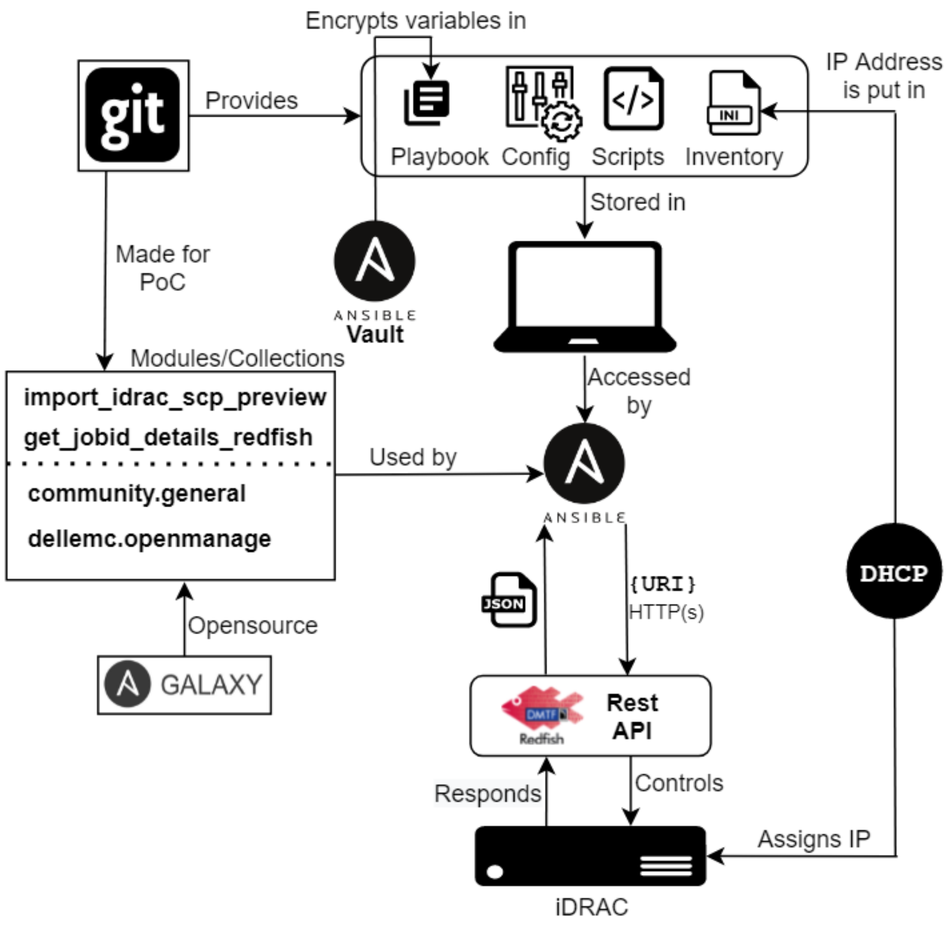


Figure 4.1: High-level overview of PoC

4.2 Proof of Concept specification

The following tasks serves as the specification for the PoC, and the remainder of this chapter shows the projects solution to them. The last task, changing the settings of multiple servers simultaneously using Redfish and Ansible, is demonstrated in the later chapter 6 - "Deployment".

- Read and save Integrated Dell Remote Access Controller (iDRAC) settings from one server to another using Redfish
- Write settings to iDRAC and BIOS with Redfish
- Collect support data with Redfish e.g. a health check
- Deploy a default iDRAC based on previously gathered settings using Redfish
- Simulate a deployment of iDRAC settings from "out of the box" server
- Investigate and if possible show how Redfish and Ansible can be used together to automate the iDRAC setup
- Change the settings of multiple servers simultaneously using Redfish and Ansible
- Demonstrate the “Read, Write and Deploy” settings with Ansible and Redfish

In addition, the Ansible modules ’Import Server Configuration Profile Preview’ (iDRAC specific) and ’Get Job Details’ has been developed to complement the workflow of certain Ansible playbooks which will be discussed later in this chapter. These are in an early stage of development, but work as an example of how to develop modules when there are no public modules for a use-case.

4.3 Lab environment

This section details the lab environment in which the demonstrations of the PoC was ran, and is included for the purpose of being able to reproduce the results presented below.

4.3.1 iDRAC

All iDRAC versions from v2.40.40.40 support the Redfish standard, but when upgrading from a version before 2.40.40.40, manually enabling the service is necessary. As seen in figure 4.2, all the models in the lab environment except node 2 supports Redfish. Originally ’node 2’ was updated to firmware version 2.65.65.65 but the group requested a reset so that one could observe its default state. Dell PowerEdge C6420 is one of the newer models used in the production environment by the employer today, and comes with iDRAC9 v4.32.10.00 out of the box which has the Redfish specification version 1.6.0 implemented. ’Node 1’ works as a reference to older or different versions of iDRAC. The latest version at the time of writing is v4.40.00.00. For full summary of the differences between the versions an excel-sheet was created [70].

redfish@bachelor.hpc.ntnu.no is the hostname of the master node, which is used as a gateway to connect to nodes 1-4, where all nodes share the same internal network. Nodes 3 and 4 has Centos8 installed to work as a backdoor in case of configuration errors which could result in loss of connectivity to the iDRAC.

Hostname	**OS**
redfish@bachelor.hpc.ntnu.no	Centos7

Switch Model
Dell N3048

Node	**iDRAC Version**	**Model**	**Firmware Version**	**OS**
1	iDRAC7	R720	2.65.65.65	
2	iDRAC7	R720	2.10.10.10	
3	iDRAC9	C6420	4.32.10.00	Centos8
4	iDRAC9	C6420	4.32.10.00	Centos8

Figure 4.2: List of hardware used in the lab

4.3.2 DHCP

In the beginning, the lab environment consisted of two Dell R720 (iDRAC7) servers which were assigned a static IP configuration by the employer. Adding new servers or resetting their configurations would require additional manual iDRAC configuration to grant the group remote access through the bachelor-server. However, since DHCP is enabled by default, a DHCP server was installed so that an IP configuration is automatically assigned to the iDRAC, removing the requirement of locally configuring the network settings on the iDRAC on resets or new installments.

The configuration file as well as the commands used for installation can be found in appendix B.5. It was decided to use the IP-range 192.168.0.0/24, reserving the first 10 addresses in case some static IP addresses were needed. To avoid manually updating the static inventory files when working with Ansible, the lease time is set to be equivalent to the remaining duration of the project. As an alternative, IP-addresses can be reserved in the DHCP-configuration by binding them to a specific MAC address so that each server will receive the same IP-address even if the lease expires. However, the temporary solution of just extending the lease-time was found to be a sufficient for this lab environment because of the few amount of servers and the short-term nature of this project. The figure 4.3 shows an overview of the IP ranges and the address types.

IP-Range	**Address Type**
192.168.0.1-10	Static
192.168.0.11-254	DHCP Pool

Figure 4.3: IP addresses

4.3.3 Prerequisites

This subsection includes a few prerequisites which are needed for attempting to test any of the Redfish or Ansible demonstrations below, including any of the project made plays, tasks or modules. Lastly it describes where and how to get the PoC.

Host operating system

The master node uses the operating system Centos7. This was pre-installed by the employer and is the go-to operating system for the HPC group.

Python

Ansible and its modules should be compatible with both Python versions 2 and 3, however, this implementation is only tested on a host with Python version 3.6.8.

OMSDK

Some Ansible modules relating to iDRAC implementations of Redfish require the DellEMC OpenManage Python Software Development Kit (OMSDK). It is a Python library for developers working with PowerEdge servers. For example, the playbooks which demonstrates the import and export of iDRAC Server Configuration Profiles, uses Ansible Modules which requires this library. It is an open-source project, and installation steps can be found in its official repository [71].

Ansible

The PoC was tested with the latest Ansible version at the time, version 2.10.7. Because of backwards-compatibility, installing the latest released version on a supported operating system should have no effect on the functionality of this PoC.

Ansible collections

The community.general v2.5.1 collection includes many modules and plugins which is supported by the Ansible community. The latest documentation can be found in Ansible official documentation [40], and any changes in newer versions should provide backwards-compatibility so no changes have to be made to the module or collection references in the PoC. It is installed using the Ansible Galaxy CLI with the command `ansible-galaxy collection community.general`. This is an open-source project on GitHub [72], and in some cases the files containing the code for the Ansible-modules provide better documentation than Ansibles official documentation.

File share

A network file share is required for only one locally developed module in this PoC: 'Import Configuration Profile Preview'. Currently only Network File System (NFS) is supported by this module. The installation steps and configuration file used for this PoC can be found in the appendix B.2.

Git and cloning the PoC repository

The repository [69] contains all code, templates, playbooks, roles, and tasks referenced in this thesis. Running the playbooks which are located in the PoC repository requires cloning or forking the repository to a controller machine with root access and direct access to the OOB controllers. The repository is cloned using the git tool [73] and the command `git clone`. Keep in mind that there would be a need to configure the lab environment specific files such as the inventory file, and the host files. These files contains information which is specific to the lab environment, and without editing these files the PoC would fail to reproduce results.

4.4 Demonstrations

The following demonstrations are made on a node with iDRAC9 version 4.32.10.00. They mostly consist of simple curl commands to the Redfish RESTful API, but when appropriate a Python script is used instead.

4.4.1 Reading and saving iDRAC settings from one compute node with Redfish

iDRAC instances are found in the Managers collection: `/redfish/v1/Managers`. To get the specific resource ID of the iDRAC, a GET request is sent using the tool 'curl', which returns the body of the response in a JSON format. In this response the ID of the instance can be found in the 'Members' array. To pretty-print the response it is possible to pipe the command to the Python module json.tool, which makes the response more human readable. An example of this is shown in listing 13.
The response shows one instance in the `Members` array in the `Managers` collection with the ID `iDRAC.Embedded.1`. When developing tools for Redfish clients, one can never assume that this ID is the same across all vendors. For example, Lenovo and HPE implementations uses integers as the instance ID [74] [75]. An additional GET request to `/redfish/v1/iDRAC.Embedded.1` will return all properties/attributes of the iDRAC which is exposed by the implemented Redfish specification, as shown in listing 14. This is also documented in Dells API guide [76].

The response is a JSON object containing information about the instance itself (iDrac.Embedded.1), as well as the modifiable attributes (settings). The attributes are stored as a list in the "Attributes"-key of the response. The example in listing

```
[root@bachelor redfish-ansible]# curl -sX GET  -u root:redfish -k
↪   https://192.168.0.123/redfish/v1/Managers | python -m json.tool
{
    "@odata.context":
    ↪   "/redfish/v1/$metadata#ManagerCollection.ManagerCollection",
    "@odata.id": "/redfish/v1/Managers",
    "@odata.type": "#ManagerCollection.ManagerCollection",
    "Description": "BMC",
    "Members": [
        {
            "@odata.id": "/redfish/v1/Managers/iDRAC.Embedded.1"
        }
    ],
    "Members@odata.count": 1,
    "Name": "Manager"
}
```

Listing 13: GET request to find the resource id of a Manager instance

14 shows an excerpt of the response as the attributes-list is very long, though a complete list of all attributes is documented by Dell in their iDRAC 9 Attribute Registry [77]

Because of the length of the list it can be an advantage to save the response to a file. It is possible to redirect the standard output to a file using the `>` operator, as shown in the listing 15. This will store the output of the command to a file in the path specified. If the file does not exist it will be created.

```
[root@bachelor redfish-ansible]# curl -sX GET  -u root:redfish -k
↪   https://192.168.0.123/redfish/v1/Managers/iDRAC.Embedded⌋
↪   .1/Attributes | python -m
↪   json.tool
{
    "@Redfish.Settings": {
        "@odata.context": "/redfish/v1/$metadata#Settings.Settings",
        "@odata.type": "#Settings.v1_2_2.Settings",
        "SettingsObject": {
            "@odata.id": "/redfish/v1/Managers/iDRAC.Embedded⌋
            ↪   .1/Attributes/Settings"
        },
        "SupportedApplyTimes": [
            "Immediate",
            "AtMaintenanceWindowStart"
        ]
    },
    "@odata.context":
    ↪   "/redfish/v1/$metadata#DellAttributes.DellAttributes",
    "@odata.id": "/redfish/v1/Managers/iDRAC.Embedded.1/Attributes",
    "@odata.type": "#DellAttributes.v1_0_0.DellAttributes",
    "AttributeRegistry": "ManagerAttributeRegistry.v1_0_0",
    "Attributes": {
        "SupportAssist.1.DefaultProtocolPort": 0,
        "SupportAssist.1.HostOSProxyPort": 1,
        "CurrentNIC.1.DedicatedNICScanTime": 5,
        "CurrentNIC.1.MTU": 1500,
        "CurrentNIC.1.NumberOfLOM": 3,
        "CurrentNIC.1.SharedNICScanTime": 30,
        "CurrentNIC.1.VLanID": 1,
        "CurrentNIC.1.VLanPriority": 0,
        "CurrentIPv6.1.IPV6NumOfExtAddress": 0,
        "CurrentIPv6.1.PrefixLength": 64,
        "SysInfo.1.LocalConsoleLockOut": 1,
        "SysInfo.1.POSTCode": 127,
        "SysInfo.1.SystemRev": 0,
        "GpGPUTable.1.TierEncoding": 1,
        "GpGPUTable.2.TierEncoding": 1,
        "GpGPUTable.3.TierEncoding": 1,
        "GpGPUTable.4.TierEncoding": 1,
```

Listing 14: GET request to the manager instance and an excerpt of the response

```
[root@bachelor redfish-ansible]# curl -sX GET  -u root:redfish -k
 ↪   https://192.168.0.123/redfish/v1/Managers/iDRAC.Embedded
 ↪   .1/Attributes | python -m json.tool >
 ↪   fileincurrentdirectory.json
```

Listing 15: Redirecting standard output to a file

4.4.2 Write BIOS settings with Redfish

BIOS settings are found in the 'Attributes'-key of the system instance in the `Systems` collection resource. In listing 16, one can see an example curl command for retrieving these settings.

```
[root@bachelor redfish-ansible]# curl -X GET  -u root:redfish -k
 ↪   https://192.168.0.123/redfish/v1/Systems/System.Embedded.1
```

Listing 16: GET request to the system instance

Writing settings requires a PATCH request to the `bios/settings` resource with the attributes that will be changed as the payload in a JSON format. A POST request is not supported as that would overwrite all settings that were not included in the payload, while a PATCH request will only change the specified attributes, leaving the others as they were.

For example, to disable hyper-threading, the relevant attribute name is called 'LogicalProc'. Additional information about the resource and allowed values can be found in the `bios/biosregistry` resource. An excerpt of the response can be seen in appendix B.13, showing the details of the 'LogicalProc' attribute.

From this information one can gather that the current value is 'null', and accepted values are enabled and disabled. A restart would be required after a BIOS change.

The listing 17 shows a PATCH request to disable hyper-threading by setting the the value of the 'LogicalProc' attribute to disabled.

The response body message responds that the operation was completed. This can be confirmed with a GET request to the `bios/settings` resource of the instance as shown in appendix B.14. The attribute(s) to be changed is in the "Attributes"-key of the response, and signifies the attributes which will be changed on next system reboot.

```
[root@bachelor redfish-ansible]# curl -sX PATCH -u root:redfish -k
↪  https://192.168.0.123/redfish/v1/Systems/System.Embedded⌋
↪  .1/Bios/Settings\
 --data '{"Attributes": {"LogicalProc": "Disabled" }}' \
-H "content-type: application/json"

{"@Message.ExtendedInfo":[
    {"Message":"Successfully Completed Request",
    "MessageArgs":[],
    "MessageArgs@odata.count":0,
    "MessageId":"Base.1.5.Success",
    "RelatedProperties":[],
    "RelatedProperties@odata.count":0,
    "Resolution":"None",
    "Severity":"OK"
    },
    {"Message":"The operation successfully completed.",
    "MessageArgs":[],
    "MessageArgs@odata.count":0,
    "MessageId":"IDRAC.2.2.SYS413",
    "RelatedProperties":[],
    "RelatedProperties@odata.count":0,
    "Resolution":"No response action is required.",
    "Severity":"Informational"}]
}
```

Listing 17: PATCH request to disable hyperthreading, and the response-body

If any errors with the pending configuration is discovered, they can be cleared by making a POST request on the `DellManager.ClearPending` action, this can be seen in listing 18.

```
curl -X POST -u root:redfish -k
↪  https://192.168.0.123/redfish/v1/Systems/System.Embedded⌋
↪  .1/Bios/Settings/Actions/Oem/DellManager.ClearPending
```

Listing 18: Clear pending configurations

A restart of the computer system can be evoked with a POST command on the Actions/ComputerSystem.Reset resource. The 'ForceRestart' option will shut down the computer system immediately and restart the system. For a full list of options and their description, reference the Dell Redfish API guide [76].

```
curl -X POST -u root:redfish -k
↪   https://192.168.0.123/redfish/v1/Systems/System.Embedded
↪   .1/Bios/Settings/Actions/ComputerSystem.Reset
--data '{"ResetType": "ForceRestart" }' \
-H "content-type: application/json"
```

Listing 19: Force system restart using Redfish

4.4.3 Write IDRAC settings with Redfish

iDRAC settings are found in the 'Attributes' key of the manager instance. In this case, the manager instance is 'iDRAC.Embedded.1', making the full redfish URI `https://{IP}/redfish/v1/Managers/iDRAC.Embedded.1/Attributes` . A PATCH request can be made on the URI, containing attributes to be changed. Listing 20 shows how Redfish is used to change the password of the default iDRAC user which is defined in the 'Users.2.Password'-attribute as well as the sample output which confirms that the attribute has been changed.

4.4.4 Collect support data with Redfish

As shown in the previous examples in subsection 2.4.2, it is possible to gather summarized health information about all instances of a System, Chassis and Managers, but gathering details about all the components of a subsystems is better accomplished using a script because it requires iterating over a number of links to an unknown number of components.

As mentioned before, a summarized health information, a 'curl' command can be used on this template URL: `https://{ip}/redfish/v1/{collection}/{instance id}` . Retrieving the "Status" key of the response will give summarized health information about the instance of the collection. This is shown in listing 21.

The information in listing 21 is usually not enough for most use-cases. To be able to troubleshoot and potentially fix a problem requires more information, an administrator for example would want to known which components are responsible for the 'critical' status. The pseudo code in appendix listing B.6 is derived from manually navigating the Redfish API and is written to guide the development of vendor and instance agnostic Redfish scripts (requires minimum Redfish v1.6.0 OpenAPI 3.0 support on target host). It may seem a bit complex, but all it does is show the necessary data traversal when writing vendor and instance-agnostic scripts to gather health information about the system, subsystems and their components.

```
[root@bachelor ~]# curl -X PATCH -u root:redfish -k
↪   https://192.168.0.123/redfish/v1/Managers/iDRAC.Embedded
↪   .1/Attributes -H "content-type: application/json" --data '{"Attributes":
↪   {"Users.2.Password": "redfish_new"}}' | python -m json.tool

{
    "@Message.ExtendedInfo": [
        {
            "Message": "Successfully Completed Request",
            "MessageArgs": [],
            "MessageArgs@odata.count": 0,
            "MessageId": "Base.1.5.Success",
            "RelatedProperties": [],
            "RelatedProperties@odata.count": 0,
            "Resolution": "None",
            "Severity": "OK"
        },
        {
            "Message": "The operation successfully completed.",
            "MessageArgs": [],
            "MessageArgs@odata.count": 0,
            "MessageId": "IDRAC.2.2.SYS413",
            "RelatedProperties": [],
            "RelatedProperties@odata.count": 0,
            "Resolution": "No response action is required.",
            "Severity": "Informational"
        }
    ]
}
```

Listing 20: Changing iDRAC settings using Redfish

The appendix listing B.3 shows a script written in Python. The Python language was chosen because it is well documented, easy to learn if you have some previous programming experience, and comes with many libraries for different tasks. The code is is based on the pseudo-code in appendix B.6, and uses the 'requests' library for HTTP(s) requests and the 'json' library for parsing JSON data from Redfish API responses to gather health information about a host. Modifications could be made to print all categories. It could also be modified to only print components with a different health-status than "OK". A sample output from this Python script can be found in the appendix listing B.4.

```
# curl -X GET -u root:redfish -k
↪   https://192.168.0.123/redfish/v1/Systems/System.Embedded.1/| python
↪   -m json.tool
# Excerpt:
"Status": {
        "Health": "Critical",
        "HealthRollup": "Critical",
        "State": "Enabled"
    },
```

Listing 21: Summarized health information of a 'Systems' instance

4.5 Demonstrating Redfish and Ansible

Redfish tasks as demonstrated in the previous listings in this chapter can be scaled to work on multiple hosts and automated with the tool Ansible. Ansible primarily uses Python modules which are very similar to the example in appendix B.3, except that they always return a JSON-structure and uses Python-libraries for Ansible to accept parameters in addition to following a set of coding principles for quality and backwards compatibility. The Ansible modules for Redfish which was used in this project are all open-source and available in the `community.general` collection [72]. The following modules was used: `redfish_info` [78], `redfish_config` [79], `redfish_command` [80], and `idrac_redfish_command` [47]. These modules are called from Ansible tasks.

In addition, the `redfish_utils` [81] module was used when developing local modules to cover functionality which was not yet developed or pushed to the above mentioned modules. The local modules developed were for previewing import configuration profiles (iDRAC specific) and getting job details (all vendors).

4.5.1 Server Health Check with Ansible

In a server environment one could potentially use several Python scripts running using Crontab or in other ways automated for performing continuous health checks on the environment. Though this is also possible to accomplish using Ansible in a more structured and clear way. The file `server_health_check.yml` is a playbook which contains tasks to accomplish the same goal as the Python script in appendix listing B.3. It gathers server health information on hosts specified by a host-group at the top of the file. This group is defined in the static inventory file. During this test it was a single iDRAC, the same which the previous Python script was ran on. Though this host list could be expanded, and be used for active health monitoring of larger infrastructure when integrated with other systems or scripts.

The Ansible Playbook for checking the health of one to several servers is shown in appendix listing B.7, and it consists of 6 tasks which together gathers health information about the server. The first task in this playbook is a task which includes host-specific variables, which are required if it is to access the Redfish API on the managed servers. This is because the password-variable is encrypted in the host-variable files as shown in listing 22. The encrypted string would then be decrypted at run time, and normally the Ansible-playbook command would prompt the user to enter the Ansible-vault password which would be used to decrypt the host-password. This is not the case for this PoC as in the ansible.cfg file there is a pointer to a text-file on the Master-node. This file contains the vault password in plaintext, which realistically is not considered good practice. It was found that for this PoC this was sufficient, and there are other technologies such as Ansible Tower [82] which offers better solutions to this.

After importing the variables, a common task is included from the 'playbooks/tasks'-directory which creates an empty file to store the output from the playbook in. Following that are three tasks which gather health information from the system by using the Ansible Redfish module `Redfish_info` from the community generals collection. As per the requirements of this module, parameters such as category, command, URL, and credentials are included, and finally the result is stored in a variable which is then printed to the previously created file. A sample of the output of this playbook can be found in the appendix B.1.

```
ansible\_user: "root"
ansible_password: !vault |
    $ANSIBLE_VAULT;1.1;AES256
    3032646439353637393161356639393238623163636566336437633463623537626436633161666
    6238373330336363306165313233636261623630363763310a3339653039393166636239363966396639
    63363438613964626664356562653837653365623666363132333034613432666134363238633263
    3865316336616131380a666361643237646339316332373338316138346464333066353665656236
    6164
```

Listing 22: Ansible-vault variables for iDRAC3

4.5.2 Export server configuration

Another playbook which was created is shown in listing 23, this details a play which extracts the current configurations on the server and exports it to a JSON file and saves this file in the 'tmp'-folder. This playbook was created on the request of the employer to achieve the task 'Read and save iDRAC settings from one server to another', and was made as an example in how OEM functionality of the Redfish implementation can be used just as easily as standard Redfish functionality. It is a further development of the simple demonstration in subsection 4.4.1, and it uses the iDRAC Ansible module 'iDRAC server config profile'. This module is using resources which are specific for the iDRAC and would not function on any other BMC implementation.

```
---
# Export server configuration profile, stores in /tmp/

- hosts: idrac3
  connection: local
  gather_facts: no
  name: Export server configuration profile
  vars:
    test_mode: 0
    ansible_python_interpreter: /usr/bin/python3.6

  tasks:

  - name: Include task for including host specific variables.
    include_tasks: tasks/include_host_vars.yml
    when: not test_mode

  - name: Export Server Configuration Profile to local path
    idrac_server_config_profile:  # since general collection v2.3.0, this
     ↪  redirects to dellemc-openmanage-ansible-modules
      idrac_ip: "{{ ansible_host }}"
      idrac_user: "{{ ansible_user }}"
      idrac_password: "{{ ansible_password }}"
      share_name: "/tmp/"
      export_use: "Default"
      export_format: "JSON"
      job_wait: "False"
```

Listing 23: Ansible playbook YAML file for exporting server configurations

4.5.3 Import server configuration

The listing 24 shows a playbook made for importing the JSON file exported using the Ansible playbook in listing 23. In this case the configuration file is hardcoded to '192.168.0.123_20210516_180020_scp.json', and would import the configurations which are present within this file. It uses the same module as listing 23, and is an iDRAC specific playbook. This playbook fulfills the task of deploying a default iDRAC based on previously gathered settings, and based on the hosts specified it can push this configuration to several servers simultaneously.

4.5.4 Discovery

The Redfish specification supports the Simple Service Discovery Protocol (SSDP), which allows for location of devices which conforms to the Redfish specification.

```yaml
# Import server configuration profile exported by
↪  "export_server_config_profile.yml".

- hosts: idrac3
  connection: local
  gather_facts: no
  name: Import server configuration profile
  vars:
    test_mode: 0
    ansible_python_interpreter: /usr/bin/python3.6

  tasks:
  - name: Include task for including host specific variables.
    include_tasks: tasks/include_host_vars.yml
    when: not test_mode

  - name: Import Server Configuration Profile
    idrac_server_config_profile:
      idrac_ip: "{{ ansible_host }}"
      idrac_user: "{{ ansible_user }}"
      idrac_password: "{{ ansible_password }}"
      command: "import"
      share_name: "/tmp/"
      scp_file: "192.168.0.123_20210516_180020_scp.json"
      scp_components: "ALL"
      job_wait: "True"
```

Listing 24: Ansible playbook YAML file for importing server configurations

Redfish is also investigating other (e.g DHCP-based) approaches for service entry point discovery [83]. However, SSDP is an optional implementation which means it does not need to be implemented by the vendors. This is the case for the iDRAC. Dell's iDRAC uses their own implementation of automatic discovery[84], which would require setting up a remote management console which hosts a provisioning server. This was found to be out of scope, as it is not part of Redfish, and it would only be functional for the iDRAC.

For this PoC, it is assumed that all BMCs have DHCP enabled, so that any new connected servers are automatically assigned IP addresses. These BMCs are then able to be identified by querying the DHCP server for new leases, and these servers can then be identified using HTTPS GET requests on the Redfish API, and then added to the static inventory file in the Ansible directory. This could be accomplished utilizing either infrastructure management tools such as Ansible which either uses tasks with custom modules for managing files, or through calling upon custom scripts. However, IP address management, which involves the planning and management of the assignment and use of IP addresses are not within the scope of this project.

Assuming a list of IP addresses is already in the inventory file, but the vendor or other product information is unknown, a part of auto-discovery would require their identification before pushing them. To demonstrate this functionality, an Ansible playbook has been written which makes a HTTP(s) GET request towards the Redfish API which can be seen in listing 25. This request queries the service root at the `redfish/v1/` URL and specifically checks the 'product' attribute of the return content. By adding a condition for `product == 'iDRAC'`, the role which applies the iDRAC configuration settings can be called on the newly discovered BMC.

```
---
- name: Get server information
  ansible.builtin.uri:
    url: "https://{{ ansible_host }}/redfish/v1"
    method: GET
    headers:
      Accept: "application/json"
      Content-Type: "application/json"
    validate_certs: no
```

Listing 25: Ansible task for identifying the type of BMC

4.5.5 Importing host variables

The only needed host-variables for these demonstrations are user credentials to the OOB controllers. Accessing the Redfish API of the OOB controllers requires user credentials which can be stored and organized in Ansible in a number of ways. A possible solution is to include them as host-variables in the inventory file, however this removes the option of encrypting the password with Ansible Vault. Variables can also be assigned to the groups specified in the inventory file, so for example, if there is a group named "Dell" in the inventory file, variables can be assigned in the 'group_vars/dell.yml' file. Single variables like the password to the BMC can then be encrypted with Ansible vault, and the variables can be included in a playbook using the Ansible built-in `"include_vars"` functionality. A downside of organizing variables in this way, is that all hosts in the group has to have the same password, which is not consired best practise security-wise. To assign variables for each host, a file for each inventory entry can be created in the 'host_vars' directory. If the naming-scheme for these files is consistent (same file name as the host name defined in the inventory file), the relevant host variables can be called during playbook runtime by including the variable file using the global Ansible 'inventory_hostname' variable. This allows the administrator to have unique passwords per machine, but increases the demand for maintenance, especially since password-changes would require encrypting the new password using Ansible vault and editing the relevant host-variable file.

In this demonstration, host variables are organized on a per-host basis in the 'host_vars' directory. This works great in a PoC and other small environments, but might not be a scalable solution in a production environment. Theoretically, this solution is the most secure of the mentioned host-variables methods because it allows unique, encrypted passwords on each node. But as always, security requires a cost-benefit approach where the administrative consequences of following the security best practise of unique passwords may be too large compared to the convenience of having one, strong password as a group-variable. Figure 22 shows an example host-variable file in the 'host_vars' directory. It contains the variables 'ansible_user' and 'ansible_password' where the 'ansible_password' variable has been encrypted with Ansible vault.

Because the host-variable files have a fixed location and filename, they can be included with a relative pathname and the global 'inventory_hostname' variable, as seen in figure26. This task will need to be called from multiple playbooks, so it is placed in the "common" role so that it can included in other playbooks with little effort. Figure 27 shows an example of how the "include host variables" is included in a playbook by calling the "common" role and specifying the task which should be included.

```yaml
---
- name: Include host specific Ansible Vault encrypted variables
  include_vars:
    file: "{{ playbook_dir }}/../host_vars/{{ inventory_hostname }}.yml"
```

Listing 26: Example task which includes host-specific variables

```yaml
tasks:
   - name: Include host specific variables
     include_role:
       name: common
       tasks_from: include_host_vars
     when: not test_mode
```

Listing 27: Example tasks which includes the common task 'include_host_vars' from the 'common'-role

4.5.6 Playbook and roles

Because BIOS attributes vary widely across server types and BIOS revisions, the tasks which set the configurations are split up and organized as Ansible Roles. When applying BIOS or BMC configurations on different vendors, a new role should be created, and the pre-defined configurations can be applied to the different hosts simultaneously by calling them on the correct `hosts`-group in the main main.yml playbook. Listing 28 shows how to call multiple roles on different groups of hosts.

```
---
- hosts: idrac
  roles:
    - idrac_bios_settings
    - idrac_settings

- hosts: XCCLenovo
  roles:
    - xcc_bios_settings
    - xcc_settings
```

Listing 28: Example playbook. Includes roles for configuration of different vendor BMC implementations.

Both the role that configures BIOS settings, as well as the role that configures iDRAC settings are implementations of Ansible roles, containing the tasks-directory with a 'main.yml' file which contains a set of tasks specific for the intended role functions. Other Ansible role-artifacts like default variables or handlers are not needed, as the host credential variables are imported with a separate task, and handler-functionality (rebooting system after a change) is handled by an individual task. For all intents and purposes, the two roles could be merged into one which is just called "Dell configuration". There is no wrong way to organize roles as long as the the minimum requirements for a role is met. The advantage of splitting them up in in this PoC is that they can be called and tested separately, as well as distinguishing the iDRAC configuration from the BIOS configuration at the playbook-level, which helps with readability for users of the playbook.

Role: BIOS_idrac_settings

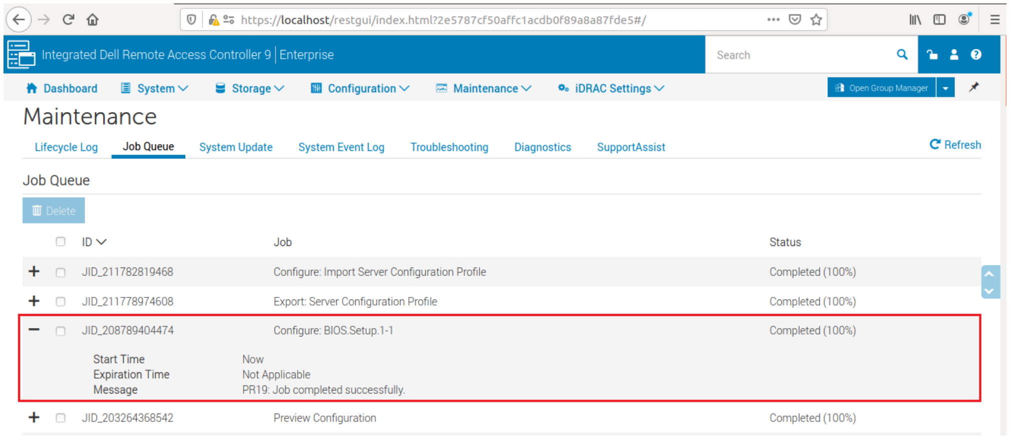
The purpose of this role is to group together tasks which are related to BIOS configuration with iDRAC. These tasks involves the host-specific variables (credentials), calling the ' `redfish_config` ' module to apply the BIOS attributes listed in figure 6.1, scheduling a BIOS configuration job for next system reboot(" `idrac_redfish_config` " module), and if the attributes were applied prompt a system reboot (" `redfish_command` ").

The scheduling of a BIOS configuration job is a Dell specific requirement to apply the BIOS configurations, otherwise the configuration is never applied. The ' `idrac_redfish_config` ' module contains the logic required to schedule the job so that the configuration is applied on next system reboot. At the time of writing, the module does not return a job-id which can be extracted by the role at playtime. It is therefore not possible to track the progress of the job during playbook

```yaml
#redfish-ansible/roles/bios_idrac_settings/tasks/main.yml
---
- name: Include host specific Ansible Vault encrypted variables
  include_tasks: ../../../playbooks/tasks/include_host_vars.yml
  when: not test_mode

- name: Set BIOS settings
  community.general.redfish_config:
    category: Systems
    command: SetBiosAttributes
    resource_id: "System.Embedded.1"
    bios_attributes:
      SysProfile: "Custom"
      EnergyPerformanceBias: "MaxPower"
    baseuri: "{{ ansible_host }}"
    username: "{{ ansible_user }}"
    password: "{{ ansible_password }}"
  register: settings

- name: Create BIOS configuration job (schedule BIOS setting update)
  community.general.idrac_redfish_command:
    category: Systems
    command: CreateBiosConfigJob
    resource_id: System.Embedded.1
    baseuri: "{{ ansible_host }}"
    username: "{{ ansible_user }}"
    password: "{{ ansible_password }}"
  when: settings.changed

- name: Restart system
  community.general.redfish_command:
    category: Systems
    command: PowerForceRestart
    resource_id: "System.Embedded.1"
    baseuri: "{{ ansible_host }}"
    username: "{{ ansible_user }}"
    password: "{{ ansible_password }}"
  when: settings.changed
```

Listing 29: Ansible tasks for BIOS configuration

runtime, however, a comment in the code [47] implies a patch is coming soon. If this is discovered to be s necessity, and the patch is still absent, a fork of the module can be created to modify the returned JSON object to include the job-id which will be located in the response-header of the POST request which creates the job. The module can then be added to the 'plugins/modules' directory of the PoC collection and called with the new module name instead of "community.general.idrac_redfish_command". Tasks which extract the job-id and monitors the progress can then be created. In this PoC, the creation of the job is confirmed if the task returns as 'changed' in the Ansible output. Further details can be observed by connecting to the iDRAC GUI and inspecting the job queue, as seen in figure 4.4.



ID	Job	Status
JID_211782819468	Configure: Import Server Configuration Profile	Completed (100%)
JID_211778974608	Export: Server Configuration Profile	Completed (100%)
JID_208789404474	Configure: BIOS.Setup.1-1	Completed (100%)
Start Time	Now	
Expiration Time	Not Applicable	
Message	PR19: Job completed successfully.	
JID_203264368542	Preview Configuration	Completed (100%)

Figure 4.4: Completion of a BIOS job as seen in the iDRAC GUI

```yaml
---
- hosts: idrac
  roles:
    - idrac_bios_settings
```

Listing 30: iDRAC Attributes

This role is included as a play in the 'bios_idrac_settings.yml' playbook as seen in listing 30. The playbook is then executed in an ad-hoc manner with the ' `ansible-playbook` ' command as seen in figure 31.
Figure 31 also shows the output of a successful run, returning either "OK" or "Changed" to indicate its success. The condition "when: settings.changed" on the last two tasks in listing 29 ensure they only run if the configuration has changed. If the condition fails, the tasks will be skipped, because there is no need to create a BIOS configuration job or restarting the system if no changes were made to the system. Listing 29 shows how this described functionality is implemented as an Ansible role.

Role: iDRAC configuration

The role that applies iDRAC configuration makes use of the community 'idrac_redfish_config' module because the 'redfish_config' does not support the change of manager (BMC) attributes. Because the 'idrac_redfish_config' module existed, there was no need to develop a new module, but because it was mainly developed for iDRAC there is no guarantee from the developers that it will work on

```
[root@bachelor redfish-ansible]# ansible-playbook -i inventory/
↪  playbooks/bios_settings.yml
PLAY [Apply BIOS settings]
↪  **********************************************************
TASK [bios_idrac_settings : Include host specific Ansible Vault encrypted
↪  variables] ************************************included:
↪  /home/redfish/test/git/bachelor_repo/redfish-ansible/roles/bios_id⌋
↪  rac_settings/tasks/../../../playbooks/tasks/include_host_vars.yml for
↪  idrac3


TASK [bios_idrac_settings : Include host specific Ansible Vault encrypted
↪  variables]
********************************ok: [idrac3]


TASK [bios_idrac_settings : Set BIOS settings]
*******************************changed: [idrac3]

TASK [bios_idrac_settings : Create BIOS configuration job (schedule BIOS
↪  setting update)] *******************************changed: [idrac3]

TASK [bios_idrac_settings : Restart system]
*******************************changed: [idrac3]

PLAY RECAP**************************************
idrac3: ok=5    changed=3    unreachable=0    failed=0    skipped=0
↪  rescued=0    ignored=0
```

Listing 31: Output of running the bios_settings playbook

other vendors than Dell. However, on closer inspection of the code [85], because the module finds the the manager instance id automatically instead of hardcoding it into the code, the module command ’ `SetManagerAttributes` ’ should work on vendor implementations of Redfish versions later than 1.6.0. This is however not tested.

The reason behind creating a role for applying iDRAC configurations is the same as for BIOS configurations. Separating vendor specific functionality makes logical sense, and helps reduce playbook complexity and improve readability. The attributes that is seen applied in the playbook (ref. listing 32) can be changed or added based on business needs, and correct attribute-names can be found with the curl command: ’ `curl -X GET -u username:password -k https://{host_ip}/redfish/v1/Managers/{resource_id}/Attributes | jq '.Attributes'` ’. This return information about all iDRAC or other BMC implementation attributes and values.

```
- name: Set iDRAC settings
    community.general.idrac_redfish_config:
      category: Manager
      command: SetManagerAttributes
      resource_id: iDRAC.Embedded.1
      manager_attributes:
        IPMISOL.1.Enable: "Enabled"
        Users.2.SolEnable: "Enabled"
        SerialRedirection.1.Enable: "Enabled"
        NIC.1.VLanEnable: "Enabled"
        NIC.1.VLanID: "1"
        NIC.1.Selection: "LOM1"
        Users.2.UserName: "root"
        Users.2.Password: "redfish"
        Telnet.1.Enable: "Disabled"
        SNMP.1.AgentEnable: "Disabled"
        WebServer.1.HttpsRedirection: "Enabled"
        WebServer.1.TLSProtocol: "TLS 1.2 Only"
        IPBlocking.1.BlockEnable: "Enabled"
        IPBlocking.1.BlockEnable: "Enabled"
        IPBlocking.1.FailCount: "3"
        IPBlocking.1.FailWindow: "60"
        IPBlocking.1.PenaltyTime: "60"
        SSHCrypto.1.KexAlgorithms:
         ↪  "curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-
         ↪  nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-
         ↪  group-exchange-sha256,diffie-hellman-group16-sha512,diffie-
         ↪  hellman-group18-sha512,diffie-hellman-group14-sha256"
        SSHCrypto.1.MACs:
         ↪  "umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-
         ↪  sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-
         ↪  128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1"
        SSHCrypto.1.Ciphers:
         ↪  "chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-
         ↪  ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com"
        SSHCrypto.1#HostKeyAlgorithms: "ssh-rsa,rsa-sha2-512,rsa-sha2-
         ↪  256,ecdsa-sha2-nistp256,ssh-ed255"
      baseuri: "{{ ansible_host }}"
      username: "{{ ansible_user }}"
      password: "{{ ansible_password }}"
```

Listing 32: iDRAC Attributes

4.5.7 Modules

There was developed two modules for this project, these were both related to testing 'server configuration profile' files before importing them to a server. The modules also serve as demonstration to how one could go about creating custom modules to cover missing functionalities. A demonstration of how these modules are used at the playbook-level is covered in chapter 6 - "Deployment".

Server Configuration Profile Preview

One of the purposes of a server configuration profile is to define a server configuration template which can be applied to Dell servers. This process may fail if the configuration is pushed to an older firmware version than the template because some configuration attributes may not have been supported or implemented yet. SCP preview is an alternative to pushing SCP files onto a server which is not guaranteed to support all attributes. The 'preview' action will only check if the import would have been successful if it had been imported, but will not make any changes on the system. This makes it a safe option compared to blindly attempting an import of a configuration from one server to another of a different model or firmware version. This action requires that the configuration file is stored in a network share on the master node.

Pre-existing Python libraries for Redfish functionality and Ansible modules can be used to help the module development process. The Python library "rf_utils" contains a class with a number of common Redfish utility functions used to interact with the Redfish API, like making GET requests. The advantage of this architecture is that developers can import a library, create a class which inherits all the functionality, and use these functions to achieve the intended module results without having to "reinvent the wheel" with basic Redfish functionality. The "AnsibleModule" library helps abstract the interactions with the module and Ansible playbook. It abstracts the passing of module parameters, as well as the program exit which always has to return a JSON response.

In summary, the 'server configuration profile preview' module will import the aforementioned Python-libraries, define a set of accepted parameters like host credentials, network share information, and which components of the configuration (all, BIOS, iDRAC, etc) should be attempted to be imported. It will then build and do a POST request on the Redfish URI of the SCP preview action, and return a response on whether it was successful. In contrast to the pre-existing "Import SCP"-module, this module will also return the job-id of the created task. This job-id can be used to query the task-service of the Redfish API to gather more in-depth information in case of failure. Appendix B.8 contains the code for the aforementioned functionality.

This module is stored in the "plugins/modules" directory, and to help Ansible find the module during playbook-execution, an entry to the Ansible configuration file is made. The entry contains the relative path to the modules directory. Listing 33 shows an example task which calls this module to preview the import of a previously exported SCP located in the network share "/nfs" folder. The success or failure of this task can be used to determine further playbook execution, for example, if an "Import SCP" task should be run which will apply the configuration to the server.

This module was developed for the purpose of being able to check if any imported SCP files would be successful instead of making any partial or failed imports. This is especially useful when making any edits to the previously exported file before importing it, and the module serves as a good example to how Ansible Redfish modules can be created for missing functionalities.

```
- name: Import server configuration preview
    import_idrac_scp_preview:
      idrac_ip: "{{ ansible_host }}"
      idrac_user: "{{ ansible_user }}"
      idrac_password: "{{ ansible_password }}"
      share_ip: "129.241.21.201"
      share_name: "/nfs"
      share_type: "NFS"
      scp_file: "192.168.0.125_20210518_170529_scp.json"
      scp_components: "ALL"
```

Listing 33: Example task calling a custom module for previewing the import of server configuration profiles

Get job id details

If the SCP preview task shown in listing 33 fails, a system administrator might be interested in which attributes in the template configuration are not supported on the target host. The job-id returned in the response of this module can be used to query the task-service of the Redfish API to gather detailed information about the task. However, this requires a separate module - 'get_job_id_details', the code of which can be found in the appendix listing B.9.

'get_job_id_details' complements the 'SCP preview' module by querying the Redfish task-service to get more details about the initiated 'SCP preview' task. The module will accept an IP, username, password, job-id, and the instance-ID of the manager resource (ex. iDRAC.Embedded.1) as parameters. The Redfish URI to the task-service resource is then built and a GET request is made. Information about the state of the task (e.g. complete or running), as well as the body of the response is then returned to Ansible. The use of this module is demonstrated in the play "Import SCP preview", shown in listing 34. The job-id returned from the "SCP preview" task is queried multiple times until the task is complete. The Ansible debug-module is used to print the response which will contain information about whether the preview was a success, or which attributes would fail to import.

```yaml
---
- name: Preview server configuration profiles and get job details
  hosts: test1
  connection: local
  vars:
    test_mode: false
  tasks:
    - name: Include host specific variables
      include_role:
        name: common
        tasks_from: include_host_vars
      when: not test_mode

    - name: Preview server configuration profile
      import_idrac_scp_preview:
        idrac_ip: "{{ ansible_host }}"
        idrac_user: "{{ ansible_user }}"
        idrac_password: "{{ ansible_password }}"
        share_ip: "129.241.21.201"
        share_name: "/nfs"
        share_type: "NFS"
        scp_file: "192.168.0.125_20210518_170529_scp.json"
        scp_components: "ALL"
      register: testout

    - name: Get job details
      get_job_details_redfish:
        ip: "{{ ansible_host }}"
        username: "{{ ansible_user }}"
        password: "{{ ansible_password }}"
        job_id: "{{ testout.job_id }}"
      register: output
      until: output.JobState != "Running"
      retries: 10

    - name: print job details output to console
      debug:
        msg: "{{ output }}"
```

Listing 34: Example play which previews the import of a server configuration profile, and retries the 'get_job_id_details' module 10 times or until the job is complete

Chapter 5

Security

This chapter was added as an expansion to the project, and was not part of the requirement specification for the PoC. It was found natural to expand into this area of IT, due to security being a vital consideration in any modern IT organization or project. This chapter presents a short summary of the best security practices published for the iDRAC, a demonstration on the vulnerabilities of the IPMI specification, and a short summary and reflection on these findings.

5.1 Best practices

This section is a short summary of the best security practices for setup and configuration of the iDRAC. As mentioned in section 2.7 - "Security" in chapter 3, there is a lack of best security practices for Redfish easily available online. Additionally, most of the best security practices are surrounding the environment which Redfish is utilized. The reason these best practices revolving around the iDRAC is simply that this is the equipment was provided for the project.

This best practice section is a shortened summary of two sources of information, one being the Dell iDRAC9 white paper [62] and the other being the iDRAC9 Security Configuration Guide [86]. Though most of these recommendations are quite universal, and should be considered for other types of BMCs.

- The iDRAC is not intended to be exposed to the Internet, it was created to be on a separate management network. This is also the case for other BMCs, with both Intel and Supermicro having this as a best practice for BMC [87][88]. Should this not be possible then a dedicated and separate VLAN is recommended.
- It is recommended to use the 'Dedicated Gigabit Ethernet' port on the tower and rack servers to connect the iDRAC to the management network.
- The BMC management network should be isolated from the rest of the internal network using technologies such as firewalls, and limiting access only to authorized administrators.

- The iDRAC has a function for IP blocking which allows an administrator to set the amount of allowable login failures, time frame for these failures, and the amount of time which the IP address is blocked. Additionally IP filtering can be enabled and set to only allow a small amount of preconfigured IP addresses to access the BMC. This would severely limit the ability of an unauthorized attacker gaining access through brute-force attacks. IP blocking and filtering are functions which are present on most other types of BMCs.
- Security configurations for the web server: Redirecting all HTTP requests to HTTPS, enable 256-bit encryption strength, configure TLS 1.2, limit the cipher suites to the strongest possible, use CA signed SSL/TLS certificates, and finally to enable Simple Certificate Enrollment Protocol.
- Additional security authentication such as LDAP or Microsoft Active Directory can and should be used.
- The BMC firmware should be continuously updated to the latest version as this reduces the chance of a known vulnerability attack.
- Disable any unused network interfaces, protocols, or modes of communication. This includes IPMI over LAN, Serial over LAN, SNMP, and Telnet. This limits the attack surface of the iDRAC, as it is only as secure as the least secure protocol used.
- Redfish specific: Dell recommends to utilize the Session-based authentication for Redfish, this is due to the Basic Authentication having the password and username in every API request.

5.2 Comparison of IPMI and Redfish

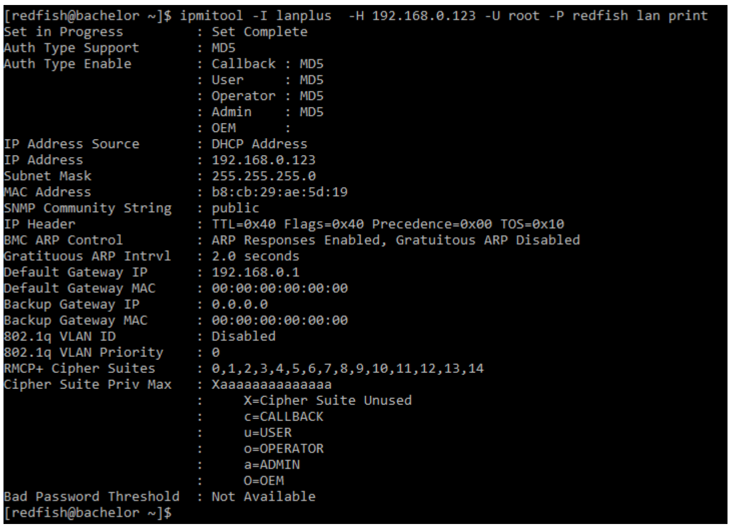
As a comparison and to test the different vulnerabilities mentioned in subsection 2.7.1 - "IPMI Security" in chapter 3, the penetration guide posted by the Rapid7 group for IPMI was utilized [89]. This guide is based on the previously mentioned work of security researcher Dan Farmer.

The limits of this short security test was to utilize only penetration testing and command line tools which are free and openly available. There was also no attempts to uncover any new vulnerabilities, the focus was only on known vulnerabilities. The server which the test was run against was the server on '192.168.0.123' which is a Dell PowerEdge C6420 running iDRAC9 on the BMC. The attacks were ran from the Bachelor HPC login server provided by the employer which is on the same network, though the Wireshark [90] traffic was piped over SSH to a local virtual Kali Linux machine [91] using an image[92] provided by Offensive Security [93].

The tools used were Metasploit [94] and wireshark. Metasploit was used for the easily available IPMI modules, while Wireshark was used to minotor the network traffic and attempt to sniff the packages. The iDRAC did not have any security hardening past the default configuration provided.

5.2.1 IPMI vulnerability

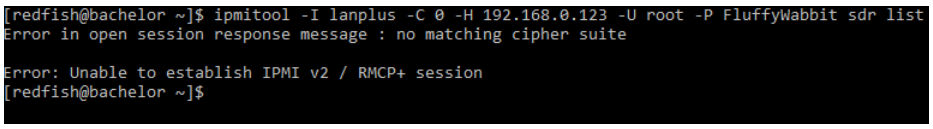
Before any testing was made there was passed a command, 'lan print', to check the current configuration of the IPMI interface using IPMItool. As shown in figure 5.1 it revealed information such as cipher 0 being disabled on the interface, MD5 being used for encryption, as well as some other configuration information. This though is a legitimate and authorized command used with IPMItools, which had the correct credentials. An unauthorized user without credentials would not have access to this command, and would require to find this information otherwise.



```
[redfish@bachelor ~]$ ipmitool -I lanplus  -H 192.168.0.123 -U root -P redfish lan print
Set in Progress         : Set Complete
Auth Type Support       : MD5
Auth Type Enable        : Callback : MD5
                        : User     : MD5
                        : Operator : MD5
                        : Admin    : MD5
                        : OEM      :
IP Address Source       : DHCP Address
IP Address              : 192.168.0.123
Subnet Mask             : 255.255.255.0
MAC Address             : b8:cb:29:ae:5d:19
SNMP Community String   : public
IP Header               : TTL=0x40 Flags=0x40 Precedence=0x00 TOS=0x10
BMC ARP Control         : ARP Responses Enabled, Gratuitous ARP Disabled
Gratituous ARP Intrvl   : 2.0 seconds
Default Gateway IP      : 192.168.0.1
Default Gateway MAC     : 00:00:00:00:00:00
Backup Gateway IP       : 0.0.0.0
Backup Gateway MAC      : 00:00:00:00:00:00
802.1q VLAN ID          : Disabled
802.1q VLAN Priority    : 0
RMCP+ Cipher Suites     : 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14
Cipher Suite Priv Max   : Xaaaaaaaaaaaaaa
                        :     X=Cipher Suite Unused
                        :     c=CALLBACK
                        :     u=USER
                        :     o=OPERATOR
                        :     a=ADMIN
                        :     O=OEM
Bad Password Threshold  : Not Available
[redfish@bachelor ~]$
```

Figure 5.1: Cipher 0 disabled

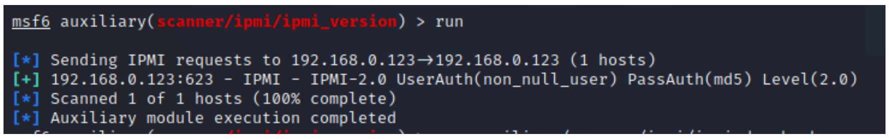
The first attempt was to uncover if the cipher 0 protocol was in use on the server, and it was attempted to authenticate through a regular command line. The input ' `-C 0` ' means that it attempts to authenticate using cipher 0, which if enabled only requires a legitimate user account and disregards the password. As shown in the figure 5.2 this fails as the cipher suite is not enabled.



```
[redfish@bachelor ~]$ ipmitool -I lanplus -C 0 -H 192.168.0.123 -U root -P FluffyWabbit sdr list
Error in open session response message : no matching cipher suite

Error: Unable to establish IPMI v2 / RMCP+ session
[redfish@bachelor ~]$
```

Figure 5.2: Attempt to use Cipher 0 through command line

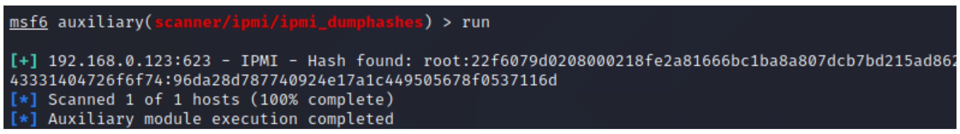
Furthermore the Metasploit module: ' `auxiliary/scanner/ipmi/ipmi_version` ' was used. This module is able to be used against an IP address range, which would reveal if there are any machines running IPMI on port 623 in a network. It will in addition extract the information regarding the IPMI version being ran on the machine, as well as which forms of supported authentication[89]. In the figure 5.3 just a single iDRAC is scanned and this reveals that IPMI 2.0 is the only version supported, as well as the password being hashed in MD5, and there being no null user available.



```
msf6 auxiliary(scanner/ipmi/ipmi_version) > run

[*] Sending IPMI requests to 192.168.0.123->192.168.0.123 (1 hosts)
[+] 192.168.0.123:623 - IPMI - IPMI-2.0 UserAuth(non_null_user) PassAuth(md5) Level(2.0)
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figure 5.3: Metasploit module revealing information

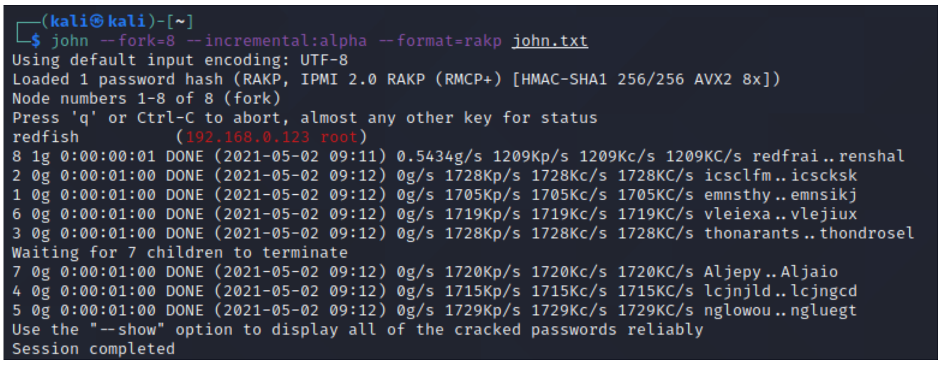
Then the module for exporting hashes from the machine running IPMI was used, as mentioned by Farmer [95] there is a vulnerability in the IPMI specification. Essentially the BMC will give the hashed password for any user requested prior to any authentication, and this is a core part of the specification which means there is no good workaround other than properly isolating the BMC. Simply by using the ' `IPMI_Dumphashes` ' module on metasploit towards an IP range with a legitimate user name it will freely disclose the password hash. This is shown in the figure 5.4, as well as the hash for the password being returned.



```
msf6 auxiliary(scanner/ipmi/ipmi_dumphashes) > run

[+] 192.168.0.123:623 - IPMI - Hash found: root:22f6079d0208000218fe2a81666bc1ba8a807dcb7bd215ad862
43331404726f6f74:96da28d787740924e17a1c449505678f0537116d
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Figure 5.4: IPMI disclosing password hash for user root

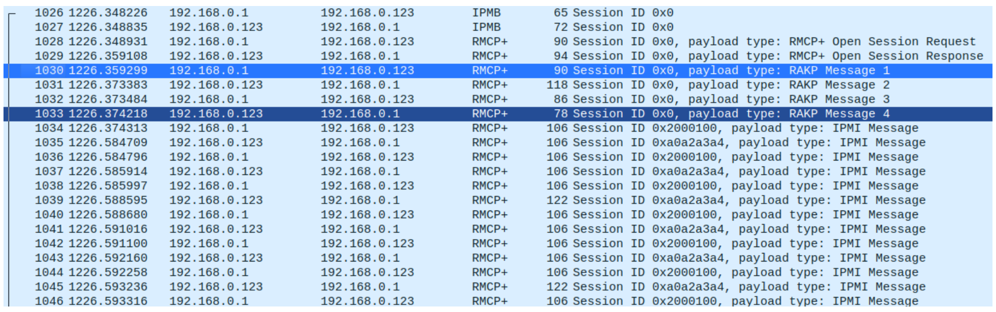
This hash can then be attempted brute forced offline without any more interaction with the BMC required. In the figure 5.5 this hash is given to the password cracker tool 'John the Ripper', and the password is quickly returned. The speed of the password being cracked due to it being a single word 'redfish', which is both simple and short. A more complex password would take longer, the time elapsed rising with the complexity. Though with the hash extracted this means essentially only the computational power of the attacker and the complexity of the password itself stands as hindrances.



```
┌──(kali㉿kali)-[~]
└─$ john --fork=8 --incremental:alpha --format=rakp john.txt
Using default input encoding: UTF-8
Loaded 1 password hash (RAKP, IPMI 2.0 RAKP (RMCP+) [HMAC-SHA1 256/256 AVX2 8x])
Node numbers 1-8 of 8 (fork)
Press 'q' or Ctrl-C to abort, almost any other key for status
redfish          (192.168.0.123 root)
8 1g 0:00:00:01 DONE (2021-05-02 09:11) 0.5434g/s 1209Kp/s 1209Kc/s 1209KC/s redfrai..renshal
2 0g 0:00:01:00 DONE (2021-05-02 09:12) 0g/s 1728Kp/s 1728Kc/s 1728KC/s icsclfm..icscksk
1 0g 0:00:01:00 DONE (2021-05-02 09:12) 0g/s 1705Kp/s 1705Kc/s 1705KC/s emnsthy..emnsikj
6 0g 0:00:01:00 DONE (2021-05-02 09:12) 0g/s 1719Kp/s 1719Kc/s 1719KC/s vleiexa..vlejiux
3 0g 0:00:01:00 DONE (2021-05-02 09:12) 0g/s 1728Kp/s 1728Kc/s 1728KC/s thonarants..thondrosel
Waiting for 7 children to terminate
7 0g 0:00:01:00 DONE (2021-05-02 09:12) 0g/s 1720Kp/s 1720Kc/s 1720KC/s Aljepy..Aljaio
4 0g 0:00:01:00 DONE (2021-05-02 09:12) 0g/s 1715Kp/s 1715Kc/s 1715KC/s lcjnjld..lcjngcd
5 0g 0:00:01:00 DONE (2021-05-02 09:12) 0g/s 1729Kp/s 1729Kc/s 1729KC/s nglowou..ngluegt
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Figure 5.5: John the Ripper Cracking the Password for 198.168.0.123

In the figure 5.6 Wireshark was being ran in an attempt to sniff the packets being transferred on the network. In this case the password and username was passed in the IPMItool traffic, though this is not visible to an attacker as the traffic is encrypted using RMCP+. Additionally on a properly configured network infrastructure this type of traffic would most likely not be readily available for Wireshark, as the traffic is directly routed or switched between the master node and the BMC.



```
1026 1226.348226   192.168.0.1      192.168.0.123    IPMB      65 Session ID 0x0
1027 1226.348835   192.168.0.123    192.168.0.1      IPMB      72 Session ID 0x0
1028 1226.348931   192.168.0.1      192.168.0.123    RMCP+     90 Session ID 0x0, payload type: RMCP+ Open Session Request
1029 1226.359108   192.168.0.123    192.168.0.1      RMCP+     94 Session ID 0x0, payload type: RMCP+ Open Session Response
1030 1226.359299   192.168.0.1      192.168.0.123    RMCP+     90 Session ID 0x0, payload type: RAKP Message 1
1031 1226.373383   192.168.0.123    192.168.0.1      RMCP+    118 Session ID 0x0, payload type: RAKP Message 2
1032 1226.373484   192.168.0.1      192.168.0.123    RMCP+     86 Session ID 0x0, payload type: RAKP Message 3
1033 1226.374218   192.168.0.123    192.168.0.1      RMCP+     78 Session ID 0x0, payload type: RAKP Message 4
1034 1226.374313   192.168.0.1      192.168.0.123    RMCP+    106 Session ID 0x2000100, payload type: IPMI Message
1035 1226.584709   192.168.0.123    192.168.0.1      RMCP+    106 Session ID 0xa0a2a3a4, payload type: IPMI Message
1036 1226.584796   192.168.0.1      192.168.0.123    RMCP+    106 Session ID 0x2000100, payload type: IPMI Message
1037 1226.585914   192.168.0.123    192.168.0.1      RMCP+    106 Session ID 0xa0a2a3a4, payload type: IPMI Message
1038 1226.585997   192.168.0.1      192.168.0.123    RMCP+    106 Session ID 0x2000100, payload type: IPMI Message
1039 1226.588595   192.168.0.123    192.168.0.1      RMCP+    122 Session ID 0xa0a2a3a4, payload type: IPMI Message
1040 1226.588680   192.168.0.1      192.168.0.123    RMCP+    106 Session ID 0x2000100, payload type: IPMI Message
1041 1226.591016   192.168.0.123    192.168.0.1      RMCP+    106 Session ID 0xa0a2a3a4, payload type: IPMI Message
1042 1226.591100   192.168.0.1      192.168.0.123    RMCP+    106 Session ID 0x2000100, payload type: IPMI Message
1043 1226.592160   192.168.0.123    192.168.0.1      RMCP+    106 Session ID 0xa0a2a3a4, payload type: IPMI Message
1044 1226.592258   192.168.0.1      192.168.0.123    RMCP+    106 Session ID 0x2000100, payload type: IPMI Message
1045 1226.593236   192.168.0.123    192.168.0.1      RMCP+    122 Session ID 0xa0a2a3a4, payload type: IPMI Message
1046 1226.593316   192.168.0.1      192.168.0.123    RMCP+    106 Session ID 0x2000100, payload type: IPMI Message
```

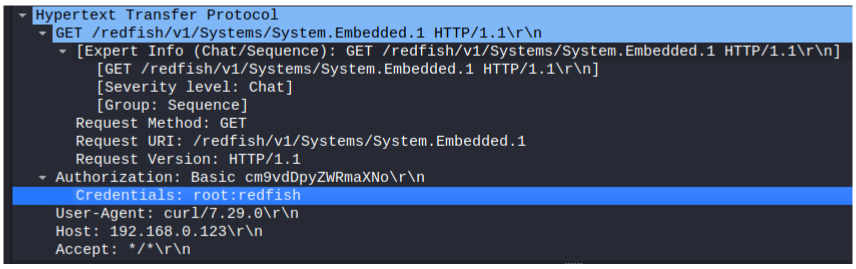
Figure 5.6: IPMI traffic from Bachelor login machine to 192.168.0.123

5.2.2 Redfish

Redfish itself is a rather new technology, and there are few to none easily available exploits or known vulnerabilities about the specification. A few searches for vulnerabilities on the Internet did not reveal any which were relevant or easily accessible. Neither did Metasploit have any inbuilt modules for Redfish available.

The few Redfish related vulnerabilities found in the iDRAC had already been patched and were no longer relevant to a modern and updated iDRAC.

The only security consideration which was relevant to Redfish itself was the usage of HTTP, as when API calls using basic authentication would reveal credentials in a sniffed package as shown in figure 5.7. In this image Wireshark has been used to capture a Redfish API call towards the iDRAC at 192.168.0.123, and both of the credentials are clearly visible in the package.



```
▾ Hypertext Transfer Protocol
  ▾ GET /redfish/v1/Systems/System.Embedded.1 HTTP/1.1\r\n
    ▾ [Expert Info (Chat/Sequence): GET /redfish/v1/Systems/System.Embedded.1 HTTP/1.1\r\n]
        [GET /redfish/v1/Systems/System.Embedded.1 HTTP/1.1\r\n]
        [Severity level: Chat]
        [Group: Sequence]
      Request Method: GET
      Request URI: /redfish/v1/Systems/System.Embedded.1
      Request Version: HTTP/1.1
  ▾ Authorization: Basic cm9vdDpyZWRmaXNo\r\n
      Credentials: root:redfish
    User-Agent: curl/7.29.0\r\n
    Host: 192.168.0.123\r\n
    Accept: */*\r\n
```

Figure 5.7: Cutout from Wireshark showing Redfish traffic with credentials

This traffic is intended to run on an internal management network for the BMCs itself, and is not meant to be exposed to the internet or any external users. Though it is considered best practice to use HTTPS as these credentials alongside the rest of the traffic would then be encrypted. In case of an infected machine on the network or a malicious actor gaining access - at the very least the password and username for the Redfish interface would not be freely available in the network packets. In addition, as previously mentioned in section 5.1 it is considered best practice to use session-based authentication, which alongside with HTTPS would both reduce the amount of credentials being passed and also encrypt them when they do.

5.3 Reflections

There are several blatant security issues prevalent in the IPMI standard, and these are vulnerabilities which have been present since at least 2013. In the previous subsection 5.2.1 it was shown how simple it is to launch attacks against this outdated specification. When one considers that the IPMI specification is no longer developed, and with the developers recommending a switch to newer specifications such as Redfish it becomes difficult to defend not making this switch.

IPMI has several legacy functions, such as cipher 0, which must always be disabled either by the vendor themselves or the administrator. It also though has other vulnerabilities such as the hash exposure, which is part of how the specification operates and cannot be disabled.

Redfish as opposed to IPMI is a specification which is actively developed, and is rather new to the market. There was not found any unremedied vulnerabilities in the specification, and new ones will be patches as they are uncovered. This is not an option for IPMI as it is no longer being developed, and Dell recommends to disable IPMI over LAN [86] as there are "known security limitations inherent in the protocol". When taking security into consideration there are several strong reasons to switch to Redfish and disable IPMI for good.

In section 5.1 - 'Best Practices', several best practice security configurations were listed. Some of these practices are on the networking or organizational level, which are out of scope for this project. Though the ones found related to the iDRAC itself are usable for creating a iDRAC template which is more secure than the one currently in use. Several of these more secure practices are simply to disable unused protocols, and in a live environment one would have to identified those which are required and those which are unnecessary.

Chapter 6

Deployment

Chapter 4 - "Implementation" served to demonstrate how Redfish can be used to accomplish common OOB management tasks in an ad-hoc manner, in addition to demonstrating the execution of Ansible playbooks. In chapter 5 - "Security", a number of best practises for the configuration of iDRAC servers were presented. This chapter builds on both of these chapters to demonstrate how Ansible playbooks, roles, and modules can be combined to perform larger and more complicated tasks using Redfish. The two combinations presented are exporting and importing server configuration from one server to another, and demonstrating deployment of a predefined BIOS and best practice iDRAC configurations on an out of the box Dell C6420 server using Ansible with open-source Redfish modules and custom playbooks.

6.1 Configuration

There are two lists beneath this paragraph. Both of the lists are based on the most commonly applied BIOS attributes applied by the employer. While second list is named 'iDRAC' also incorporates the best practice findings in chapter 5 - "Security" section 5.1. The Employer originally specified that Serial over LAN (SoL) was a functionality which was commonly enabled. SoL is an IPMI specific function which allows for the redirection of BMC traffic over an IPMI session [15]. It was decided to disable this function in order to follow the best practices, as it is not part of the Redfish specification or in other ways required for Redfish to function.

BIOS

- Performance Mode: Performance

iDRAC:

- Changing VLAN settings
- BMC Interface: Shared

- Change user and password
- IPMI over LAN: Disabled
- Serial over LAN console redirection: Disabled
- Telnet: Disabled
- SNMP: Disabled
- Redirect all web-server HTTP requests to HTTPS
- Configure TLS 1.2
- Enable 256-bit encryption strength
- Limit cipher suites to strongest possible
- IP blocking on failed attempts
- IP address filtering

6.2 Vendor specific attribute names

The two lists in section 6.1 - "Configurations" have been mapped to the equivalent iDRAC specific attribute names and accepted values. For this demonstration, the iDRAC9 attribute registry documentation [77] was used to find the specific values as well as their dependencies. The employer requires the performance mode to be set to 'Performance', which requires the 'SysProfile' attribute to be set to 'PerfOptimized'. Relevant BIOS attribute names and values are shown in 6.1, while the iDRAC attributes and values are shown in 6.2.

Bios Attributes	**Value**
SysProfile	PerfOptimized

Figure 6.1: Bios Attributes

iDRAC Attributes	**Value**
IPMISOL.1.Enable	Disabled
Users.2.SolEnable	Disabled
SerialRedirection.1.Enable	Enabled[1]
NIC.1.VLanEnable	Enabled
NIC.1.VLanID	1
NIC.1.Selection	LOM1
Users.2.UserName	root
Users.2.Password	redfish
Telnet.1.Enable	Disabled
SNMP.1.AgentEnable	Disabled
WebServer.1.HttpsRedirection	Enabled
WebServer.1.TLSProtocol	TLS 1.2 Only
IPBlocking.1.BlockEnable	Enabled
IPBlocking.1.FailCount	3
IPBlocking.1.FailWindow	60
IPBlocking.1.PenaltyTime	60
SSHCrypto.1.KexAlgorithms	curve25519-sha256,curve25519-sha256@libssh.org,ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,diffie-hellman-group-exchange-sha256,diffie-hellman-group16-sha512,diffie-hellman-group18-sha512,diffie-hellman-group14-sha256
SSHCrypto.1.MACs	umac-128-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha1-etm@openssh.com,umac-128@openssh.com,hmac-sha2-256,hmac-sha2-512,hmac-sha1
SSHCrypto.1.Ciphers	chacha20-poly1305@openssh.com,aes128-ctr,aes192-ctr,aes256-ctr,aes128-gcm@openssh.com,aes256-gcm@openssh.com
SSHCrypto.1.HostKeyAlgorithms	ssh-rsa,rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256,ssh-ed25519

Figure 6.2: iDRAC Attributes

1. NIC.1.VlanEnable is disabled in the Proof of Concept, and in the demonstrations as to not interfere with any of the VLAN settings.

6.3 Demonstration

The configuration specified in 6.1 (BIOS), and 6.2 (iDRAC), are applied by running Ansible playbooks which uses the following Ansible modules:

Redfish config[79]: A module for managing BIOS and OOB controller settings by building Redfish URIs locally and sending HTTP(s) requests to the Redfish API of specified host(s). Part of the community general collection (v2.5.1)[40].

Redfish command[80]: A module for sending commands to OOB controller (Reboot, power on/off, etc) by building Redfish URIs locally and sending HTTP(s) requests to the Redfish API of specified host(s). Part of the community general collection (v2.5.1) [40].

iDRAC Redfish command[47]: A module for scheduling a BIOS setting update by building Redfish URIs locally and sending HTTP(s) requests to the Redfish API of specified host(s). Part of the community general collection (v2.5.1) [40]. Only required for Dell servers.

6.4 Deployment

Within this section is a demonstration of how the different playbooks demonstrated in chapter 4 - "Implementation" can be used together for managing a BMC environment. As previously mentioned in subsection 4.5.4 - "Discovery", IP address management is outside of scope. This demonstration does not take into consideration how the IP address has been discovered, nor how it has been managed. It assumes that the IP has somehow been discovered and added to the inventory file, and that the machine is an out of the box Dell server with iDRAC9. The longer outputs have been placed within the appendix for ease of reading, with references linking them back.

6.4.1 Extracting then deploying server configuration

The first demonstration is using the playbooks for extracting and importing server configurations which can be found in the subsections: 4.5.2 - "Export Server Configuration" and 4.5.3 - "Import Server Configuration". First the `export_server_config_profile` playbook is ran against one of the iDRAC9 servers, and as can be seen in listing 35 is successful. This creates a JSON file with the configuration details of the first iDRAC server in the '/tmp/' folder.
The preview SCP module from section 4.5.7 - "Server Configuration Profile Preview" is then used to check if the previously exported file is compatible with the server it is being pushed towards. This returns the output which can be found in the appendix B.10, showing that imported configuration is compatible and should be successful. With this information the playbook from subsection 4.5.3 - "Import Server Configuration" is ran against the server, and in listing 36 is shown to have been successful. These playbooks are set up with hardcoded values, and would require some changes for implementation in a different environment.

```
[root@bachelor redfish-ansible]#ansible-playbook -i inventory/
↪   playbooks/export_server_config_profile.yml

PLAY [Export server configuration profile]
*******************************************************************

TASK [Include task for including host specific variables]
*******************************************************************
included: /home/redfish/test/git/bachelor_repo/redfish-
↪   ansible/playbooks/tasks/include_host_vars.yml for
↪   idrac4

TASK [Include host specific Ansible Vault encrypted variables]
*******************************************************************
ok: [idrac4]

TASK [Export Server Configuration Profile to local path]
*******************************************************************
ok: [idrac4]

PLAY RECAP
*******************************************************************
idrac4                    : ok=3    changed=0    unreachable=0    failed=0
↪   skipped=0    rescued=0    ignored=0
```

Listing 35: Export playbook successfully ran against iDRAC

6.4.2 Deploying BIOS and iDRAC configuration against a single server

In this demonstration it uses the roles and related playbooks shown in the subsection 4.5.6 - "Playbook and roles" from chapter 4. For this demonstration a custom playbook named 'server_setup.yml' has been created which uses both of the roles in the 'Playbook and Roles' subsection. This can be seen in listing 37, the playbook first runs a pre-task which gathers information about the BMC, and registers this to a variable called 'result'. It then runs the role for setting up the BIOS on the server, both of which the code and role description can be found in subsection 4.5.6 - "Role: BIOS_idrac_settings" and its listing 29.

The next role, 'idrac_settings', is only ran when the 'product' from the 'result' variable is equal to 'Integrated Dell Remote Access Controller'. This example shows that with more such conditions one could have several additional roles within the Ansible Playbook, and use it for configuring new devices as they are discovered. In this example this is an iDRAC, so the playbook continues. It changes the iDRAC attributes to those shown in figure 6.2. The successful output from this playbook can be found in the appendix B.11.

```
[root@bachelor redfish-ansible]# ansible-playbook -i inventory/
↪   playbooks/import_server_config_profile.yml

PLAY [Import server configuration profile]
**********************************************************************

TASK [Include task for including host specific variables.]
**********************************************************************
included: /home/redfish/test/git/bachelor_repo/redfish-
↪   ansible/playbooks/tasks/include_host_vars.yml for
↪   idrac3

TASK [Include host specific Ansible Vault encrypted variables]
**********************************************************************

TASK [Import Server Configuration Profile]
**********************************************************************
changed: [idrac3]

PLAY RECAP
**********************************************************************
idrac3                    : ok=3    changed=1    unreachable=0    failed=0
↪   skipped=0    rescued=0    ignored=0
```

Listing 36: Import playbook successfully ran against iDRAC

6.4.3 Deploying BIOS and iDRAC configurations against two servers simultaneously

In this example the same playbook from the previous subsection 6.4.2 is ran with some slight changes. These changes are to target the two iDRAC hosts instead of one, and that one of the iDRACs has an already correctly configured BIOS. The output from this playbook being ran can be found in the appendix B.12, the output shows that the configurations are successfully launched against both servers. It also shows how simple it is to scale upwards with Ansible and Redfish to target more hosts. In this case the unconfigured server has its BIOS changed, while the one with the already correct BIOS configuration skips this step. Additionally both of the servers have their iDRAC settings changed correctly.

```yaml
---
- hosts: test1
  connection: local
  gather_facts: false
  name: Apply BMC settings
  vars:
    test_mode: False
  pre_tasks:
    - name: Get server information
      ansible.builtin.uri:
        url: "https://{{ ansible_host }}/redfish/v1"
        method: GET
        headers:
          Accept: "application/json"
          Content-Type: "application/json"
        validate_certs: false
      register: result

  roles:
    - role: bios_idrac_settings
    - role: idrac_settings
      when: result.json['Product'] == "Integrated Dell Remote Access
       ↪  Controller"
```

Listing 37: Playbook for deploying a default iDRAC bios and server config

Chapter 7

Testing

Sommerville describes testing in his book "Software Engineering" as the practise of demonstrating that the software meets its requirements. This can be done by testing the product in two ways; validation and verification testing. Validation testing is the process of testing if the right product is being built. This is a more general process, where the goal is to ensure that the product meets the customer expectations [11]. In the context of the Proof of Concept, this is a continuous process where missing functionality as specified by the employer is tracked in the product backlog. Biweekly meetings with the employer is used to discuss the progress and to gauge if there are any expectations of the system that is not reflected by the the requirements. For each iteration of the development process (sprint meetings), the project group meets to discuss if the right product (PoC) is actually being built.

Verification testing on the other hand, is concerned with if the product is being built right [11]. These are tests to ensure that the product meets its functional and non-functional requirements. As mentioned previously, such software engineering principles are a core part of the infrastructure-as-code approach to IT infrastructure management. Integrating manual and/or automated verification tests to the development workflow, makes it possible to achieve the IaC core principle of embracing frequent changes and improvements. Testing keeps the quality of the system high by identifying errors as soon as they are made so they can be immediately fixed, and any changes can be made frequently with accuracy and confidence. By testing frequently and while still working on changes, Morris argues that everything is still fresh in mind, making small problems quick to find and fix. Morris continues by describing how testing can be integrated in agile processes which strive to be flexible to be able to adapt to changing requirements. He describes it as a change-test-fix loop, which is adopted in figure 7.1, to show that testing takes places continuously as changes are made. There is no formula to what type of tests should be used in an infrastructure code base, so to get started, it is best to begin by a minimum amount of tests which is required, and then introduce new types of tests as they are needed. [14].

The rest of this chapter describes the tests implemented or not implemented to test the PoC, and the reasoning behind them. The reader may notice a lack of focus non-functional requirements (system performance, memory utilization, etc.). This is because there were no non-functional requirements to this PoC.

7.1 Testing Ansible playbooks

According to their own documentation, Ansible is designed to be a "fail-fast" and ordered system[96]. This means that tests can be embedded into the playbooks by declaring a desired state, such as if a service should be enabled, started, installed, etc. If a desired state is not reached during task execution, the task will return as failed. Some tasks in this project defines a desired state by defining which attributes should be applied to a server configuration. A successful execution of this task should call a new task which reboots the system to apply these changes. Instead of having to check the return codes of the request before reboot, adding a condition that checks if the task which applies the configuration was successful is sufficient, as Ansible is checking return codes automatically. In other words, no external testing tools should be necessary to validate the workings of an Ansible playbook.

The syntax of the playbooks and tasks themselves however, can be verified by checking for syntax-errors and code quality. The Ansible CLI tool `ansible-playbook` [97] has the `–syntax-check` option which performs a syntax-check on the playbook(s) specified without executing them.

Code quality is tested using static code analysis tools. The tool `ansible-lint` is a command-line tool used to parse playbooks, roles or collections and checks for common errors or bad habits which can lead to bugs or just code that is harder to maintain [98]

In software testing, and therefore by extension Infrastructure as Code, unit tests are often used to execute small subsections of a program to make sure they run correctly [99]. As mentioned in section 2.5 about Ansible, playbooks consist of a series of tasks. These tasks can be seen as the small units (subsections) that can be tested, but because of the Ansible mechanics described in the beginning of this section, no external tools are needed to perform such unit testing. Independent, common tasks can be separated into different files and included at runtime in playbooks, and its functionality and code quality can be tested with methods described above.

The purpose of the lab-environment described in earlier in section 4.3, is to be able to verify that the playbooks and tasks behave as they should by running them on one or multiple hosts in that environment. These tests are done continuously and manually. By observing the CLI output while running the Ansible playbooks, the developer is quickly able to determine if a task failed or succeeded and continue the workflow as pictured in figure 7.1.

7.2 Continuous integration

Continuous integration is the practise of frequently integrating and testing all changes to a system as they are being developed [99]. For the development of the PoC, the CI tool ' `TravisCI` ' [100] was used to enable this practise. Each commit to the git repository will trigger a CI process which builds a Linux Python environment, installs `Ansible` and `ansible-lint` and performs syntax and code quality tests on all playbooks or tasks in the branch, giving relatively quick feedback to the developer if the pushed changes succeeded or failed. The full configuration can be found in B.15.

Because the whole purpose of the lab environment is be the testing ground of the PoC, a complete, virtual, testing environment was not considered to be necessary for this project. However, DMTF has developed two tools which can help in cases where such environment is necessary. The "Redfish mockup creator" [101] creates a mockup of a live Redfish service by saving all resources it finds on the specified target host. This mockup can then be used to create a web server with the tool "Redfish mockup server" [102] to simulate Redfish API requests and responses on an identical, but virtual machine. These tools could be used to improve the testing suite of a Redfish client development project by integrating them in CI tools such as TravisCI so tests can be run in a virtual environment instead of bare-metal servers.

A problem can arise when a CI tool, in this case, TravisCI, tries to run playbooks that calls variables which are encrypted using Ansible Vault. Since the CI tool does not possess the decryption key, the tests will fail before any syntax or lint testing can be done. Author of "Ansible for devops", Jeff Geerling, wrote a blog-post discussing the possible solutions to this problem. Like anyone else thinking of security as a continuous process, he did not feel comfortable storing the decryption key in a third-party system, where he could not guarantee its security [103]. The solution used in this project is based on Geerling's blog-post. By adding the variable `check_mode` and adding the check `when not test_mode` to the task which includes the encrypted variables to the play, it is possible to avoid importing those variable all together by setting the variable to true and passing it to the task when running it in a CI system. Because the encrypted variables will not be imported, the tests can proceed, and the decryption key never have to be stored in a third-party CI system.

7.3 Testing Ansible modules

This project has shown the development of two Ansible modules which did not exist before: "Get job details", and "Import server configuration profile preview". If such modules were to be developed for a real production environment, a more comprehensive testing-process could and should be implemented into the development process. This involves writing testing-requirements and unit-tests based on those requirements found in subsection 8.2.1, as well as performing sanity-tests to make sure the code conforms to the Ansible coding standards requirements [104].

Because the nature of a PoC is to merely show the principle and the feasibility of the end result, as well as to spare the limited project resources, such a large task was taken out of consideration for this project. Some error-handling like failed HTTP requests was embedded into the code, so the success or failure of the module is observed when running the Ansible tasks which calls the PoC modules.

7.4 Testing summary

Summarized, the testing workflow for the development of this PoC, is described in figure 7.1. The local feedback loop is the practise of integrating testing into the development workflow as described in the beginning of this chapter. The functional requirements, playbook/task syntax and behaviour is manually tested, and frequently committed to the code-repository to trigger an automated universal test. For the moment, these tests include syntax and code quality tests, and if accepted, the developer can merge the `testing` and `main` branches and move on to a new task from the product backlog, repeating the process. Validation is a continuous process not showed in figure 7.1, but is continuously revised and discussed throughout the project period.

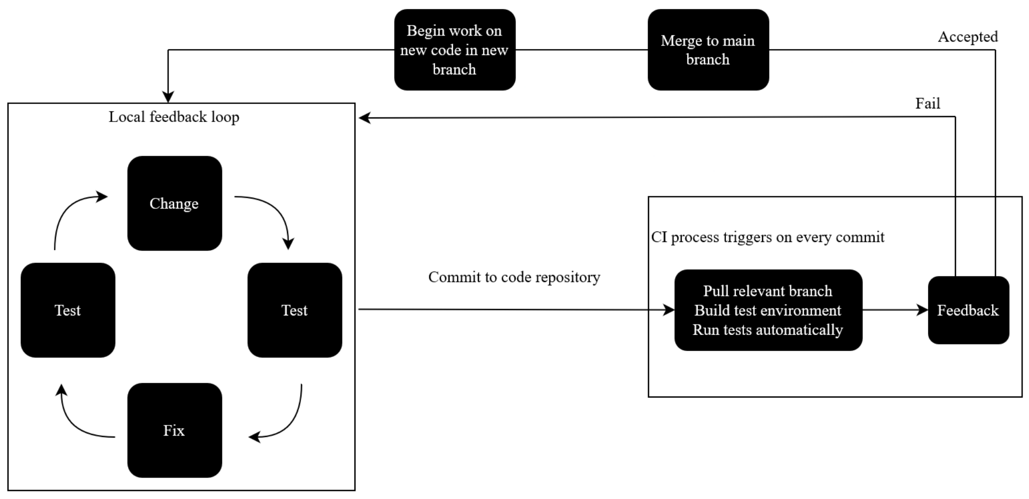


Figure 7.1: A figure showing the testing workflow of this project

Chapter 8

Discussion

This chapter contains the groups reflections on the development process and difficulties encountered during the project with the different technologies. It also contains the groups thoughts about future work related to this thesis, some of which is relevant for the employer while the other is more general in nature.

8.1 iDRAC limitations and challenges

From start to finish, the thesis group has had no physical access to the hardware in the development environment. The setup of the servers, switch and cables were all done by the employer. There are both pros and cons related to delegating setup of the development environment to a group located elsewhere, the biggest pro being the ability to focus on Redfish and Ansible rather than setup and troubleshooting of hardware. The largest con on the other hand was that new configurations could not be freely deployed, as to not crash or render the server unreachable. This was due to having to contact the employer and request for them to remediate such issues, which especially outside of regular working hours would have to wait for the next workday.

Initially only one BMC was provided to the group in the form of a Poweredge R720 server with the iDRAC7, when this was found to not be sufficient a second identical server was provided. The iDRAC7 R720 has Redfish functionality with firmware version 2.40.40.40 and beyond, however the R720 model was released in March of 2012. Although it is still a good machine, 9 years is a lot of time in the technology industry and when one considers that the first Redfish specification was published in 2015 there are some factors that come into play. These factors are as follows; the R720 is no longer being updated, which makes it incompatible with multiple Ansible modules. In addition, there are less Redfish related documentations compared to the iDRAC9. Initially this delayed the development as there was a lack in the iDRAC7 Redfish implementation, and time was spent troubleshooting iDRAC7 specific issues. Specifically there was considerable time spent troubleshooting an issue with importing configurations. This issue was fixed in a later firmware version [105], a version not supported by the iDRAC7 model. This made the iDRAC7 unable to deliver one of the tasks set by the employer, which caused the group to eventually reach the decision that the iDRAC7 was unsuited for the end goal of this project.

The iDRAC9 on the C6420 server on the other hand supports most Redfish functionality and implementations. In addition, existing public Ansible modules for Redfish management are compatible with the iDRAC9, with the exception of a few vendor specific modules. The documentation provided by Dell pertaining to the iDRAC9 was overall sufficient, and both the server and iDRAC9 firmware are actively updated. It would have been better to start with the iDRAC9 from the beginning as it has a full implementation of Redfish, meaning less time would have been spent troubleshooting the iDRAC7. Once the iDRAC9 was provided and it was found to have better compatibility to the project, the decision to focus the development solely on the iDRAC9 was made. Though with Redfish being an open source specification we did not and could not rule out the possibility that other vendors might have had a better implementation, better documentation, and/or better product support, all of which are important factors to acknowledge.

The one limitation which was found to be present on both of the iDRAC versions was that the Redfish simple service discovery protocol was not implemented, and that Dell instead has their own proprietary service discovery protocol. This made it so that the group did not attempt any implementation of automatic service discovery and deployment into the Proof of Concept to fully automate configuration of out-of-the-box servers, as this would have been very vendor specific and not Redfish related. The thesis did however investigate how Redfish could be incorporated into a parts of the service discovery. This was presented in 4.5.4 - "Discovery" chapter 4, and it showed how the API could be used for identifying if Redfish is present on a arbitrary IP address and what type of product it is. The part which has been left untouched is the IP discovery and management of service discovery, and a possible continuation or future work of this thesis could be to research and incorporate open source alternatives to IP address discovery and management, and combine those with the presented PoC.

8.2 Expansions

The group decided to expand the thesis a bit further than the original assignment provided by the Idun HPC group. Amongst these expansions was a chapter on security relevant for the project, and development of two modules related to configuration testing.

8.2.1 Expanding to security

It was decided to expand the project into security, this was natural as half of the study program is related to information security, and there was an interest to delve a bit deeper into this particular aspect of the task. IT security is a rather broad subject, and it was decided to narrow it down to a single chapter within the thesis. Therefore the group limited the contents of this chapter to mainly two things; A summary of the most important security best practices for BMCs and a slight delve into vulnerabilities of the two BMC specifications. Finally it was narrowed further by limiting it to known vulnerabilities, and using exploits provided by open source penetration testing tools. This was to make it possible to complete the tasks in a reasonable time, and also as the group wanted to check for themselves if the security vulnerabilities in IPMI discovered by Farmer were still relevant.

8.2.2 Expanding to module-development

All Redfish functionality that the employer needed a Proof of Concept of was available as modules in the Ansible community collection "General". There could however be cases where modules for a specific action or task in the day to day management of servers does not exist yet.

While looking for a way to test configurations on a server without the fear of losing connectivity, it was discovered that the function for previewing the import of server configuration profiles on Dell PowerEdge servers was not implemented in any open-source Ansible modules. Although simply using the built-in Ansible functionality for creating HTTP-requests could have dissolved the need for an own module, this was seen as an opportunity to demonstrate how an Ansible module could be developed, tested, and used in a local environment for cases where simple or more complex Redfish-tasks need to be automated.

The task was added to the project backlog, defining the functionalities needed: accept a set of defined parameters, and do a GET request to a Redfish URI. The Redfish resource in question could only accept configuration files from a network file-share, and a decision was made to focus the implementation on module-functionality for NFS. This was because of time-constraints, and NFS is the go-to network share for linux-based environments. The result serves its purpose as a Proof of Concept module to demonstrate the possibilities when a Redfish functionality is not present in any open source modules.

8.3 Future work

The group has compiled two lists for future work, the first is a more general list for the furtherment of this type of study. This list is based on areas which the group has found BMC research to lack, areas which the group was unable to delve into due to scope, and due to the limitations of the project. The second list on the other hand is recommendations for future work for the HPC group in Idun in their use of Redfish.

8.3.1 Future work in general

- Redfish is a rather young specification which has just recently seen more widespread use. This project has mostly looked at the known vulnerabilities of the IPMI specification, and the security options of Redfish. Future work should take this into consideration, and attempt to uncover any security issues with the specification or vendor implementation of Redfish.
- A look into the best practices both for optimization and security in the usage of Redfish and Ansible in the HPC environment.
- BMCs are an area of computing which have been left relatively vulnerable for several years, and most vendors do not recommend having them connected to the internet in any way. IT systems are only becoming more connected, and working remotely is becoming more common. More expansive work should be taken to uncover and test the security and vulnerabilities of the different BMC implementations, as a proper mapping could lead to better security as a whole.
- The project itself has been developed using Dell servers, and has not had access to any other vendor equipment. This has naturally meant that most of the focus has been on this line of servers. Future work should take this into consideration and use a more diverse set of vendors for their servers. This will help to thoroughly test the ability of Redfish across vendors.
- Service discovery of Redfish devices is an important part of automating BMC configurations. In the cases that the vendors choose not to implement SSDP, there is a need to find other ways to discover new Redfish devices. Research into what other open source tools could be utilized for discovery is a possible future work which could aid in making BMC management simpler for diverse server environments.

8.3.2 Future work for the HPC group

- Specifically future work for the HPC group should be to map the exact current use cases for IPMI, and then translate these over to Redfish. This will let them eventually phase IPMI completely out of their environment. In addition there should be more work set to see what other functionalities Redfish has to offer that might be beneficial, such as Redfish functionality for network devices.
- The PoC presented in this thesis is just that, a Proof of Concept. It should not be used for direct implementation into the HPC environment. The HPC group should use this as a reference on the capabilities of Redfish alongside Ansible, and expand on their knowledge. The HPC group should read and understand the best practices for both server settings using Redfish as well as Ansible.

Chapter 9

Conclusion

This chapter contains the conclusion of this thesis. It presents the ending words about Redfish and IPMI, the project result, and the final words of the thesis.

9.1 Redfish & IPMI

IPMI is an aged specification which was developed with little to no consideration of security, and with very little functionality across all vendors. It is limited to the least common denominator, and more specific actions with it are made with the archaic and non-intuitive raw commands which are vendor specific. Vendors such as Dell, Intel, and HP no longer recommends using IPMI and rather recommends more modern technology such as Redfish.

The specification and its adaptations have outlived their usefulness in modern server environments. When looking at the sheer functionality that Redfish offers across vendors it becomes hard to justify not making the switch. When one considers the multitude of security issues which are easily exploited, and the fact that it is no longer developed, it no longer stays hard to justify, it becomes reckless. After all, computer systems are only as secure as the weakest link. This weakest link just happens to be embedded on the motherboard of any servers with OOB management capabilities. In the case of a successful attack on this link it would be as if the attacker had physical access to the server, and as the BMC operates at a level lower than the OS, this becomes difficult to discover and handle.

Redfish on the other hand is a more modern type of OOB management specification. It was developed with both ease of learning, security, and well known and used protocols in mind. It offers this functionality across vendors with its standardized URL API calls, making it easier to manage a diverse environment consisting of different server vendors. This is done by specifying a RESTful interface that follows OData conventions, which allows for ease of learning as it is human readable, as well as creating the possibility for clients to do or automate tasks in any programming language without knowing the details of the implementation. Generic libraries, like the one used in in this project ('redfish_utils'), can be created to help system administrators create internal or open-source tools for OOB management. Additionally the specification supports TLS v1.2 through the use of HTTPS when making API calls and transferring data. Essentially Redfish is a specification which has far greater functionality across vendors, better security, and is easier to learn. In combination with Ansible, it becomes possible to automate and orchestrate these configurations across vast environments.

9.2 Project results

The NTNU Idun HPC group wanted a PoC concerning the use of Redfish together with Ansible. They wanted to gather information about the use of Redfish and its capabilities before making any sort of switch from IPMI. They provided the bachelor group with a requirement specification containing specific tasks which they needed Redfish to be able to do. The thesis demonstrates all of the required tasks using Redfish and Ansible. In addition the thesis was expanded to include security considerations, as well as showcasing how Ansible-modules for Redfish can be developed if an OOB management use-case is not covered by any public open-source projects.

9.3 Final words

At the very start of this project, the authors of this thesis assigned themselves to watch a video presentations presented by the system architect Bruno Cornec [106]. The presentation gave a good summary of Redfish and its capabilities which helped a fresh group get started on their work. The title of the presentation is rather fitting as the final words of this thesis:

"IPMI is dead, long live Redfish!"

Bibliography

[1] insidehpc, *What is high performance computing?* Accessed on 31.01.2021, 2021. [Online]. Available: `https://insidehpc.com/hpc-basic-training/what-is-hpc/`.

[2] N. H. P. C. Group, *Idun*, Accessed on 31.01.2021, 2021. [Online]. Available: `https://www.hpc.ntnu.no/idun`.

[3] Techopedia, *Intelligent platform management interface (ipmi)*, Accessed on 31.01.2021, 2021. [Online]. Available: `https://www.techopedia.com/definition/2219/intelligent-platform-management-interface-ipmi`.

[4] Dell, *Dell - bios characterization for hpc with intel cascade lake processors*, Accessed on 27.03.2021, 2021. [Online]. Available: `https://www.dell.com/support/kbdoc/en-no/000176921/bios-characterization-for-hpc-with-intel-cascade-lake-processors`.

[5] Wikipedia, *Intelligent platform management interface (-2021)*, Accessed on 27.03.2021, 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Intelligent_Platform_Management_Interface&oldid=1011016%059`.

[6] DMTF, *Dmtf - redfish standard®*, Accessed on 31.01.2021, 2021. [Online]. Available: `https://www.dmtf.org/standards/redfish`.

[7] O. source, *What is ansible?* Accessed on 31.01.2021, 2021. [Online]. Available: `https://opensource.com/resources/what-ansible`.

[8] Openstack, *Heat*, Accessed on 19.04.2021, 2021. [Online]. Available: `https://wiki.openstack.org/wiki/Heat`.

[9] Puppet, *Puppet homepage*, Accessed on 19.04.2021, 2021. [Online]. Available: `https://puppet.com/`.

[10] Guru99, *Software configuration management in software engineering*, Accessed on 19.04.2021, 2021. [Online]. Available: `https://www.guru99.com/software-configuration-management-tutorial.html`.

[11] I. Sommerville, "*software engineering, global edition*", Accessed on 27.03.2021, 2015. [Online]. Available: `https://www.pearson.com/us/higher-education/program/Sommerville-Software-Engineering-10th-Edition/PGM35255.html`.

[12] S. Association, *Ieee 828 ieee standard for configuration management in systems and software engineering*, Accessed on 27.03.2021, 2012. [Online]. Available: https://standards.ieee.org/standard/828-2012.html.

[13] S. Norge, *Iso 10007 quality management - guidelines for configuration management*, Accessed on 27.03.2021, 2017. [Online]. Available: https://www.standard.no/no/Nettbutikk/produktkatalogen/Produktpresentasjon/?ProductID=911000.

[14] SamGu, M. Jacobs and D. Hellem, *What is infrastructure as code?* Accessed on 19.04.2021, 2021. [Online]. Available: https://docs.microsoft.com/en-us/azure/devops/learn/what-is-infrastructure-as-code.

[15] Intel, *Intel intelligent platform management interface specification second generation*, Accessed on 27.03.2021, 2013. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html.

[16] Dell, Accessed on 27.03.2021, 2021. [Online]. Available: https://www.dell.com/no-no.

[17] HP, Accessed on 27.03.2021, 2021. [Online]. Available: https://www8.hp.com/no/no/home.html.

[18] NEC, Accessed on 27.03.2021, 2021. [Online]. Available: https://www.nec.com/.

[19] J. Hargrave, *An introduction to the intelligent platform management interface*, Accessed on 27.03.2021, 2021. [Online]. Available: https://www.dell.com/downloads/global/power/ps2q04-019.pdf.

[20] W. Fischer, *Ipmi basics*, Accessed on 27.03.2021, 2021. [Online]. Available: https://www.thomas-krenn.com/en/wiki/IPMI_Basics#Intelligent_Platform_Management_Bus_.28IPMB.29.

[21] Intel, *Intelligent chassis management bus bridge specification*, Accessed on 27.03.2021, 2003. [Online]. Available: https://www.intel.com/content/www/us/en/servers/ipmi/icmb-spec-v1-rev1-3.html.

[22] PC/104, *Pci-104 specification*, Accessed on 27.03.2021, 2003. [Online]. Available: https://resources.winsystems.com/specs/PCI-104Spec_v1_0.pdf.

[23] W. Fischer, *Overview of software utilities for ipmi*, Accessed on 27.03.2021, 2021. [Online]. Available: https://www.thomas-krenn.com/en/wiki/Overview_of_Software_Utilities_for_IPMI.

[24] S. Compxtreme, *Configuring drac with ipmitool*, Accessed on 27.03.2021, 2015. [Online]. Available: https://sysadmin.compxtreme.ro/configuring-drac-with-ipmitool/.

[25] *Lenovo bmc command list*, Accessed on 03.05.2021, 2020. [Online]. Available: https://download.lenovo.com/servers_pdf/bmc_command_list_specification_v0.1.1.pdf.

[26] DMTF, *Distributed management task force, technical note*, Accessed on 27.03.2021, 2015. [Online]. Available: `https://www.dmtf.org/sites/default/files/standards/documents/Redfish%20Tech%20Note.pdf`.

[27] DMTF, *About dmtf*, Accessed on 19.04.2021, 2021. [Online]. Available: `https://www.dmtf.org/about`.

[28] H. Bruning, *Scaling ipmi to the data center: Object building blocks*, Accessed on 27.03.2021, 2013. [Online]. Available: `https://dzone.com/articles/scaling-ipmi-data-center`.

[29] json-schema-org, *Json schema homepage*, Accessed on 19.04.2021, 2021. [Online]. Available: `https://json-schema.org/`.

[30] OData, *Common schema definition language (csdl) (odata version 3.0)*, Accessed on 19.04.2021, 2021. [Online]. Available: `https://www.odata.org/documentation/odata-version-3-0/common-schema-definition-language-csdl/`.

[31] O. Ben-Kiki, C. Evans and I. d. Net, *Yaml™ specification index*, Accessed on 19.04.2021, 2009. [Online]. Available: `https://yaml.org/spec/`.

[32] DMTF, *Redfish - simple and secure management for converged, hybrid it*, Accessed on 11.03.2021, 2018. [Online]. Available: `https://www.dmtf.org/sites/default/files/Redfish_Tech_Note-November_2018.pdf`.

[33] DMTF, *Redfish - simple and secure management for converged, hybrid it*, Accessed on 26.04.2021, 2018. [Online]. Available: `https://www.dmtf.org/sites/default/files/Redfish_Tech_Note-November_2018.pdf`.

[34] restfulapi, *What is rest*, Accessed on 27.03.2021, 2021. [Online]. Available: `https://restfulapi.net/`.

[35] R. T. Fielding, *Architectural styles and the design of network-based software architectures*, Accessed on 20.04.2021, 2000. [Online]. Available: `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm#sec_5_2_1_1`.

[36] R. Hat, *What is a rest api?* Accessed on 27.03.2021, 2021. [Online]. Available: `https://www.redhat.com/en/topics/api/what-is-a-rest-api`.

[37] DMTF, *Redfish specification 1.8.0*, Accessed on 27.03.2021, 2019. [Online]. Available: `https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.8.0.pdf`.

[38] JQ, Accessed on 30.03.2021, 2021. [Online]. Available: `https://stedolan.github.io/jq/`.

[39] P. S. Foundation, *Python homepage*, Accessed on 19.04.2021, 2021. [Online]. Available: `https://www.python.org/`.

[40] Ansible, *Community general collection*, Accessed on 27.03.2021, 2021. [Online]. Available: `https://galaxy.ansible.com/community/general`.

[41] Dell, *Dell emc openmanage ansible modules*, Accessed on 27.03.2021, 2021. [Online]. Available: https://galaxy.ansible.com/community/general.

[42] Ansible, *How ansible works*, Accessed on 08.03.2021, 2021. [Online]. Available: https://www.ansible.com/overview/how-ansible-works.

[43] A. Community, *How to build your inventory*, Accessed on 10.05.2021, 2021. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html.

[44] Ansible, *Intro to playbooks*, Accessed on 27.03.2021, 2021. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html#about-playbooks.

[45] A. community, *Ansible roles documentation*, Accessed 14.05.2021, 2021. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html.

[46] Ansible, *Ansible galaxy*, Accessed on 27.03.2021, 2021. [Online]. Available: https://galaxy.ansible.com/home.

[47] A. Community, *Community general idrac_redfish_command.py*, Accessed on 04.05.2021, 2021. [Online]. Available: https://github.com/ansible-collections/community.general/blob/main/plugins/modules/remote_management/redfish/idrac_redfish_command.py.

[48] Y. Pan, *Community.general.xcc_redfish_command*, Accessed 15.05.2021, 2021. [Online]. Available: https://docs.ansible.com/ansible/latest/collections/community/general/xcc_redfish_command_module.html#ansible-collections-community-general-xcc-redfish-command-module.

[49] OpenManageAnsible, *Dell emc openmanage ansible modules*, Accessed 15.05.2021, 2021. [Online]. Available: https://galaxy.ansible.com/dellemc/openmanage.

[50] Ansible, *Encrypting content with ansible vault*, Accessed on 27.03.2021, 2021. [Online]. Available: https://docs.ansible.com/ansible/latest/user_guide/vault.html.

[51] D. Farmer, *Ipmi: Freight train to hell*, Accessed on 30.03.2021, 2013. [Online]. Available: http://fish2.com/ipmi/itrain.pdf.

[52] rapid7, *A penetration tester's guide to ipmi and bmcs*, Accessed on 30.03.2021, 2013. [Online]. Available: https://blog.rapid7.com/2013/07/02/a-penetration-testers-guide-to-ipmi/.

[53] D. Farmer, *Farmer jan 2021 blog post about ipmi*, Accessed on 12.05.2021, 2013. [Online]. Available: http://fish2.com/ipmi/.

[54] U. C.
bibinitperiod I. S. Agency, *Risks of using the intelligent platform management interface (ipmi)*, Accessed on 30.03.2021, 2013. [Online]. Available: https://blog.rapid7.com/2013/07/02/a-penetration-testers-guide-to-ipmi/.

[55] Dell, *Dell response to cve (common vulnerabilities and exposures) id's*, Accessed on 30.03.2021, 2021. [Online]. Available: https://downloads.dell.com/manuals/all-products/esuprt_software/esuprt_remote_ent_sys_mgmt/esuprt_rmte_ent_sys_rmte_access_cntrllr/integrated-dell-remote-access-cntrllr-6-for-monolithic-srvr-v1.95_faq3_en-us.pdf.

[56] Intel, *Joint message from the ipmi promoters*, Accessed on 21.04.2021, 2021. [Online]. Available: https://www.intel.com/content/www/us/en/products/docs/servers/ipmi/ipmi-home.html.

[57] P. K. B.E., *Validation of redfish - the scalable platform management standard*, Accessed on 29.03.2021, 2017. [Online]. Available: https://ttu-ir.tdl.org/bitstream/handle/2346/72726/KUMARI-THESIS-2017.pdf?sequence=1&isAllowed=y.

[58] M. L., *What is ssl inspection and how does it work? quit wondering if your server is being bogged down by requests from illegitimate sources — here's what to know about https inspection and how it works*, Accessed on 30.03.2021, 2020. [Online]. Available: https://ttu-ir.tdl.org/bitstream/handle/2346/72726/KUMARI-THESIS-2017.pdf?sequence=1&isAllowed=y.

[59] F. R. and R. J., *Hypertext transfer protocol (http/1.1): Authentication*, Accessed on 07.03.2021, 2014. [Online]. Available: https://www.ietf.org/rfc/rfc7235.txt.

[60] DMTF, *Redfish specification 1.8.0*, Accessed on 30.03.2021, 2018. [Online]. Available: https://www.dmtf.org/sites/default/files/Redfish_2018_Release_1_Overview.pdf.

[61] Dell, *Poweredge rack servers*, Accessed 15.05.2021, 2021. [Online]. Available: https://www.dell.com/en-us/work/shop/dell-poweredge-servers/sc/servers/poweredge-rack-servers.

[62] Dell, *Idrac 9 white paper*, Accessed on 26.04.2021, 2017. [Online]. Available: https://downloads.dell.com/manuals/common/dell-emc-idrac9-lc-overview.pdf.

[63] Softpanorama, *Racadm command line interface for drac on linux*, Accessed 16.05.2021, 2019. [Online]. Available: http://www.softpanorama.org/Hardware/Dell/Servers/DRAC/racadm_command_line_interface.shtml.

[64] DMTF, *Web services management*, Accessed 16.05.2021, 2021. [Online]. Available: https://www.dmtf.org/standards/ws-man.

[65] A. Community, *Best practices*, Accessed 20.05.2021, 2021. [Online]. Available: `https://docs.ansible.com/ansible/2.8/user_guide/playbooks_best_practices.html`.

[66] A. Community, *Ansible configuration settings*, Accessed 20.05.2021, 2021. [Online]. Available: `https://docs.ansible.com/ansible/latest/reference_appendices/config.html#ansible-configuration-settings-locations`.

[67] V. Systems, *All about cifs*, Accessed 20.05.2021, 2021. [Online]. Available: `https://cifs.com/`.

[68] J. Whitaker, *What is linux nfs server?* Accessed 20.05.2021, 2020. [Online]. Available: `https://cloud.netapp.com/blog/azure-anf-blg-linux-nfs-server-how-to-set-up-server-and-client`.

[69] M. W. Refnes, R. Aardalsbakk, G. Hiis-Hauge and N. Zakharov, *Bachelor bitbucket repository*, Accessed on 03.05.2021, 2021. [Online]. Available: `https://bitbucket.org/nkzk/bachelor_repo/src/master/`.

[70] M. W. Refnes, R. Aardalsbakk, G. Hiis-Hauge and N. Zakharov, *Redfish api differences across idrac versions, summary*, Accessed on 05.03.2021, 2021. [Online]. Available: `https://docs.google.com/spreadsheets/d/1eYII-d8R6Cr0W6mQA9sauDJi6iKNkcWGTifeEbaKQ9o/edit?usp=sharing`.

[71] OpenManageSDK, *Dell emc openmanage python sdk git repository*, Accessed on 04.05.2021, 2021. [Online]. Available: `https://github.com/dell/omsdk`.

[72] A. community, *Community general collection, github repository*, Accessed on 04.05.2021, 2021. [Online]. Available: `https://github.com/ansible-collections/community.general`.

[73] Git, *Download for linux and unix*, Accessed on 04.05.2021, 2021. [Online]. Available: `https://git-scm.com/download/linux`.

[74] Lenovo, *Lenovo xclarity controller rest api guide*, Accessed 20.05.2021, 2020. [Online]. Available: `https://sysmgt.lenovofiles.com/help/topic/com.lenovo.systems.management.xcc.restapi.doc/xcc_restapi_book.pdf`.

[75] H. P. Enterprise, *Hewlett packard enterprise ilo5 restful api documentation*, Accessed 20.05.2021, 2021. [Online]. Available: `https://hewlettpackard.github.io/ilo-rest-api-docs/ilo5/?shell#find-the-ilo-5-management-processor`.

[76] Dell, *Idrac9 redfish api guidefirmware version 4.20.20.20*, Accessed on 04.05.2021, 2020. [Online]. Available: `https://dl.dell.com/topicspdf/idrac9-lifecycle-controller-v4x-series_api-guide_en-us.pdf`.

[77] Dell, *Integrated dell remote access controller 9attribute registry*, Accessed on 12.05.2021, 2020. [Online]. Available: https://dl.dell.com/topicspdf/idrac9-lifecycle-controller-v4x-series_reference-guide_en-us.pdf.

[78] A. Community, *Community general redfish_info.py*, Accessed on 04.05.2021, 2021. [Online]. Available: https://github.com/ansible-collections/community.general/blob/main/plugins/modules/remote_management/redfish/redfish_info.py.

[79] A. Community, *Community general redfish_config.py*, Accessed on 04.05.2021, 2021. [Online]. Available: https://github.com/ansible-collections/community.general/blob/main/plugins/modules/remote_management/redfish/redfish_config.py.

[80] A. Community, *Community general redfish_command.py*, Accessed on 04.05.2021, 2021. [Online]. Available: https://github.com/ansible-collections/community.general/blob/main/plugins/modules/remote_management/redfish/redfish_command.py.

[81] A. Community, *Community general redfish_utils.py*, Accessed on 04.05.2021, 2021. [Online]. Available: https://github.com/ansible-collections/community.general/blob/main/plugins/module_utils/redfish_utils.py.

[82] R. H. Ansible, *Red hat ansible tower*, Accessed 15.05.2021, 2021. [Online]. Available: https://www.ansible.com/products/tower.

[83] DMTF, *Redfish scalable platforms management api specification*, Accessed on 03.05.2021, 2018. [Online]. Available: https://www.dmtf.org/sites/default/files/standards/documents/DSP0266_1.6.0.pdf.

[84] Dell, *Enabling auto-discovery*, Accessed 18.05.2021, 2021. [Online]. Available: https://www.dell.com/support/manuals/no-no/integrated-dell-remote-access-cntrllr-7-v1.50.50/idrac7ug1.50.50-v1/enabling-auto-discovery?guid=guid-1b945426-efb5-42ca-8d0f-2243d20ecb0c.

[85] A. Community, *Community general idrac_redfish_config.py*, Accessed on 04.05.2021, 2021. [Online]. Available: https://github.com/ansible-collections/community.general/blob/main/plugins/modules/remote_management/redfish/idrac_redfish_config.py.

[86] Dell, *Idrac9 security configuration guide*, Accessed on 13.05.2021, 2017. [Online]. Available: https://downloads.dell.com/manuals/all-products/esuprt_software_int/esuprt_software_ent_systems_mgmt/idrac9-lifecycle-controller-v4x-series_reference-guide_en-us.pdf.

[87] *Intel best practices fir bmc and bios security white paper*, Accessed on 03.05.2021, 2020. [Online]. Available: `https://www.intel.com/content/dam/support/us/en/documents/server-products/bmc-bios-security-bestpractices.pdf`.

[88] Supermicro, *Best practices for managing servers with ipmi features enabled in datacenters*, Accessed on 30.03.2021, 2013. [Online]. Available: `https://www.supermicro.com/products/nfo/files/IPMI/Best_Practices_BMC_Security.pdf`.

[89] H. Moore, *A penetration tester's guide to ipmi and bmcs*, Accessed on 02.05.2021, 2013. [Online]. Available: `https://www.rapid7.com/blog/post/2013/07/02/a-penetration-testers-guide-to-ipmi/`.

[90] W. Foundation, *Wireshark homepage*, Accessed on 02.05.2021, 2021. [Online]. Available: `https://www.wireshark.org/`.

[91] O. Services, *Kali linux homepage*, Accessed on 03.05.2021, 2021. [Online]. Available: `https://www.kali.org/`.

[92] O. Security, *Download kali linux images*, Accessed on 02.05.2021, 2021. [Online]. Available: `https://www.offensive-security.com/kali-linux-vm-vmware-virtualbox-image-download/`.

[93] O. Services, *Offensive security home page*, Accessed on 03.05.2021, 2021. [Online]. Available: `https://www.offensive-security.com/`.

[94] Rapid7, *Metasploit homepage*, Accessed on 02.05.2021, 2021. [Online]. Available: `https://www.metasploit.com/`.

[95] D. Farmer, *Leaky hashes in the rakp protocol*, Accessed on 02.05.2021, 2013. [Online]. Available: `http://fish2.com/ipmi/remote-pw-cracking.html`.

[96] A. Community, *Testing strategies*, Accessed on 10.05.2021, 2021. [Online]. Available: `https://docs.ansible.com/ansible/latest/reference_appendices/test_strategies.html`.

[97] A. Community, *Ansible-playbook*, Accessed on 10.05.2021, 2021. [Online]. Available: `https://docs.ansible.com/ansible/latest/cli/ansible-playbook.html`.

[98] A. Community, *Ansible lint documentation*, Accessed on 10.05.2021, 2021. [Online]. Available: `https://ansible-lint.readthedocs.io/en/latest/`.

[99] K. Morris, *Infrastructure as code, 2nd ed.* Released December 2020, Accessed on 12.05.2021, 2020. [Online]. Available: `https://www.oreilly.com/library/view/infrastructure-as-code/9781098114664/`.

[100] TravisCI, *Core concepts for beginners*, Accessed on 12.05.2021, 2021. [Online]. Available: `https://docs.travis-ci.com/user/for-beginners/`.

[101] P. V. (contribution) *et al.*, *Redfish mockup creator*, Last updated April 2021, Accessed 04.05.2021, 2016. [Online]. Available: `https://github.com/DMTF/Redfish-Mockup-Creator`.

[102] P. V. (contribution) *et al.*, *Redfish mockup server*, Last updated February 2018, Accessed 04.05.2021, 2016. [Online]. Available: `https://github.com/DMTF/Redfish-Mockup-Server`.

[103] J. Geerling, *Ci for ansible playbooks which require ansible vault protected variables*, Published Sep. 29, 2017, Accessed on 12.05.2021, 2017. [Online]. Available: `https://www.jeffgeerling.com/blog/2017/ci-ansible-playbooks-which-require-ansible-vault-protected-variables`.

[104] A. Community, *Sanity tests*, Accessed 13.05.2021, 2021. [Online]. Available: `https://docs.ansible.com/ansible/latest/dev_guide/testing_sanity.html`.

[105] Dell, *Idrac with lifecycle controller v.,2.75.75.75*, Accessed on 03.05.2021, 2020. [Online]. Available: `https://www.dell.com/support/home/en-us/drivers/driversdetails?driverid=krcxx`.

[106] T. L. Foundation, *Ipmi is dead, long live redfish cornec bruno, hpe*, Accessed 05.05.2021, 2019. [Online]. Available: `https://www.youtube.com/watch?v=nBCjuuOjxRQ`.

Appendix A

Referat

A.1 Ernst referat

Januar

21.01.21

-------------------- Preparasjon --------------------
hva er godkjent av digital signatur fra oppdragsgiver
(Signere, scanne, sende, laste ned pdf og skrive ut?)
(Holder det med å ta screenshot av signatur?)

Plan framover?

Tips til å strukturere jobbinga utover det som ble sagt i lynkurset (hvis han fulgte med på det)?

Personlig anbefaling til rapport å lese? (I tilfelle Ernst har eit anna svar, som Tom ikkje foreslo)

Personleg erfaring med redfish, og/eller Ansible?

Prosjektplan:
Problemstilling, noko spesielt viktig å tenkje på?

Tenkte kanban, sprint eller scrum

Referat - 21.01

Ernst oppgaver vs våre oppgaver

Vi driver/Eiger prosjektet (vi er prosjektledere)
Ernst er veiledere - skal stille spørsmål - sparringpartner - skal ikkje komme med svar:)

må sette opp plan og tidsfrister sjølv:(- ikkje vent på kverandre(!)

sende epost etter møte:
    stikkord:
    poenget blir en ekstrasjekk for å passe på at vi begge har forstått kverandre riktig

redfish og ansible - to overordna teknologier

formulere problemstilling og kva vi skal jobbe med
kartlegge litt kva som er gjort med redfish og ansible

justere oppgaven litt så vi har noko nytt,
prøve å ikkje berre realisere noko som allereie er gjort
    meir spennande om vi kjem med noko som tar det eit steg vidare

ta med kamera til neste møte:)

signatur ang avtale
- relativt fleksibelt
- printer ut, sender, scanner
- kopi av signatur, sendt bilde og lage ny pdf

Anbefalt rapport å lese
- Eurekaprisen i fjor - Security score card(?)
- søk på ntnuopen - apacheoppgaver evt
- Authentication of Network Components
- Måling av Informasjonssikkerhet ved hjelp av Scorecard (eureka)

Søk etter ansible (ang rapport)
- Machine goes bling, raspberry pi cluster

Bruk tid på å formulere
- enten som forskningsspørsmål eller problemstilling
- gjerne involvere oppdragsgiver

Om vi skal ha møte med både Ernst og Einar,
- ønsker Ernst veldig gjerne hans tid (torsdag 1600)

Ernst har ikkje noko imot eit møte med Einar,
- kjem til å kontakte han i Mai uansett, om hans formeininger.

innspill til planlegging av prosjekt
- påske, start av april
- ha mykje på plass til påske
- det som er gjort då er dokumentert då er tidleg start av oppgåve
- kom tidleg i gang med skriving.
- fint om vi kan sende det vi har så langt i påska til Ernst
    - kan komme med tilbakemeldinger, og har tid igjen til endringer

start skriveprosessen tidleg.

ta notater, ta notater om avgjerelser,
- frykteleg vanskeleg å ta opp igjen det man sa i februar f.eks
- pass på å ikkje gå i smellen “ja dette var jo selvfølgeleg”,
    - dokumenter alt, anta at lesaren ikkje veit noko

-------------------- Preparasjon --------------------

Møte 28?.01.21

Husk å markere (i fet eller med farge) viktigare ting ~

Høre med Ernst
- Har cisco-laben eller andre områder tilgang til BMC-utstyr

Referat 28.01.2021

problemstilling OK

risikoanalyse
- f.eks tilgang på utstyr
- **teknologien fungerer slik som forventa** (eller uforventa)

^-tenke litt på
- bruke virtuelt utstyr
- vinkling
- kan hende teknologien ikkje fungerer

kan skrive om
- (problem) med utstyr
- vinklingsendring
- skissere bredt kva oppdragsgiver ynskjer
- avgrensing (snakk med og - bytte om til server, istedenfor switch osv)

kan seinare sjå på (å ta med)
- ethernetswitch o.l.
  - ikkje nødvendigvis i prosjektplan

om tidleg ferdig (sei april)
- sjå heller på å utbreie, og å ta prosjektet vidare, f.eks teknologi o.l.

huske å sende mail til Ernst etter møte med OG i morgon
- for å bekrefte at utstyr o.l. er i orden
- avklaring

------
egne tanker

Februar

Referat 04.02.2021

Sjekker gjennom plan ila. lørdag

Har inkl grupperegler
	Eget dokument som vi dokumenterer
	Sjekk opp det
	(ha det som eit eige dokument og levere inn) (?)

prosjektplan - skal kun godkjennast

Ikkje vere redd for å stille harde krav til hardware til OG
	(om vi føler det stopper opp fordi vi ikkje har det vi trenger)

Prosjektmål
	muligens skille mellom effekt- og prosjekt-mål
	effektmål = langtid, noko og ønsker å få balanse i, effekt - noko dei ønsker på sikt
	resultatmål = resultatet vi får ila. semesteret
	kan tenke på om vi vil ha det med (VALG!! :))
er på en måte “resultatmålene”

Ang 1. prosjektmål
	kva gjer man om det ikkje fungerer i det heile tatt (åpent spørsmål) (plan B?)
		Oppdragsgiver har nok en ide om teknologien funker, men veit ikkje før man
		har prøvd
	om 1. ikkje fungerer, fungerer ikkje proof-of-concept

risikodel av prosjektplan - Innafor
	fornuftig løsning

Kommentar
	Alle kan lage generell regler
	alle tabeller og figurer burde(må :D) ha tittel (eks table/figure 1.) (5.3.1 i plan f.eks)
		introduserast og forklarast i teksten
		korleis den skal lesast og forstås

Bytte av møte til fredag
	kl 10 på fredager

Referat 04.02.2021

Husk å vere kritisk til Ernst!!!!

- bruke tida fornuftig

Spesifikasjoner

- Intel? “Opprinnelege” kilde (Spesifikt IPMI)
- Opprinneleg dokumentasjon

To ting å huske mtp referanse

- gi leser mulighet til å finne meir informasjon
- gi credit til dei som har komt opp med løsninga
- (om det er noko tilleggsinformasjon, kanskje annerleis strukturert, gjerne fleire kilder)

Seinare f.eks snakker om host-specifications kan heller referere direkte til punktet i dokumentasjonen, eller ein heilt ny kilde igjen (Redfish)

Ang grupperegler

- trenger ikkje signere noko meir på blackboard e.l.
- **huske å signere (grupperegler) i eit faktisk dokument**
  - om man legger det i prosjektplan eller som eit nytt dokument igjen er opp til oss

Referat 19.02.2021

løfte oppgave
- forskning, ta det litt vidare, “gå litt fra oppskrifta”
- kanskje vanskelig oppfordring, spes. for programmering - men, ekstra funksjonalitet?

“kva fant dykk på sjølv?”
- ta utgangspunkt i det dei ynskjer først
- skriv om utfordringer som dukker opp
- igjen, komme med noko i tillegg, løft opp oppgåva.

eksempel
- kode forbedring
- meir elegant
- meir funksjoner

detaljert spesifikasjon
- følg den, på ein kritisk måte
- ha alltid i bakhovudet, er det noko som kan forbedrast her?

om alt skal gå smertefritt
- ville ha sett på noko å legge til
  - **men handler ikkje strengt tatt om “ekstra funksjonalitet”**

passe på å få med i endeleg rapport
- korleis var det? utfordringer? var det hardt løp?
- alltid husk å addressere (spesielt vansker!)

mulig rapportforbedring:
- fortsette med iDRAC7
  - sette seg inn i iDRAC9 - endringer, moglegheiter det kunne gitt
  - (iDRAC9 er nok) meir sikkerhet
- **(om iDRAC9) kan hende syntax o.l. er annerleis.**
- **kva er annerleis, moglege forbedringer**
- Mykje blir nok oppklart når vi begynner å jobbe med stoffet, ideer kjem nok sakte men sikkert naturleg (forhåpentligvis;))

reflektere rundt
- ting man gjer undervegs
- **det å skrive en god rapport TAR TID.**

Referat 26.02.2021

Møte kl 10

ang kode
nevne i teksten og referere til github
- kode laga av Dell, iDRAC er dell
- klart i tekst kor koden kjem i fra
  - Bidragsyter :)
- Unngå å finne opp hjulet om man kan!

Tenk framover korleis man kan ta det vidare i så fall om man har funne kode.

Diskutere korleis/om vi kan ta det vidare innen neste møte med Einar.
- Forvent at det er stikker i hjulet, ikkje ta for gitt at koden vi har funne fungerer “som reklamert”

Mars

Referat 05.03.2021

Oppgava er litt enkel
- Kunne trengt å gjere noko nytt
  - Diskuter litt meir med oppdragsgiver - høre hans tanker
  - Inkludere andre einheiter - switsjer, ethernet, infiniband.
    - switch - dell har ikkje heilt på plass enda
    - infiniband - mellanox, kjøpt opp av nvidia
      - infiniband har redfish støtte
      - HCA - host channel adapter, nettverkskort

Venter akkurat no på server
- Til write-funksjoner bl.a
- HPC

Snakke med oppdragsgiver - Høre hans tanker om å utvide oppgåva
- Har kontroll på det vi har moglegheit til å gjere
- Lufte at vi nærmer oss det vi har lyst å gjere, kanskje han trenger å lufte det vidare

Kanskje prøvd å sydd alt heilt sammen før vi tar kontakt med oppdragsgiver
- Ta kontakt snarast mogleg uansett, greit å formidle kor vi står
  - Neste steg sette opp plan for kva vi kan gjere vidare.
  - Prøve å få en åpen oppgave, så vi kan sette en strek om det trengst
    - Få en utvidelse av oppgava snarast mogleg, trenger å få svar
    - så vi ikkje bruker all tida på å vente på svar, plutseleg er det påske

Referat 11.03.2021

Fekk ikkje server før onsdag

Ut av boks har ikkje støtte for Redfish
    Må vere ein viss versjon
    Unngår å gå på webinterface
Venter på svar oppdragsgiver for no

Kan prøve å ta det vidare til Dell (høre om firmware o.l.) - Høre med Dell - Før hadde dei store kunder i alle fall, har som regel en kontaktperson der osv.

Kommentar: Veit at rapporten ikkje er ferdig til påske - om større avsnitt/kapitler er uskrevet, sei ifra i dokumentet/rapporten, slik at Ernst er obs på det og kan komme med tilbakemelding på andre steder.
    Kan merke av kapitler som kladd

Førsteutkast innen 26. Mars - evt ila den helga. Ønsker helst ikkje å få 3 rapporter på skjærtorsdag!

Kan evt sende dokument på fredag, og sei det kjem en oppdatert versjon av kapittel X etterpå.

Referat 26.03.2021

// Prøver ut nytt format \\

Oss:
Har hatt nok å gjere, har hatt litt tekniske problem o.l.
Ha med utfordringer? Kodefeil?
Alternative teknologier

Ernst:
Ila søndag går bra, ser på rapporten på mandag.
Lurte på om vi har spurte OG om ekstraoppgaver.

Nei, inkl meir utfordringer som er spennande, noko som mange intuitivt tenker mange kan gjere. Fort nevne at dette er noko mange tenker man, intuitivt, gjer det på “denne” måten, men skyldes feil man har misforstått, eller skrevet feil.
Lett å drukne i detaljer, spes. med å inkludere alle config-tabber.

Alt.Tek. - I starten, bakgrunn for å inkl for forståelse for teksten seinare. Eit eller anna sted i teksten (Gjerne tidlig) nevner det for oversiktens skyld. Viser bredere kunnskap (halv-> en side)

April

Referat 09.04.2021

Rapport

Husk å finne mal på bachelor

Restrukturere innholdsoversikt
    Bakgrunn/Teori/Teknologi separert fra “oppgaven” ->gjennomgangen skal ha en naturlig kobling mot oppgavens tematikk og teknologier etc.
    Beskrivelse av teknologiene osv
    Hjelp leseren dypt inn!
    Om det blitt for langt og for mange tekniske detaljer, mister man leseren litt
    Testing (demonstrert at det funker o.l.)
        om det kun blir en side ish kan det vere på slutten av kap 4 (implemen.)
    Kan dele sikkerhet i spesifikke deler i kap 2 og kanskje summere i større kappitel lengre nede.
    KONKLUSJON SKAL IKKJE INNEHOLDE NOKO NYTT:)

[2] Referanser skal ikkje vere til nettside i rapport-teksten
    Det skal linkest ned til bibliografien.
(1.1) Unødvendig å ha med cores and GPUs. Gi leseren inntrykk av mange maskiner, ikkje mange kjerner.

IT-Department’s HPC → IT Department’s HPC

referanse 3 og 8 er generelle om IPMI. kanskje holde oss til en

Flytte kapittel 1.6 til å komme etter 1.2

SMÅ forklarande setninger for goals, og resten av rapport.
    (PKI?)

Under 1.6 Scope (Vurder å ikkje bruke iDRAC betegnelse.
Skrive generelt, og la det stå.

Unngå å dokumentere ting som “Vi ser om vi får tid, --generelt virker det rart å snakke om ting som er i framtidsform når rapporten er skreve når rapporten er ferdig.”

Referat 16.04.2021

Oss:
Ang teori-delen, flytte det til to deler? (Bryte opp av del 2)
Ang krav: Krav som blei satt av oppdragsgiver. Egne krav satt seinare.

Sette opp CI-testing i Git f.eks. Følge best practice - Ansible struktur, setter opp Vault. Må jo ha brukernavn og passord på ein eller anna plass, men må bli håndtert sikkert.
Meir naturleg i scope eller implementasjon/design?

Ang sikkerhet: Redfish er meir sikkert enn IPMI. Veldig få steder det er dokumentert korleis det er sikrare. Veldig få "korleis, eller kor mykje sikrare"-dokumentasjoner ute.

Om vi ikkje får tid, legge det til i vidare arbeid. Kva vi har planlagt o.l.
Såg på avklaring av presentasjonsdager. 7. Juni? (Ser riktig ut)
Figurer, utviklermiljø o.l. ser fornuftige ut?

Ernst:
Mange av førstepunkta, begrepa, f.eks Redfish, Ansible o.l. Når eg kjem til Proof-of-concept, så lurer eg litt på kva proof-of-concept ER, og når eg ser på rapporten deres, lurer eg på om det burde bli definert eit anna sted. Flytte til design. Bunnpunktet er å flytte det ut av teori-delen.

Sikkerhet var skrevet veldig generelt. -> Moglegheit til å plukke det opp seinare, hadde vert hyggeleg.

Husk å alltid sjå på kommentarane med kritiske blikk!

Litt forstyrrende å lese om prosjektgruppa der det blei skrevet, blei avbrutt litt. Leser om problemområdet osv, også hopper det litt med prosjektgruppa, også går rapporten tilbake til problemet. Flytt om litt.

Det er ikkje galt å skrive om krav, kanskje tydeliggjere det litt meir. Prøve å la vere å bruke begrep som iDRAC sida leseren ikkje er så kjent med det. I alle fall spesifisere leverandør i så fall.

Det at dåke utvider meir enn dei spesifikke krava. Kan legge om strukturen litt, klart tydeliggjere at det er tilleggsoppgaver dykk har lagt til.
implementasjon/design - kan gjere - men presiserer at det er ekstraarbeid som er tatt opp, kanskje legge en setning under scope, tar det litt vidare. Nevne til sensor at "hei vi har gjort litt meir enn vi har fått beskjed om" tidleg.

Supert å gjere ein vurdering av comparison, kan begynne litt tidleg på det i oppgaven. Kan sjå på det etterpå om man skal legge til i scope om man er klar over det.

Hoppa litt over figurer, sida det ikkje var introduserande tekst. Husk å ALLTID forklare figurer og evt. diagram som blir brukt. Poenget, roller, kort intro o.l.

Inkscape - for vector og bitmap o.l.

Referat 23.04.2021

Presentere-/Vise demo --- Vise kva vi jobber på

3-Technical Design - struktur, kva vi burde ha med. Trenger vi kravspesifikasjon osv?

Implementasjon - Finne ut kva vi vil ha med?

Om vi har forstått implementation og deployment

ang 5. Lurer på bruk av tid er effektiv

Oss:
i en xml fil det er en linje er attributer - ca tusen linjer
Tenkte å vise demo på tirsdag til oppdragsgiver
Kanskje viktigst å få fram i rapporten - kor mykje det er å sette seg inn i
Prøve å fange opp packets, gå litt djupare, og gjerne vise sikkerhetsfeature som redfish har.
Liste features, og liste kor/kva vi har gjort, og vår drøfting av evalueringa.
Kor ting bør bli bedre, og kor ting er ganske opp til standard.

Kva vi skal gjere med Technical Design - kravspec? Mappestruktur?
Mulighet til å lese rapporten på nytt? Etter vi har komt litt lengre --

Ernst:
no som dykk har **lagd noko sjølv**, pass på å få det skikkelig fram i rapporten. Om dykk ikkje skriver om det, er det vanskelig for sensor å vite det. Ikkje regn med at sensor sjekker og setter seg inn i git-repoet.

Litt vanskelig å sei kor man skal legge innsatsen - syns det er viktig å ha grundig sikkerhetsbit i oppgaven.
konklusjonen, best practice forsåvidt - ganske langt ned i stacken. det fundamentale i stacken, potensielt det som styrer alt, så det hørest fornuftig ut.
Må ikkje vere redd for å avsløre områder der man ikkje har fulgt best-practice. Nevne det til oppdragsgiver som vidare arbeid o.l. Viser berre at vi har kunnskap om det mangler, så det går bra.
Ha kontroll på det mest fundamentale først, kravspecen, før vi går vidare på det andre i så fall. Ha det slik som oppdragsgiver ynskjer, men dykk er 4 så det går an å ha 4 ting i parallell.

(Technical design) det viktige er kort og konsis beskrivelse av det som er gjort, har på plass litt allereie, men kontroll på det lesaren - oss som forfattere er trygge på det lesaren kan, bygd en felles grunnmur før man leser resten av oppgaven. Ha med dei aspektene som er viktige for det vi gjer. Tenke at det er naturleg å presentere korleis vi har løst design og sjølve implementasjonen.
Development environment gir meining.
Sikkert greit å ha litt om enheitene, som forklarer KVA det er. (Tabell kan holde)
Forskjell på utstyr (iDRAC 7 og 9 f.eks)
Implementasjon og design -
5 - Deployment viser en test - viser kommando, output o.l. Ser korleis det fungerer i praksis.
For å lese rapporten skikkelig må (Ernst) en ha tid, og dykk tid til å endre på.

20. Mai - siste frist. ----- Kan lese i.l.a 04. Mai - Dykk sender i.l.a Mandag natt
Husker ikkje om har sagt det før, men smart å få nokon andre til å lese. To kategorier.
Noken som har informatikk-bakgrunn -> litt forutsetning for å skjønne det.
Nokon som er flink i språk -> Til å sjå på språket i rapporten

Referat 30.04.2021

Oss:
Venter på default-config (ang Ansible)
Ila. mandags natt?
Lurer på om vi har gjort nok?
Sikkerhetsdel: Best Practice av korleis sette opp for en iDRAC - universelt -
En chapter om andre vendorer og kanskje korleis deira miljø er annerleis?
Snakka om rundt ca 400 servere, for deira skuld hadde det vert fint å ha med andre vendorer om korleis andre vendorer har tatt med iDRAC.

Ernst:
Hadde vert fint å få rapporten før 9-tida (tirsdag 04.05)
Lurer på om man har fylt opp forventningane ila. en bachelor, gjort nok jobb.
Mellom 50-80 sider er ganske vanleg. Då er man greit plassert.
Så lenger unna 50 eller så lenger unna 80 man har destod viktigare blir sjølve innhaldet.
Om man ikkje har noko å gjere, sjå på rapporten, er det noko meir å gjere, vurdering av videre arbeid, vurdering av servere eller switcher, har dykk nokon betraktninger? Føler at man er ferdig med det man fekk som hovedoppdrag.

Om man føler man har gjort en god jdobb tidlegare, kan ein utvide til andre vendorer -
Kor legger man det inn? Kjem nok mot slutten - Kapittel etterpå som tar for seg (ubestemt navn), andre vendorer. Det er nok ryddigare å ha det litt mot slutten. - Burde vere mot slutten, er jo ganske tydelig kva dykk skulle bruke.
Noter det dykk måtte ynskje med early draft o.l. så eg veit.

Mai

Referat 11.05.2021

Oss: //Ellers referert som (“ tekst “)
Såg på feedback, mykje bra, tenkte å flytte sikkerhet til etter PoC del. Deployment kjem litt etterpå som tar ting fra PoC som viser at det vi har. Har fått liste av dei mest vanlege tinga/settingsa HPC gruppa gjer.
Du skriver at høynivås-forklaring burde komme før man kjem nitty-gritty inn i forklaringane. No gjer vi på ein måte det motsatte.
Tenkte på slutten av kap 4- figur eller kapittel som forklarer kva vi har gjort og korleis det heng sammen.
Er kodesnuttene for mykje?
Figurer - LaTeX legger figurer der det er plass,
Bilder - må dei vere rett under der man snakker om det
Er det en anna mal vi tenkte å bruke?
Korleis passa sikkerheitsdelen inn?
Ka som er fornuftig å ha med i testing-kapittelet, Ansible playbooks o.l. Ansible moduler. Testing der blir ein heilt anna greie. I eit skikkelig environment hadde man hatt ordentlege tester - konkluderte med at det blir litt for omfattende for oppgaven. Ganske stor oppgave, for Ansible-moduler egentlig var ein ettertanke. Har skreve litt om korleis moduler burde ha vert med i ein slik produksjon, men kor mykje meining det betyr å ha med eit slikt kapittel om man ikkje har gjort det sjølv.

Ernst:
Mangler en høynivås forklaring av korleis ting henger sammen. Moglegheit at lavnivå kjem først. Men lesaren må tidleg skjønne, kva akkurat dykk har bidratt med. Den som leser ikkje leser mykje uten å tenke på kva dykk har gjort og kva som er “standard oppsett” osv. Tenkte at det er lettare for lesaren at høynivå kjem først så ein får ein forståelse først. Liten bit som mangler,, ein overordna oversikt over korleis alt henger sammen, korleis komponenter henger sammen. Har høynivå modell i bakhovudet, men er ikkje sikkert det er beste måte å gjere det på heller.
Ofte fint med figur, slik at det gjerne er enklare å sette seg inn i det. I kapittel 3 har vi figur som begynner med maskinene, kan ha ein versjon av denne figuren kanskje lagt til nokre piler for å vise korleis det fungerer litt nærmere, danne eit bedre bilde for den som leser. Dei kortere er fine for å eksemplisere det, men dei lengre - der må man passe på, sjå pent ut bl.a, det som strekker seg ut fleire side om en snutt av det heller hadde vert bedre, og referere til appendix, der ligger heile koden. -(“Pseudo? er det bedre?) Ja kanskje det, tenkte (s. 37-39), output av vifte bl.a, for det er tre forskjellige, pseudo, python så output? Kanskje

komt litt bedre fram, kunne vert litt kortere. Litt usikker sjølv, må spør om lesaren har interesse av å lese alt. Går ikkje gjennom koden i teksten og forklarer det som skjer. Referer alltid til figur 11 below o.l. f.eks. Introduser figuren i teksten
Bilder/Figurer - gylden regel er at figuren kjem etter teksten introduserer. Ikkje alltid det er mulig, men om man gjer det konsekvent trenger man ikkje sjå på figuren før det blir nevnt. Om latex krangler mtp referanser, at man klipper ut og limer inn seinare, for å få det der man vil.

Sjekka andre rapporter og gjenkjente ikkje heilt malen, så var litt nysjerrig på malen.
Usikker om det er formelle krav ang. mal, så burde kanskje sjekke det opp først.
Er ikkje noko problem sånt sett med malen, men pass på at det er lov.
Sikkerhetsdelen passa inn fint, tenker at dykk selger dykk sjølv litt ned, fekk ikkje tid til dette osv, prøve å vinkle det litt annerleis, at dykk gjerne har brukt meir tid her f.eks. Formulere det på ein måte som er "allmenngyldig".

Litt usikker på kapittel 3.
Trudde først at det initialt var lab environment, men så kjem det litt meir teori, som kanskje gjerne burde ha vert i kapittel 2. Side 25, står det generelt om Redfish relaterte ting, god intensjon, de generelle tingene om Redfish og Ansible burde kanskje ha flytta til kapittel 2. Kapittel 3 burde vere meir om lab environment og litt korleis ting henger sammen før man hopper rett inn i ting. Kjem først kanskje på side 24, føler at man hopper litt tilbake til kapittel 2, så kjem det litt om Ansible, så kjem det litt oppsummering av det som har komt tidlegare og føles litt unødvendig. Knytt gjerne meir opp mot figur 2 og 3 her. File storage, kor er ting plassert, moduler, Ansible Redfish (relatert til figur 2 og 3), overordna design over korleis proof-of-concept skal sjå ut. "-Intensjonen med kap 3 var å knytte sammen tingene for å vise meir korleis ting hører til"
Mista litt den tråden når eg leste det kapittelet.

Kap 2 heiter no OOB - management. Dette burde kanskje heite Background. Men etter eg leste handler det jo om OOB management så det gjer jo meining. Technologies? Det store delkapittelet hadde fortsatt vert OOB management, men ein side om software-management og testing. Ikkje stor, men skriv gjerne ein halv side-side om korleis man burde gå fram når man skal - skriv generelt, det er introduksjon for at lesaren forstår resten av oppgaven. Etter man har skreve om dette generelle, kan man knytte det opp seinare. I siste avsnittet, dette knytter opp mot management blablabla - veldig bra. Seie f.eks dette blir utvikla og blir testa etter desse rammene. Kan godt hende at testing er eit eget kapittel, eller delkapittel i PoC, litt avhengig av kor mykje det blir. "Godt å ikkje skrive om alt, og forklare at dette vi ikkje har gjort" - Etter kap 2 å bruke det man har snakka om tidlegare. Sikkerheit fungerte dog sjølv som ein egen komponent. På ein måte ein analyse av det dykk hadde gjort- best practices o.l. Reagerte ikkje at det var for mykje teori der. Det eg savna litt (fra security) var kanskje

gjerne retningslinjer - kanskje vanskeleg uten repetisjon, men en kort oppsummering av kva som er viktig om man skal implementere, Redfish, Ansible, sjølv.
Det er en PoC, det er ikkje uproblematisk at det er satt opp til alle best-practices punktene, men det er viktig å tydeliggjere at dette er viktig i eit produksjonsmiljø. Har vist at vi har fylt måla, men understreke at dykk ikkje har gjort det sida det er ein prototyp, PoC.

Kap 8 og 9 - Syns det var fint, begynte å bli seint på kveld. Tenkte det var greit, er ingen fasitsvar.
"Har ikkje noko krav om sikkerheit-" Når eg leste om expansion - Ansible modulen var noko eg ikkje fekk inntrykk av dykk hadde gjort. At kap 8.3.2 blir kortare.
("Future work inndelinga?") - Fint, tenkte litt på om det skulle stå academia, eller vidare begrep, Redfish Community o.l. Gjelder jo industrien eller alle som prøver Redfish, bedrifter o.l. som gjerne har lyst til å gjere implementasjonane.

("konklusjon? fungerer den? Prøvde å halde oss til ei side") Fint. Konklusjonen burde vere kort. Fungerte bra, var eit par småplukk men ingen grøvere!
Snakka om TLS versjon 1.2 - Den er vel på veg ut -
("Trur dokumentasjonen deira v1.2 er det laveste de supporter kanskje.")
Tenkte feil. Eg blanda 1.2 og 1.1.
("Redfish har moglegheiter for 1.0, 1.1 og 1.2, ikkje 1.3.")
Har ikkje moglegheit til å lese heile rapporten på nytt. Har litt mykje arbeid for tida. Om det er kort, f.eks testing-kap. Verste som skjer er at eg seier eg ikkje rekker det. Då får dykk gjerne ein gjennomlesing som er kortare. No fekk dykk kanskje litt meir tid, sida påskeinnleveringa var litt kortare.

"Korleis ligger det ann" - Artig oppgave, er subjektiv, syns driftsbiten er interessant, er spent på - ingen dårleg oppgave, om man klarer å få fram det dykk har produsert, om man har komt med noko nytt, kva som er satt opp, sensor er mest relevant i denne sammenheng. Har fått fram sikkerheitsbiten o.l. nå, burde ikkje vere ein bekymring sånt sett. Men kva er det ein med friske auger ser?
Kunst å få fram at man har gjort en god innsats uten å skryte eksplisitt av seg sjølv.
Kap - pass på at det er klar sammenheng mellom 1.5 lista (scope) og tilsvarande liste under 4.1 - og den er ikkje heilt lik, den er litt annerleis. Pass på at det er tydelig om det er endra. Om det er en del av opprinnelege oppgava eller ekstra jobb.
Snakkes på fredag! Før fredag kjem eg ikkje til å få lest noko nytt.

Referat 14.05.2021

Planlegge tredjepart til å lese. Begynner å bli kort med tid, dei trenger tid til å lese og vi trenger evt tid til å endre.

Trello-board, tid til endring?
Sjekk blackboard om det er formelle krav. Tidligere har det vert variert, fra møtereferat til ingenting.. Har eigentleg aldri sjekka formelle krav til kva man skal ha med. Fra sensor side tviler eg på at at h*n kjem til å pirke på det. Sjekk om det er noko som er nødvendig.

Høre med oppdragsgiver om det er greit å legge til møtereferatene
Dobbelsjekke kryssreferanser - usikker om det skal i bibliografi eller adskilt kryssreferanser.
Fint at det er section A.9 - ingen krise om det ikkje er med (I tilfelle han ønskjer at referat ikkje skal vere med)
Trur det er greit å kun spør. Meir høflighet enn mandatory signatur o.l.

Greit å ha med dine referat? Hyggeleg at du spør *nervøs latter* - det skal gå bra

Technical design -
PoC - klarare på kva demonstreringane er, har med klarare kva vi har gjort og viser til det.
Deployment - Demonstrasjon, viser eit basic oppsett

Har ikkje satt opp fleire møter - Om dykk ønskjer møte før presentasjon sei ifra
7. Juni presentasjonsdato i alle fall

Uke 22- er foreløbig ganske åpen. foreløbig tirsdag onsdag og fredag rel. åpen
4. Juni - presentasjon for oppdragsgiver
Om dykk sender melding i Teams, tag gjerne med navn så det er større sannsynlighet for at eg ser det!

A.2 Einar referat

15. Desember

Møte med Einar Ness Jensen

Slack - kanskje discord som møtemiljø

HPC - high performance computing

provisjonering av compute-noder

trondhjem - tre systemer -
    idun - som er ntnu sin lokale klynge/cluster

to nasjonale maskiner
    saga og betsy

ncna top 500 liste? av superdatamaskin-liste

bakgrunn for oppgave -
maskinene våres har compute-noder
de virkelige store systemene som betsy, så er det ikkje noke problem for den er ganske skreddersydd
det kjem fra en leverandor og har admin.verktøy som er tilpassa den maskina
fungera ganske greit for den maskina, men kan ikkje bruke det noke særlig anna sted.

har lokal klynge (idun)

komponenter av server (hardware)
er jo berre en pc, har ram pcu osv,
men en server ofte har I tillegg noko som heiter baseboard management console (bmc) (controller)
kalles for idrack, bvo (lenovo?)
kan konfigurere alt av hardware på serveren med bmc, har en egen nettverksport, er en liten cpu med os.

kan styrast (bmc) med ipmi 2.0 - problemet er at det er veldig gammalt. type 90- talet.

I dei siste åra har det komt noko som heiter RedFish-
Målet med å pitche oppgåva,
vi ønsker at nokon finner ut om redfish for oss, både teoretisk og kva det eigentleg er for noko, men også I praktiske eksempler - på korleis vi kan bruke det
tanken og håpet er å ha management på bmc modulene på alle serverene vi har, på ein litt meir automatisert måte vi har I dag.

per I dag - får en server
pakker ut og putter I racket, putter I kabler,
ideelle når vi setter inn strøm og kabler, så burde ting konfigurerast automatisk.

I dag, når vi kobler server må vi konfigurere masse ting I BIOS. Må normalt sett konfigurerast gjennom bios.
Første gang man skrur den på, må den konfigurerast. Må sette opp ip, passord osv.

Hadde vunne ganske mykje om vi hadde gjort dette automatisk, og ikkje stå manuelt med konsollen for å konfigurere. Gjer det ein del. I alle fall I løpet av dei to siste vekene har vi fått rundt 20 servere å konfigurere. Ikkje kun for HPC men også IT-drift.

Skulle gjerne møte fysisk, men korona.

Begynner 12. eller 13. Januar. Då begynner vi med lynkurs I akademisk skriving, problemstilling osv.
Sluttdato er ikkje heilt bestemt, men slutten av Mai/starten av Juni skal vi vere ferdig.

Kan veldig lite om redfish
Kan mykje om BMC og ipmi, og servere generelt.

Bruker Ansible som konfigurasjonsverktøy.

Berre send mail, latency på mail kan vere stor -
sende melding på teams, skikkelig dårleg på teams, men klarer å lese meldinger!
Kan godt sette opp Teams, sida vi alle er del I NTNU.

Antageligvis to andre som kjem til å vere involverte.
Pavlo Khmel
Anders [RETRACTED]

SSH til server.

Kan sjå vidare på litt det med switcher. Kan ta det etterkvert. Om det finnest switcher som støtter redfish.
Hovedmål er BMC på server, switcher kjem etterpå (liten bonus)
Switcher har som regel en BMC, tilsvarande som server har.

Tanken er å sette opp nokon noder, her I Trondheim, som vi kan få tilgang til BMC'en på.

Annakver veke fra 20. Januar.
Tirsdager 12.30- halvtime
Ekstramøte om det er noko spesifikt.

Januar

================== 20/1-21 11:00 Referat ===================

Brukernavn
- Gautehi
- Raymonpa
- Magnuswr
- nikitaiz

Tema
1. Møtetid, passer fortsatt annenhver tirsdag 12.30? Setter opp fra 2.februar?
2. Ble nevnt at planen var å sette opp noen noder i trondheim vi kunne få ssh-tilgang til. Ordnet det seg eller trenger du litt mer tid?
3. Prosjektavtalen, vi kommer til å sende en kopi på mail iløpet av dagen, men hvis det er noen egne krav til konfidensialitet må vi inngå en ny avtale som spesifiserer det.

Referat:
20/1-21 11:00

Opprettet gruppe i LDAP “”

Gruppe: itea_lille-bachelor
Ssh idun-login[1-3].hpc.ntnu.no
https://www.hpc.ntnu.no/idun

Tirsdag 12.30 - Sett opp på teams - så hadde Einar blitt veldig veldig glad:)
To maskiner mulig fleire ->
Muligens allereie konfigurert som dei vil ha det.
Poenget er å bruke redFish til å konfigurere for dei. (Gjerne med ansible)
(innlogget på hpc er berre for å sjå bruker-perspektiv)

Neste møte tirsdag 26. -> annakver etter det

Mål for oppgaven
I dei siste åra har det komt noko som heiter RedFish-
Målet med å pitche oppgåva,
vi ønsker at nokon finner ut om redfish for oss, både teoretisk og kva det eigentleg er for noko, men også I praktiske eksempler - på korleis vi kan bruke det

tanken og håpet er å ha management på bmc modulene på alle serverene vi har, på ein litt meir automatisert måte vi har I dag.

Minimumsinnhold er prosjektets mål, tema med avgrensning, ansvarsforhold, deltakere, ressursbehov, fremdriftsplan og prosjektavtalen.

Tilgang til testmiljø // SSH tilgang

1. 1. Feb - prosjektplan
2. 1. Feb - prosjektavtale (ja det er to separate)
3. 20. Mairs - Sluttrapport

[[[[25. Feb - malware innlevering 1]]]]]

Oppdatering på om de andre to som kom til å vere med?
(Pavlo Khmel, og Anders [REDACTED]) // ikke særlig relevant, i dunno

pakker ut og putter I racket, putter I kabler,
ideelle når vi setter inn strøm og kabler, så burde ting konfigurerast automatisk.
(hovedmål er BMC på server, switcher som kjem etterpå (liten bonus)

Liste over det som
må konfigureres på servere eller er det noe vi må finne ut av selv?

================== 26/1-21 12:30 Referat ==================

Møte 26. Januar

Husk å markere ting i fet, eller i farge

Ingen tilgang til HPC, noe vi gjør feil? ssh, skolenett, ntnu-servern.

Spurte forrige gang: Liste over det som må konfigureres på servere eller er det noe vi finner ut av selv?

Helt til slutt: signatur?
    skal få fikset det idag

maskinene var for gamle
    dell 710-servere -1.gen for gammalt
        burde vere (dell) 620, 630

Vente på svar-

idrack,

sjekke cisco-lab om det er bmc-utstyr
    Kan høre med Ernst på torsdag

prat på fredag igjen
    alt etter halv 10 går fint
        ~ møte 14:00

------------------- Til neste gang ---------------------

Clarify forklare kvifor vi gjer det vi gjer
    Kva er hensikten med oppgåva

================== 29/1-21 14:00 Referat ==================

Møte 29.01 Fredag

Clarify forklare kvifor vi gjer det vi gjer
    Kva er hensikten med oppgåva

Fått egen login-server

server
    har fått inn to med støtte for redfish atm (som vi må vente på)
    den vi har no (per 29.01) er usikker om har støtte for redfish
bachelor.hpc.ntnu.no ikke støtte for redfish med mindre idrac minst v7, finn ut av det selv.
    installer hva dere vil, ipmi tools for å kommunisere med switchen.

google drive
    gitt epost og fått tilgang til egen drive

diagram
    redfish - to nederste
    dell switch, 4 porter

em1
    burde ha idrac 7
    er en r720

direkteport fra hpc til switch
    konfigurere sjølv og finne ut av ting
        vente på svar - må koble til først

passord på begge r720 (er gitt)
    skal vere samme som vi allereie har fått (ligger i teams)

signert prosjektavtale
    ligger i teams .docx fila

IPMI
    Kan snakke IPMI lokalt med loginboksen
    interessert i redfish for å ERSTATTE ipmi
    trenger “ipmi tools” for å begynne

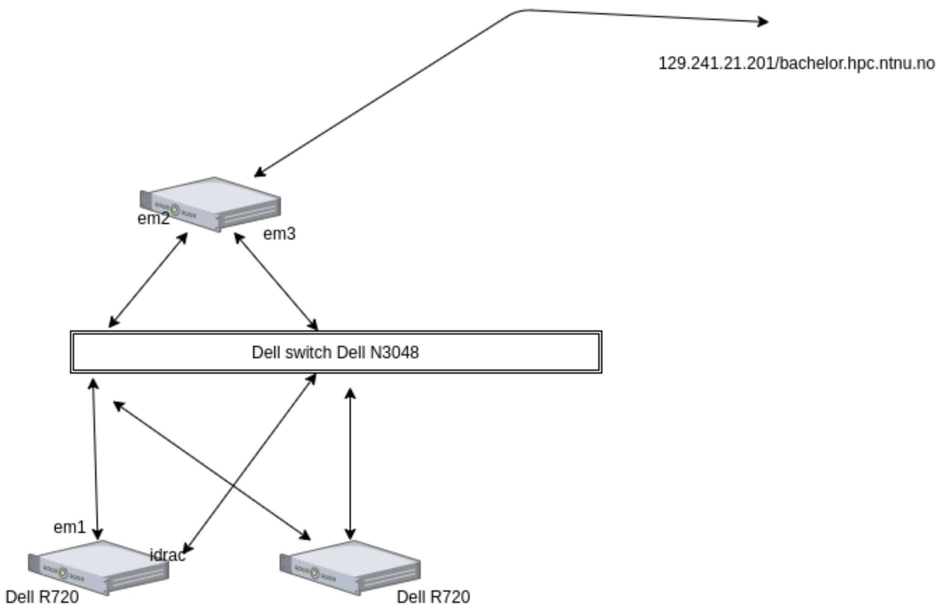
ikkje endre nøkler:)
    burde dokumentere nøklene i tilfelle situasjon oppstår
neste møte tirsdag 02.02 - 12:30

Februar

================== 02/2-21 12:30 Referat ==================

Møte 02.02 Tirsdag

-



-

Nytt dokument fra Einar, ligger i delt mappe, inneholder bilder av iDrac, og instruks til ssh port forwa   rding

    skal vere redfish compliant

skal muligens få opptil 1-2 servere til.

loginserver kobla opp med seriekabel/port til switchen

    em1-sjekk sjølv

    bruker til switch (?)

user: calvin

bruke screen på seriedevicen

    screen /dev/<console_port> 115200
    baudrate 9600 115200

to veker fra i dag, 16.02 - neste møte

================== 16/2-21 12:30 Referat ==================

Møte 16.02 Tirsdag

Request:
Flytte møtet til klokka 14:00 framover

Kravspec:
    Ikkje meir enn opprinnelige oppgaveteksten per no
    Bruke redfish
    Heilt blank server som kjem inn
    Skal kunne bruke redfish til å konfigurere idrack(eller kva leverandøren kaller det)
    Visst det er tid -> Bruke Ansible til å automatisere
    Hoved: Bruke redfish til å konfigurere BMC

Får ny server: prosedyre:
    Liten smørbrødliste over ting som skal endrast
    Manuell prosess
    Funker på Idun-kluster, som er lokal
    gjennomsnitt 2 servere i måneden i klusteret
        tar fort en times tid ++
        om ny modell kjem inn, kanskje litt annerleis, ta punkt til det

interessant for resten av IT-avdelingen
har 100 noko maskiner
for det første
    litt usikker kva configen faktisk er på ulike noder

hadde vert interessant: config/inventory management
full oversikt over servere og faktisk sjå config: kan ikkje garantere at alle er like, så hadde vert interessant å sjå individuell forskjell på configs.

Oppdatering av system:
    Semi-manuelt
    HPC har nokre verktøy for å oppdate (DSU)
    Har også brukt iDRAC direkte

om mogleg på redfish:
    mogleg å oppdatere direkte med redfish?

18. Februar 15:00-16:00
    melding fra dell, presentasjon av nye idrac versjon 9. (Vi jobber med idrac v.7)
        webinar, kan vere interessant
        einar skal videresende til oss

Skal få sendt en ny kravspec, litt meir detalj, men er stort sett det samme som før.

Test ut idrac 7 først (det vi har), sei ifra om det mangler redfish funksjonaliteter

Dei har allereie eit ansible oppsett av OS på maskiner (fra før av)

Kun OS - Gjelder ikkje idrac!

Skal sende gitrepo om mogleg, men ikkje ta det som eit sjølvfølge

Sjekk idrac på nodene vi har tilgjengelege (om idrac7 er godt nok, kan evt få nyere maskiner med idrac9 fra produksjonsmiljøet)

Mars

================== 02/3-21 14:00 Referat ===================

Møte 02.03 Tirsdag

Spm:

Clarification på storage

/ *** \

Med Storage, meinte Einar lagring av konfigurasjonen.

Ansible config til NTNU -

Ang en server til

Kun tilgang til en atm

Burde vere tilgang til 2, men kan hende har glømt å gi beskjed (lik iDRAC, så 2.65-)

Skal sjekke opp, får tilbakemelding når den andre serveren er satt opp

Var i grunn 2.40.40.40 vi fekk tildelt

Med oppgradering fekk vi 2.65.65.65

Med iDRAC9 kan vi få firmware updates, og wipe config med redfish.

Skal vere mogleg å få en iDRAC9 for nokon dager.

Ang oppdatering skal det berre vere å logge på dell og få det man trenger.

Lisens er knytta til funksjonalitet på iDRACn (i alle fall hos Dell)

================= 16/3-21 14:00 Referat ==================

Møte 16.03 Tirsdag

Likte Redfish og webløsning mest
Skal eigentleg bruke Redfish (interface) til å connecte, istedenfor CLI
    Brukest hovedsakleg til å fikse IPMI - (kommandolinje)

Ang Rapporten (Technical Report)
    Engelsk eller norsk går fint
    Så lenge engelsken er forståelig!
    (Engelsk)

Visst du enabla DHCP -måten å få tilbake
sette opp en DHCP server på den interfacen som er kobla til samme nettet. (På loginserveren vårest)
    (Kan sjå på sjølv eller vente på at Einar fikser) Installere en DHCP server på jumpnoden vi bruker - interface på samme iDRAC - då vil iDRACn få ein ip adresse (om satt opp med DHCP) og IPn vil visest

(DHCP er enabla som default når dei får inn iDRACsa nye.)
    interessant å sjå kva som skjer ved å sette DHCP og resette

EM3 - Kan sette opp DHCP server på

Regner med å få den andre serveren i relativt nær framtid. Blir samme modell som dei andre.

================== 23/3-21 14:00 Referat ==================

Møte 23.03 Tirsdag

For copy

Oss som snakker:
Einar som snakker:

Oss som snakker:

Har rota til nokon credentials - lurer på om det er enkel workaround.
192.168.0.120
.122 - finner ingenting særlig
.124 Student/R
DHCP config fil - har kun satt range
.123 blei det importert en config. Skulle importere en SCP, hadde ikkje eksportert ip'en, så den hadde fått .122 av config-filen.
Har ikkje bytta passord, i config er det default C- passord.
Har ikkje prøvd Student/R på .120 → Var ikkje riktig
Har sett at Dell kan levere default brukernavn/passord på iDRAC - lurer på om credentials har blitt resatt til dette.

Einar som snakker:

Var .122 den vi snakka om sist gang?
Og .120 - default
De to med iDRAC9 som sliter?
.121 - Der finnest noko - Root / C
.122 - finnest ingenting
.123 - Var noko der før, ingenting no.
kan hende .123 har fått ein default ip-adresse.
Har ikkje bytta passord?
Til og begynne med hadde vi eit anna passord på BMC, kan det stemme?
Interessant, må reise bort å sjå på konsollet.
Kan vere når ein resetter iDRAC at den får default credentials.
Skal reise bort å sjå på det fysisk, skal seie ifra etter eg har sett på det.

April

================== 13/4-21 14:00 Referat ==================

Møte 13.04 Tirsdag

Samtaletema:
Briefing over kva vi har gjort
Forventninga/Ønskjer til rapportstruktur
    Kor generell eller detaljert

nick / evt oss
    kor mykje teori om redfish og ansible (o.l.) skal vi ha med
    om det strengt tatt er implementasjon
    Fikser opp litt i playbooks. Roller for andre leverandører bl.a.
    Skal fikse draft en draft før innleveringsfrist.
    1-2-3 og en med statisk. Alle har samme IP som før.

einar
    ta med det dykk kan.
    ta med så mykje om redfish som mogleg - ansible er ikkje prioritert, trenger ikkje i grunn. Executive Summary om Ansible, litt meir detaljert om Redfish.
    Mogleg å få ein draft av rapporten? (som skal til Einar, ikkje bachelor)

    //Tilleggs-interesse-spørsmål om nettverkskort
    kombinert BMC og NIC node.
    tillegg skal ha 2x10gb nettverkskort på compute-nodene
        (Var ikkje tidlegare, var feilbestilling)
    1gb for BMC og provisjoneringssystem
    10gb og 25gb for infiniband, o.l.

================= 27/4-21 14:00 Referat =================

Møte 27.04 Tirsdag

Antall servere / CPU osv
Sikkerhet (nmap, wireshark)
rutiner rundt ipmi
server hardening?
> Gjennomgang av demo

Oss:
Fekk tilbakemelding på første draft, fekk antall på cores o.l.
Har dykk eit “ish” antall servere i HPC miljøet?

Sikkerhetsdelen - går det fint å laste ned verktøy som nmap + wireshark?

Rutiner - fekk nye servere og satt pop, mange funksjonar dykke måtte manuelt inn og disable?
Med redfish er det mykje fleire ting som er legacy så å sei.

Sett litt på dynamisk inventory, men blir litt avansert
kan bruke nmap for å discovere - men må fortsatt holde oversikt på en eller anna måte i en database eller liknande. korleis hold dykk styring på server?

Noken tanker?

Einar:
For NTNU sin del har vi rundt 100 servere for HPC-miljøet.
For heile NTNU er det vel ca 600.
På nasjonalt nivå ligger vi oppe i tre eller fire tusen.
HPC styrer også nasjonalt, og det er oppe i fire tusen.

Laste ned går fint, men **nervøs latter**
Visst du holder deg til internnettverket til nodene (nettverk/IDUN) er det ok, visst dåke begynner å scanne eksternt nettverk, vil det bli plukka opp.

Må heller inn for å *enable* funksjoner ved oppsett - helst det vi prøver å unngå.

På idun bruker vi noko som heiter xcat - er eit slags deployment system
om ein maskin booter på nettverket, vil den plukke opp det, vil då etter korleis vi har konfigurert xcat boote noe som heiter genesis-image og prøve å sette BMC-nettverket riktig for å få riktig ip-addresse, vlan, brukernavn og passord.

ip er definert i xcat - kva som er riktig ip.
har xcat og 4man. forskjellige måter å gjere det på.

Ønskjer å holde ein presentasjon, litt lengre for meg og mine kollegaer - på eit eller anna tidspunkt som passer dykk.
kva dykk har gjort,
korleis det fungerer,
kva dykk trur vi må gjer for å implementere og
kva er forutsetningen for at det skal funke, korleis ein BMC må vere konfigurert fra en leverandør for å gjere disse tingene. når vi kjøper server, har vi en mulighet til å få den ferdigkonfigurert som vi vil ha den i iDRACn, så kva er minimum-konfigurasjonen vi må ha for å kjøre desse redfish-configene uten å vere nær konsollene på den.

//Blei en del diskusjon dette møtet, så videre er kursiv oss, og vanleg refererer til Einar
oss: idrac9, redfish er enabled by default- dell prøver å bevege seg vekk fra å bruke ipmi, dell hp osv har komt ut med statement at ipmi ikkje kjem til å bli vidareutvikla og nevner då spesifikt Redfish som ein av dei standardane. Ein av konfigurasjonen eg har sett på med iDRAC9 trur eg redfish skal komme pre-enabled.

Vet dere om seriekonsoll?
En av de viktigste tingene, med HPC, er at de har seriekonsoll. Det får vi via IPMI. Lurer på om Redfish har den samme moglegheiten.

Redfish er Restful API som blir brukt med Curl kommandoer, så man har ikkje teknisk sett en terminal for det. - Kobling til BMC, så skal man ha alle funksjonaliteter med Redfish. Den standarden er ganske lik fra vendor til vendor, har kanskje nokon iDRAC-settings som er unike - hovedsakelig alle vendorer som støtter Redfish.

Henger litt opp i at IPMI ikkje kjem til å bli vidareutvikla
IPMI sin standard kjem ikkje til å bli vidareutvikla.
Serial over LAN alternativ?
Vil veldig gjerne å få ein full demo- for meg og mine kollegaer
Kunne tenkt at dykk simulerer en server er blank.
Få den riktig konfigurert med settings o.l.
Kan vi anta at dei serverane vi har, er riktig default settings? eller noko meir dykk spesifikt setter?
Kan finne på nokre ting, kan sette på f.eks
    Skru av hyperthreading-
    single root IO virtualization (SR-IOV) - kan ønske å skru på

kan sette opp fra default - einar har du brukt server_config_profile før? Kan spesifisere kun dei settingsa man vil ha forandra også -

Har sett på det ein gang i tida, men var litt for mykje på den tida.
Kanskje det er Dell-spesifikt også, skal sjå på det
På NTNU er det omtrent kun Dell, vi har noko Lenovo. Poenget er at vi inngår årlige avtaler, 2 år sida var det Lenovo f.eks.

Dell endra på kommandoer så lenge man har fila, men om man har andre settings at det må bli gjort en etter en.
På nasjonale maskiner har vi Dell, HP, Lenovo, (trur) Gigabyte-servere også
Slutten av Mai, begynnelsen av Juni?

Appendix B

Code

Code listing B.1: Results for server health Check - 4.5.1

```
{
    "changed": false,
    "failed": false,
    "redfish_facts": {
        "health_report": {
            "entries": [
                [
                    {
                        "system_uri": "/redfish/v1/Systems/System.Embedded.1"
                    },
                    {
                        "EthernetInterfaces": [
                            {
                                "Status": {
                                    "Health": "OK",
                                    "State": "Enabled"
                                },
                                "ethernetinterface_uri": "/redfish/v1/Systems/
                                    System.Embedded.1/EthernetInterfaces/NIC.
                                    Embedded.1-1-1"
                            }
                        ],
                        "Memory": [
                            {
                                "Status": {
                                    "Health": "OK",
                                    "State": "Enabled"
                                },
                                "memory_uri": "/redfish/v1/Systems/System.Embedded
                                    .1/Memory/DIMM.Socket.B5"
                            },
                            {
                                "Status": {
                                    "Health": "OK",
                                    "State": "Enabled"
                                },
                                "memory_uri": "/redfish/v1/Systems/System.Embedded
                                    .1/Memory/DIMM.Socket.A5"
                            },
                            {
```

```
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.A1"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.B4"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.B2"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.A4"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.A3"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.B6"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.B3"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.A6"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.A2"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "memory_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Memory/DIMM.Socket.B1"
            }
        ],
        "NetworkDeviceFunctions": [
            {
                "Status": {
                    "Health": "OK",
                    "HealthRollup": "OK",
                    "State": "Enabled"
                },
                "networkdevicefunction_uri": "/redfish/v1/Chassis/
                    System.Embedded.1/NetworkAdapters/NIC.Embedded
                    .1/NetworkDeviceFunctions/NIC.Embedded.1-1-1"
            },
            {
                "Status": {
                    "Health": "OK",
                    "HealthRollup": "OK",
                    "State": "Enabled"
                },
                "networkdevicefunction_uri": "/redfish/v1/Chassis/
                    System.Embedded.1/NetworkAdapters/InfiniBand.
                    Slot.4/NetworkDeviceFunctions/InfiniBand.Slot
                    .4-1"
            }
        ],
        "NetworkPorts": [
            {
                "Status": {
                    "Health": "OK",
                    "HealthRollup": "OK",
                    "State": "Enabled"
                },
                "networkport_uri": "/redfish/v1/Chassis/System.
                    Embedded.1/NetworkAdapters/NIC.Embedded.1/
                    NetworkPorts/NIC.Embedded.1-1"
            },
            {
                "Status": {
                    "Health": "OK",
                    "HealthRollup": "OK",
                    "State": "Enabled"
                },
                "networkport_uri": "/redfish/v1/Chassis/System.
                    Embedded.1/NetworkAdapters/InfiniBand.Slot.4/
                    NetworkPorts/InfiniBand.Slot.4-1"
            }
        ],
        "Processors": [
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "processor_uri": "/redfish/v1/Systems/System.
                    Embedded.1/Processors/CPU.Socket.2"
            },
            {
                "Status": {
                    "Health": "OK",
                    "State": "Enabled"
                },
                "processor_uri": "/redfish/v1/Systems/System.
                    Embedded.1/Processors/CPU.Socket.1"
            }
        ],
        "SimpleStorage": [
            {
                "Status": {
                    "Health": null,
                    "HealthRollup": null,
                    "State": "Enabled"
                },
                "simplestorage_uri": "/redfish/v1/Systems/System.
                    Embedded.1/SimpleStorage/AHCI.Embedded.2-1"
            },
            {
                "Status": {
                    "Health": null,
                    "HealthRollup": null,
                    "State": "Enabled"
                },
                "simplestorage_uri": "/redfish/v1/Systems/System.
                    Embedded.1/SimpleStorage/AHCI.Embedded.1-1"
            }
        ],
        "Storage": [
            {
                "Status": {
                    "Health": null,
                    "HealthRollup": null,
                    "State": "Enabled"
                },
                "storage_uri": "/redfish/v1/Systems/System.Embedded
                    .1/Storage/AHCI.Embedded.2-1"
            },
            {
                "Status": {
                    "Health": null,
                    "HealthRollup": null,
                    "State": "Enabled"
                },
                                        "storage_uri": "/redfish/v1/Systems/System.Embedded
                                            .1/Storage/AHCI.Embedded.1-1"
                                    }
                                ],
                                "System": {
                                    "Status": {
                                        "Health": "Critical",
                                        "HealthRollup": "Critical",
                                        "State": "Enabled"
                                    }
                                }
                            }
                        ]
                    ],
                    "ret": true
                }
            }
        } {
            "changed": false,
            "failed": false,
            "redfish_facts": {
                "health_report": {
                    "entries": [
                        [
                            {
                                "chassis_uri": "/redfish/v1/Chassis/System.Embedded.1"
                            },
                            {
                                "Chassis": {
                                    "Status": {
                                        "Health": "Critical",
                                        "HealthRollup": "Critical",
                                        "State": "Enabled"
                                    }
                                },
                                "Fans": [
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "State": "Enabled"
                                        },
                                        "fan_uri": "/redfish/v1/Chassis/System.Embedded.1/
                                            Thermal#/Fans/0"
                                    },
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "State": "Enabled"
                                        },
                                        "fan_uri": "/redfish/v1/Chassis/System.Embedded.1/
                                            Thermal#/Fans/0"
                                    },
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "State": "Enabled"
                                        },
                                        "fan_uri": "/redfish/v1/Chassis/System.Embedded.1/
                                            Thermal#/Fans/1"
                                    },
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "State": "Enabled"
                                        },
                                        "fan_uri": "/redfish/v1/Chassis/System.Embedded.1/
                                             Thermal#/Fans/1"
                                    },
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "State": "Enabled"
                                        },
                                        "fan_uri": "/redfish/v1/Chassis/System.Embedded.1/
                                             Thermal#/Fans/2"
                                    },
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "State": "Enabled"
                                        },
                                        "fan_uri": "/redfish/v1/Chassis/System.Embedded.1/
                                             Thermal#/Fans/2"
                                    },
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "State": "Enabled"
                                        },
                                        "fan_uri": "/redfish/v1/Chassis/System.Embedded.1/
                                             Thermal#/Fans/3"
                                    },
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "State": "Enabled"
                                        },
                                        "fan_uri": "/redfish/v1/Chassis/System.Embedded.1/
                                             Thermal#/Fans/3"
                                    }
                                ],
                                "PCIeDevices": [
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "HealthRollup": "OK",
                                            "State": "Enabled"
                                        },
                                        "pciedevice_uri": "/redfish/v1/Systems/System.
                                             Embedded.1/PCIeDevices/0-31"
                                    },
                                    {
                                        "Status": {
                                            "Health": "OK",
                                            "HealthRollup": "OK",
                                            "State": "Enabled"
                                        },
                                        "pciedevice_uri": "/redfish/v1/Systems/System.
                                             Embedded.1/PCIeDevices/0-28"
                                    },
                        {
                            "Status": {
                                "Health": "OK",
                                "HealthRollup": "OK",
                                "State": "Enabled"
                            },
                            "pciedevice_uri": "/redfish/v1/Systems/System.
                                Embedded.1/PCIeDevices/0-23"
                        },
                        {
                            "Status": {
                                "Health": "OK",
                                "HealthRollup": "OK",
                                "State": "Enabled"
                            },
                            "pciedevice_uri": "/redfish/v1/Systems/System.
                                Embedded.1/PCIeDevices/94-0"
                        },
                        {
                            "Status": {
                                "Health": "OK",
                                "HealthRollup": "OK",
                                "State": "Enabled"
                            },
                            "pciedevice_uri": "/redfish/v1/Systems/System.
                                Embedded.1/PCIeDevices/3-0"
                        },
                        {
                            "Status": {
                                "Health": "OK",
                                "HealthRollup": "OK",
                                "State": "Enabled"
                            },
                            "pciedevice_uri": "/redfish/v1/Systems/System.
                                Embedded.1/PCIeDevices/0-0"
                        },
                        {
                            "Status": {
                                "Health": "OK",
                                "HealthRollup": "OK",
                                "State": "Enabled"
                            },
                            "pciedevice_uri": "/redfish/v1/Systems/System.
                                Embedded.1/PCIeDevices/4-0"
                        },
                        {
                            "Status": {
                                "Health": "OK",
                                "HealthRollup": "OK",
                                "State": "Enabled"
                            },
                            "pciedevice_uri": "/redfish/v1/Systems/System.
                                Embedded.1/PCIeDevices/0-17"
                        }
                    ],
                    "PowerSupplies": [
                        {
                            "Status": {
                                "Health": "Critical",
                                "State": "Disabled"
```

```
                                },
                                "powersupply_uri": "/redfish/v1/Chassis/System.
                                    Embedded.1/Power#/PowerSupplies/0"
                            },
                            {
                                "Status": {
                                    "Health": "OK",
                                    "State": "Enabled"
                                },
                                "powersupply_uri": "/redfish/v1/Chassis/System.
                                    Embedded.1/Power#/PowerSupplies/1"
                            }
                        ]
                    }
                ]
            ],
            "ret": true
        }
    }
} {
    "changed": false,
    "failed": false,
    "redfish_facts": {
        "health_report": {
            "entries": [
                [
                    {
                        "manager_uri": "/redfish/v1/Managers/iDRAC.Embedded.1"
                    },
                    {
                        "Manager": {
                            "Status": {
                                "Health": "OK",
                                "State": "Enabled"
                            }
                        }
                    }
                ]
            ],
            "ret": true
        }
    }
}
```

Code listing B.2: Import SCP Profile Preview - Referenced in -

```
#!/usr/bin/python
# Copyright: (c) 2020,
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl
     -3.0.txt)
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

DOCUMENTATION = r'''
---
module: import_idrac_scp_preview

short_description: Import a given idrac server configuration profiles without
     making any changes to target host (preview).

version_added: -

description: This module previews an idrac server configuration profile from a
     network share on host(s) by utilizing Redfish API, specifically the OEM action
     EID_674_Manager.ImportSystemConfigurationPreview. It returns an "Accepted" or "
     Error" message. This can be used to test import of an SCP on multiple hosts
     safely without making any changes. Currently only supports network share type
     NFS.

options:
    idrac_ip:
        description: This is the message to send to the test module.
        required: true
        type: str
    idrac_user:
    idrac_password:
    share_type:
    share_name:
    scp_file:
    scp_components:


author:
    - Your Name (@yourGitHubHandle)
'''

EXAMPLES = r'''
# Pass in a message
- name: Test with a message
  my_namespace.my_collection.my_test_info:
    name: hello world
'''

RETURN = r'''
# These are examples of possible return values, and in general should use other
     names for return values.
original_message:
    description: The original name param that was passed in.
    type: str
    returned: always
    sample: 'hello world'
message:
    description: The output message that the test module generates.
    type: str
    returned: always
    sample: 'goodbye'
my_useful_info:
    description: The dictionary containing information about your system.
    type: dict
    returned: always
    sample: {
        'foo': 'bar',
        'answer': 42,
    }
'''

from ansible.module_utils.basic import AnsibleModule
from ansible.module_utils.urls import open_url
from ansible.module_utils.six.moves.urllib.error import URLError, HTTPError
from ansible.module_utils._text import to_native
import json
from ansible_collections.community.general.plugins.module_utils.redfish_utils
     import RedfishUtils



class ExtendedRedfishUtils(RedfishUtils):
    def import_config_preview(self):
        result = {}
        action_uri = "Actions/Oem/EID_674_Manager.ImportSystemConfigurationPreview"
        full_uri = self.root_uri + self.resource + action_uri
        postheader = {'content-type': 'application/json'}

        payload = {
            'ShareParameters':
            {
            'ShareType': module.params['share_type'],
            'ShareName': module.params['share_name'],
            'IPAddress': module.params['share_ip'],
            'FileName': module.params['scp_file'],
            'Target': module.params['scp_components']
            }

        }
        response = self.post_request(full_uri, payload)

        if response['ret'] is False:
            return response

        result['ret'] = True
        response_output = response['resp'].__dict__
        headers = response_output['headers']
        return {'ret': True, 'msg': headers }


def run_module():
    # define available arguments/parameters a user can pass to the module
    module_args = dict(
        name=dict(type='str', required=True),
    )

    # seed the result dict in the object
    # we primarily care about changed and state
    # changed is if this module effectively modified the target
    # state will include any data that you want your module to pass back
    # for consumption, for example, in a subsequent task
    result = dict(
        ret=False,
        failed=False,
        error=False,
        msg= {},
    )

    # the AnsibleModule object will be our abstraction working with Ansible
    # this includes instantiation, a couple of common attr would be the
    # args/params passed to the execution, as well as if the module
    # supports check mode
    module = AnsibleModule(
        argument_spec = dict(

            #idrac credentials
            idrac_ip = dict(required=True, type='str'),
            idrac_user = dict(required=True, type='str'),
            idrac_password = dict(required=True, type='str', no_log=True),
            # network share
            share_ip = dict(required=True, type='str'),
            share_name = dict(required=True, type='str'),
            share_type = dict(required=True, type='str'),

            scp_file = dict(required=True, type='str'),
            scp_components = dict(required=False, choices=['ALL', 'IDRAC', 'BIOS',
                'NIC', 'RAID'], default='ALL'),
            resource_id = dict(required=True, type='str'),
        ),
        supports_check_mode=False)


    # Resource id TODO: ADD TO MODULE.PARAMS
    resource_id = module.params['resource_id']

    # credentials
    creds = {'user': module.params['username'],
             'pswd': module.params['password']}
    # Build URL
    root_uri = "https://" + module.params['idrac_ip'] + "/Managers/"

    # resource_id = idrac.embedded.1
    rf_utils = ExtendedRedfishUtils(creds, root_uri, timeout,module, resource_id=
        resource_id)


    # in the event of a successful module execution, you will want to
    # simple AnsibleModule.exit_json(), passing the key/value results
    module.exit_json(**result)


def main():
    run_module()


if __name__ == '__main__':
    main()
```

Code listing B.3: health_check.py - Referenced in 4.4.4 - 4.5 - 4.5.1

```
#!/usr/bin/python
import requests
import json
import urllib3
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)

ip = "192.168.0.123"
basic_uri = "https://" + ip
username = "root"
password = "redfish"

category = "Chassis" # Systems | Chassis | Managers

VERBOSE = True

# Functions to avoid code duplication
def verbose(text):
    if VERBOSE:
        print(text)

def console_msg_json(text1="", text2="", text3=""):
    if text1:
        print(json.dumps(text1, indent=4))
    if text2:
        print(json.dumps(text2, indent=4))
    if text3:
        print(json.dumps(text3, indent=4))


def GET(url):
    response = requests.get(url, auth=(username, password), verify=False)
    return response.text


def get_instance_status(instances):
    for member in instances["Members"]:
        member_url = basic_uri + member["@odata.id"]
        element = json.loads(GET(member_url))
        console_msg_json(element["@odata.id"], element["Status"])


# Program

console_msg("Health report for host: " + ip)

if category == "Systems":
    verbose("\n\n***********Systems***************\n\n")
    URL = basic_uri + "/redfish/v1/" + category + "/"
    response = json.loads(GET(URL))
    members = response["Members"]
    for member in members:
        full_category_url = basic_uri + member["@odata.id"]
        instance = json.loads(GET(full_category_url))


        # Memory:
        verbose("Memory status summary:")
        console_msg_json(instance["MemorySummary"])
        memory_resource_url = basic_uri + instance["Memory"]["@odata.id"]
        memory_instances = json.loads(GET(memory_resource_url))
        get_instance_status(memory_instances)


        # Processor
        verbose("Processor status summary: ")
        console_msg_json(instance["ProcessorSummary"])
        processor_resource_url = basic_uri + instance["Processors"]["@odata.id"]
        processor_instances = json.loads(GET(processor_resource_url))
        get_instance_status(processor_instances)


        # Network interfaces
        verbose("Network interfaces: ")
        network_interfaces_url = basic_uri + instance["NetworkInterfaces"]["@odata.
             id"]
        network_interfaces_instances = json.loads(GET(network_interfaces_url))
        get_instance_status(network_interfaces_instances)


        # PCIeDevices
        verbose("PCIeDevices: ")
        for device in instance["PCIeDevices"]:
            PCIeDevices_url = basic_uri + device["@odata.id"]
            PCIeDevices_instances = json.loads(GET(PCIeDevices_url))
            console_msg_json(PCIeDevices_instances["@odata.id"], PCIeDevices_
                 instances["Status"])


        # Storage
        verbose("Storage: ")
        storage_url = basic_uri + instance["Storage"]["@odata.id"]
        storage_instances = json.loads(GET(storage_url))
        get_instance_status(storage_instances)




if category == "Managers":
    verbose("\n\n***********Managers***************\n\n")
    full_category_url = basic_uri + "/redfish/v1/" + category + "/"
    response = json.loads(GET(full_category_url))
    members = response["Members"]
    for member in members:
        manager_url = basic_uri + member["@odata.id"]
        manager_instance = json.loads(GET(manager_url))
        print("@odata.id:", member["@odata.id"])
        console_msg_json(manager_instance["Status"])


if category == "Chassis":
    verbose("\n\n***********Chassis***************\n\n")
    URL = basic_uri + "/redfish/v1/" + category + "/"
    response = json.loads(GET(URL))
    members = response["Members"]
    for member in members:
        full_category_url = basic_uri + member["@odata.id"]
        instance = json.loads(GET(full_category_url))
        verbose("Chassis health summary: ")
        console_msg_json(instance["Status"])


        # Power
        verbose("Power supplies: ")
        power_url = basic_uri + instance["Power"]["@odata.id"]
        power = json.loads(GET(power_url))
        for powersupply in power["PowerSupplies"]:
            console_msg_json(powersupply["@odata.id"], powersupply["Status"])


        # Thermal
        verbose("Thermal: ")
        thermal_url = basic_uri + instance["Thermal"]["@odata.id"]
        thermal = json.loads(GET(thermal_url))
        # Thermal - Fans
        for fan in thermal["Fans"]:
            console_msg_json(fan["@odata.id"], fan["Status"])


        # Thermal - Temperatures
        for temp in thermal["Temperatures"]:
            console_msg_json(temp["@odata.id"], temp["Status"])


        # Network adapters
        verbose("Network adapters: ")
        network_adapters_url = basic_uri + instance["NetworkAdapters"]["@odata.id"]
        network_adapters_instances = json.loads(GET(network_adapters_url))
        get_instance_status(network_adapters_instances)
```

Code listing B.4: health check output - Referenced in 4.4.4

```
[root@bachelor scripts]# ./health_check.py
***********Chassis***************

Chassis health summary:
{
    "Health": "Critical",
    "HealthRollup": "Critical",
    "State": "Enabled"
}
Power supplies:
"/redfish/v1/Chassis/System.Embedded.1/Power#/PowerSupplies/0"
{
    "Health": "Critical",
    "State": "Disabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Power#/PowerSupplies/1"
{
    "Health": "OK",
    "State": "Enabled"
}
Thermal:
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Fans/0"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Fans/0"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Fans/1"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Fans/1"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Fans/2"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Fans/2"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Fans/3"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Fans/3"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Temperatures/InletTemp"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Temperatures/CPU1Temp"
{
    "Health": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/Thermal#/Temperatures/CPU2Temp"
{
    "Health": "OK",
    "State": "Enabled"
}
Network adapters:
"/redfish/v1/Chassis/System.Embedded.1/NetworkAdapters/NIC.Embedded.1"
{
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/NetworkAdapters/NIC.Mezzanine.3"
{
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
}
"/redfish/v1/Chassis/System.Embedded.1/NetworkAdapters/InfiniBand.Slot.4"
{
    "Health": "OK",
    "HealthRollup": "OK",
    "State": "Enabled"
}
```

Code listing B.5: DHCP config

```
# Command used to install dhcp
yum -y install dhcp



# /etc/dhcp/dhcpd.conf

# dhcpd.conf

default-lease-time 3600;
max-lease-time 7200;

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).
log-facility local7;

# A slightly different configuration for an internal subnet.
subnet 192.168.0.0 netmask 255.255.255.0 {
  range 192.168.0.10 192.168.0.254;
  option routers 192.168.0.1;
  default-lease-time 1814400; #21 days
  max-lease-time 3628800; #42 days
}
```

Code listing B.6: Pseudo Code - Referenced in 4.4.4

```
If category == Systems

    Get system instance(s)
        Get contents of  instance ['Members']
    for each instance in instance['Members']
        # Memory health
        Get memory status summary from instance["MemorySummary"]
        Find memory instance url from  instance['Memory']['@odata.id']
        For each member in ['Members'] array of instance:
            Print ['@odata.id']
            Print ['Status']

        # Processor
        Get processor status summary from instance['ProcessorSummary']
        Find processor instance url from instance['Processors']['@odata.id']
            For each member in ['Members'] array:
                Print ['@odata.id']
                Print ['Status']


        # Network interfaces
        Get ['NetworkInterfaces'] resource url
        For each element in ['Members']
            Print ['@odata.id']
            Print ['Status']

       # PCIeDevices
       Get ['PCIeDevices'] resource url
        For each device in ['PCIeDevices'] array
                Get device URI
                Print ['@odata.id']
                Print ['@Status']

        # Storage
        Get ['Storage'] resource url
        For each member in storage['Members']
            Get member URL
            Print response['@odata.id']
            Print response['Status']

If category == Managers
    Build full URL
    Get manager instance(s) url
    For each member of manager instances ['Members']
        Get member URL
        Print response['Status']

If category == Chassis
    Build full URL
    Get  chassis instance(s) url
    Get chassis health summary from instance['Status']
    #Power
    #PowerSupplies
    For each element in ['PowerSupplies']:
                Print ['@odata.id']
                Print ['Status']

   #Thermal
```

```
Get Thermal URL ['Thermal']['@odata.id']
#Thermal - fans
for each fan in thermal['Fans'] array
    print ['@odata.id']
    print ['Status']


# Thermal - Temperatures
For each element in thermal['Temperatures'] array
      print ['@odata.id']
      print ['Status']

 #Network adapters:
 Get network adapters URL ['NetworkAdapters']['@odata.id']
 For each network adapter instance in ['Members']
     Get instance URL ['@odata.id]
         print response ['@odata.id']
         print response['Status']
```

Code listing B.7: The Ansible playbook YAML file for server health check using Redfish - Referenced in 4.5.1

```
---
# Playbook for server health check using the redfish plugin in community.general
    collection (version 2.2.0)
- hosts: test2
  connection: local
  gather_facts: yes
  name: Get server health information
  vars:
    test_mode: 0
    ansible_python_interpreter: /usr/bin/python3.6


  tasks:
  - name: Include host specific Ansible Vault encrypted variables
    include_tasks: tasks/include_host_vars.yml
    when: not test_mode


  - name: Create output file
    include_tasks: tasks/create_output_file.yml


  - name: Get system health report
    redfish_info:
      category: Systems
      command: GetHealthReport
      baseuri: "{{ ansible_host }}"
      username: "{{ vault_default_user }}"
      password: "{{ vault_default_password }}"
    register: systems_result


  - name: Get chassis health report
    redfish_info:
      category: Chassis
      command: GetHealthReport
      baseuri: "{{ ansible_host }}"
      username: "{{ vault_default_user }}"
      password: "{{ vault_default_password }}"
    register: chassis_result


  - name: Get manager health report
    redfish_info:
      category: Manager
      command: GetHealthReport
      baseuri: "{{ ansible_host }}"
      username: "{{ vault_default_user }}"
      password: "{{ vault_default_password }}"
    register: manager_result


  - name: Save results to output file
    copy:
      content: "{{ systems_result | to_nice_json }} {{ chassis_result | to_nice_
          json }} {{ manager_result | to_nice_json }}"
      dest: "{{ template }}.json"
```

Code listing B.8: Ansible Module for Import Configuration Preview - Referenced in 4.5.7

```
#!/usr/bin/python
# Copyright: (c) 2020,
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl
     -3.0.txt)
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

DOCUMENTATION = r'''
---
module: import_idrac_scp_preview

short_description: Import a given idrac server configuration profiles without
     making any changes to target host (preview).

version_added: -

description: This module previews an idrac server configuration profile from a
     network share on host(s) by utilizing Redfish API, specifically the OEM action
     EID_674_Manager.ImportSystemConfigurationPreview. It returns an "Accepted" or "
     Error" message. This can be used to test import of an SCP on multiple hosts
     safely without making any changes. Currently only supports network share type
     NFS.

options:
    idrac_ip:
        description: This is the message to send to the test module.
        required: true
        type: str
    idrac_user:
    idrac_password:
    share_type:
    share_name:
    resource_type:

    scp_file:
    scp_components:


author:
    - Your Name (@yourGitHubHandle)
'''

EXAMPLES = r'''
# Pass in a message
- name: Test with a message
  my_namespace.my_collection.my_test_info:
    name: hello world
'''

RETURN = r'''
# These are examples of possible return values, and in general should use other
     names for return values.
original_message:
    description: The original name param that was passed in.
    type: str
    returned: always
    sample: 'hello world'
```

```
message:
    description: The output message that the test module generates.
    type: str
    returned: always
    sample: 'goodbye'
my_useful_info:
    description: The dictionary containing information about your system.
    type: dict
    returned: always
    sample: {
        'foo': 'bar',
        'answer': 42,
    }
'''

from ansible.module_utils.basic import AnsibleModule
from ansible.module_utils.urls import open_url
from ansible.module_utils.six.moves.urllib.error import URLError, HTTPError
from ansible.module_utils._text import to_native
import json
from ansible_collections.community.general.plugins.module_utils.redfish_utils
     import RedfishUtils
import re
import time

class ExtendedRedfishUtils(RedfishUtils):

    def import_config_preview(self):
        result = {}
        action_uri = "iDRAC.Embedded.1/Actions/Oem/EID_674_Manager.
             ImportSystemConfigurationPreview"
        full_uri = self.root_uri + action_uri

        post_payload = {
            'ShareParameters':
            {
            'ShareType': self.module.params['share_type'],
            'ShareName': self.module.params['share_name'],
            'IPAddress': self.module.params['share_ip'],
            'FileName': self.module.params['scp_file'],
            'Target': self.module.params['scp_components']
            }

        }
        response = self.post_request(full_uri, post_payload)
        result['ret'] = response['ret']
        if response['ret'] is False:
            result['Message'] = response['msg']
            return result

        # Identify job id
        response_output = response['resp'].__dict__
        headers_loc = response_output["headers"]["Location"]
        job_id = re.search("JID_.+", headers_loc).group()

        # Fill result dict
        result['Message'] = "Import SCP preview job created"
        result['job_id'] = job_id

        return result
```

```

def run_module():

    # define result dict
    result = {}
    # the AnsibleModule object will be our abstraction working with Ansible
    module = AnsibleModule(
        argument_spec = dict(
            # defaults
            timeout = dict(required=False, type='int', default = 10),
            #idrac credentials
            idrac_ip = dict(required=True, type='str'),
            idrac_user = dict(required=True, type='str'),
            idrac_password = dict(required=True, type='str', no_log=True),
            # network share
            share_ip = dict(required=True, type='str'),
            share_name = dict(required=True, type='str'),
            share_type = dict(required=True, type='str'),
            scp_file = dict(required=True, type='str'),
            scp_components = dict(required=False, choices=['ALL', 'IDRAC', 'BIOS',
                'NIC', 'RAID'], default='ALL'),
        ),
        supports_check_mode=False)

    # credentials
    creds = {'user': module.params['idrac_user'],
             'pswd': module.params['idrac_password']}
    # Build URL
    root_uri = "https://" + module.params['idrac_ip'] + "/redfish/v1/Managers/"

    # create RedfishUtils object
    rf_utils = ExtendedRedfishUtils(creds, root_uri, module.params['timeout'],
         module)

    result = rf_utils.import_config_preview()

    # Return #TODO: Error handling
    if result['ret'] is True:
        module.exit_json(msg=to_native(result),job_id=to_native(result['job_id']))

    else:
        module.fail_json(msg=to_native(result))

def main():
    run_module()


if __name__ == '__main__':
    main()
```

Code listing B.9: Ansible Module Get_Job_Details - Referenced in 4.5.7

```
#!/usr/bin/python
# Copyright: (c) 2020,
# GNU General Public License v3.0+ (see COPYING or https://www.gnu.org/licenses/gpl
     -3.0.txt)
from __future__ import (absolute_import, division, print_function)
__metaclass__ = type

DOCUMENTATION = r'''
```

```
---
module: get_job_details_redfish

short_description: Get details about an asynchronous Redfish job/task.

version_added: -

description: This is a PoC module which will return the body of the GET request on
     the Redfish URI of the task, return value (true or false), and the state of the
      task (complete, running).

notes:
    - Run this module from a system that direct access to a BMC with implemented
        Redfish specification > 1.6.0.
    - Tested on DellEMC iDRAC fw version 4.32.10.00.

options:
    ip:
        description: IP address of the OOB controller
        required: true
        type: str
    username:
        description: Username for authentication with the OOB controller
        required: true
        type: str
    password:
        description: Password for authentication with the OOB controller
        required: true
        type: str
    job_id:
        description: Job id of the task to query
        required: true
        type: str
    manager_id:
        description: ID of the OOB manager instance
        required: true
        type: str

'''

EXAMPLES = r'''
# Get details about the job, store return in output
  - name: Get job details
    get_job_details_redfish:
      ip: "{{ ansible_host }}"
      username: "{{ ansible_user }}"
      password: "{{ ansible_password }}"
      job_id "{{testout.job_id }}" # job_id is gotten from previous task that
          starts a job and returns job_id
    register: output

# Get details about the job untill state is either failed or complete ( != running)
     . Retry 10 times.
  - name: Get job details
    get_job_details_redfish:
      ip: "{{ ansible_host }}"
      username: "{{ ansible_user }}"
      password: "{{ ansible_password }}"
      job_id: "{{ testout.job_id }}"
    register: output
    until: output.JobState != "Running"
    retries: 10

# Print response in console
  - name: print
    ansible.builtin.debug:
      msg: "{{ output }}"
'''


RETURN = r'''
# Return messages coming soon.
'''

from ansible.module_utils.basic import AnsibleModule
from ansible.module_utils.urls import open_url
from ansible.module_utils.six.moves.urllib.error import URLError, HTTPError
from ansible.module_utils._text import to_native
import json
from ansible_collections.community.general.plugins.module_utils.redfish_utils
     import RedfishUtils

class ExtendedRedfishUtils(RedfishUtils):
     def get_job_id_details(self, job_id):
        result = {}
        task_uri = "TaskService/Tasks/" + job_id
        full_uri = self.root_uri + task_uri
        response = self.get_request(full_uri)
        if response['ret'] is False:
            result['response'] = "fail url: %s" %full_uri
            result['ret'] = False
            return result
        result['response'] = response
        result['ret'] = response['ret']
        result['JobState'] = response['data']['TaskState']
        return result

def run_module():

    # define result dict
    result = {}
    # the AnsibleModule object will be our abstraction working with Ansible
    module = AnsibleModule(
        argument_spec = dict(
            # defaults
            timeout = dict(required=False, type='int', default = 10),
            # credentials
            ip = dict(required=True, type='str'),
            username = dict(required=True, type='str'),
            password = dict(required=True, type='str', no_log=True),
            # module specifics
            job_id = dict(required=True, type='str'),
        ),
        supports_check_mode=False)

    # credentials
    creds = {'user': module.params['username'],
             'pswd': module.params['password']}
    # Build URL
    root_uri = "https://" + module.params['ip'] + "/redfish/v1/"
    # create RedfishUtils object
    rf_utils = ExtendedRedfishUtils(creds, root_uri, module.params['timeout'],
        module)
    result = rf_utils.get_job_id_details(module.params['job_id'])

    # Return the result
    if result['ret'] is True:
        module.exit_json(msg=to_native(result), JobState=to_native(result['JobState
            ']))

    else:
        module.fail_json(msg=to_native(result))

def main():
    run_module()


if __name__ == '__main__':
    main()
```

Code listing B.10: Output from running preview SCP config file - Referenced in 6.4.1

```
[root@bachelor redfish-ansible]# ansible-playbook -i inventory/ playbooks/import_
    scp_preview.yml

PLAY [test new module]
****************************************************************************

TASK [Gathering Facts]
****************************************************************************
ok: [idrac3]

TASK [Include task include_host_vars]
****************************************************************************
included: /home/redfish/test/git/bachelor_repo/redfish-ansible/playbooks/tasks/
    include_host_vars.yml for idrac3

TASK [Include host specific Ansible Vault encrypted variables]
****************************************************************************
ok: [idrac3]

TASK [run new module]
****************************************************************************
ok: [idrac3]

TASK [dump test output]
****************************************************************************
ok: [idrac3] => {
    "msg": "{'msg': \"{'ret': True, 'Message': 'Import SCP preview job created', '
        job_id': 'JID_213488750785'}\", 'job_id': 'JID_213488750785', 'failed':
        False, 'changed': False} -------------------------\n jobid: JID
        _213488750785 "
}

TASK [Get job details]
****************************************************************************
FAILED - RETRYING: Get job details (10 retries left).
FAILED - RETRYING: Get job details (9 retries left).
FAILED - RETRYING: Get job details (8 retries left).
ok: [idrac3]

TASK [dump job details output]
***************************************************************************
ok: [idrac3] => {
    "msg": {
        "JobState": "Completed",
        "attempts": 4,
        "changed": false,
        "failed": false,
        "msg": "{'response': {'ret': True, 'data': {'@odata.context': '/redfish/v
            1/$metadata#Task.Task', '@odata.id': '/redfish/v1/TaskService/Tasks/JID
            _213488750785', '@odata.type': '#Task.v1_4_2.Task', 'Description': '
            Server Configuration and other Tasks running on iDRAC are listed here',
             'EndTime': '2021-05-18T16:41:31+02:00', 'Id': 'JID_213488750785', '
            Messages': [{'Message': 'A system reboot is required to apply
            configuration changes.', 'MessageArgs': [], 'MessageArgs@odata.count':
            0, 'MessageID': 'SYS087'}, {'Message': 'Successfully previewed Server
            Configuration Profile import operation.', 'MessageArgs': [], '
            MessageArgs@odata.count': 0, 'MessageId': 'SYS081'}], 'Messages@odata.
            count': 2, 'Name': 'Preview Configuration', 'Oem': {'Dell': {'@odata.
            type': '#DellJob.v1_0_4.DellJob', 'CompletionTime': '2021-05-18T
            16:41:31', 'Description': 'Job Instance', 'EndTime': None, 'Id': 'JID
            _213488750785', 'JobState': 'Completed', 'JobType': '
            PreviewConfiguration', 'Message': 'Successfully previewed Server
            Configuration Profile import operation.', 'MessageArgs': [], 'MessageId
            ': 'SYS081', 'Name': 'Preview Configuration', 'PercentComplete': 100, '
            StartTime': 'TIME_NOW', 'TargetSettingsURI': None}}, 'PercentComplete':
             100, 'StartTime': '2021-05-18T16:41:15+02:00', 'TaskState': 'Completed
            ', 'TaskStatus': 'OK'}, 'headers': {'date': 'Tue, 18 May 2021 14:41:33
            GMT', 'server': 'Apache', 'link': '</redfish/v1/Schemas/Task.v1_4_2.
            json>;rel=describedby', 'odata-version': '4.0', 'access-control-allow-
            origin': '*', 'cache-control': 'no-cache', 'x-frame-options': 'DENY', '
            strict-transport-security': 'max-age=63072000; includeSubDomains;
            preload', 'content-length': '1200', 'connection': 'close', 'content-
            type': 'application/json;odata.metadata=minimal;charset=utf-8'}}, 'ret
            ': True, 'JobState': 'Completed'}"
    }
}

PLAY RECAP
***************************************************************************
idrac3                     : ok=7    changed=0    unreachable=0    failed=0
    skipped=0    rescued=0    ignored=0
```

Code listing B.11: Output from Server_Setup.yml - Referenced in 6.4.2

```
[root@bachelor redfish-ansible]# ansible-playbook -i inventory/ playbooks/server_
    setup.yml

PLAY [Apply BMC settings]
***************************************************************************

TASK [Get server information]
    ***************************************************************************
ok: [idrac3]
```

```
TASK [bios_idrac_settings : Include host specific Ansible Vault encrypted variables
     ]
****************************************************************************
included: /home/redfish/test/git/bachelor_repo/redfish-ansible/roles/bios_idrac_
     settings/tasks/../../../playbooks/tasks/include_host_vars.yml for idrac3

TASK [bios_idrac_settings : Include host specific Ansible Vault encrypted variables
     ]
****************************************************************************
ok: [idrac3]

TASK [bios_idrac_settings : Set BIOS settings]
****************************************************************************
changed: [idrac3]

TASK [bios_idrac_settings : Create BIOS configuration job (schedule BIOS setting
     update)]
****************************************************************************
changed: [idrac3]

TASK [bios_idrac_settings : Restart system]
****************************************************************************
changed: [idrac3]

TASK [idrac_settings : Include host specific Ansible Vault encrypted variables]
****************************************************************************
included: /home/redfish/test/git/bachelor_repo/redfish-ansible/roles/idrac_settings
     /tasks/../../../playbooks/tasks/include_host_vars.yml for idrac3

TASK [idrac_settings : Include host specific Ansible Vault encrypted variables]
****************************************************************************
ok: [idrac3]

TASK [idrac_settings : Set iDRAC settings]
****************************************************************************
changed: [idrac3]

TASK [idrac_settings : debug output]
****************************************************************************
ok: [idrac3] => {
    "msg": {
        "changed": true,
        "failed": false,
        "msg": "SetManagerAttributes: Modified Manager attributes {'IPMISOL.1.
             Enable': 'Disabled', 'NIC.1.VLanID': '1', 'Users.2.Password':
             '********', 'IPBlocking.1.FailCount': '3', 'IPBlocking.1.FailWindow':
             '60', 'IPBlocking.1.PenaltyTime': '60', 'SSHCrypto.1.HostKeyAlgorithms
             ': 'ssh-rsa,rsa-sha2-512,rsa-sha2-256,ecdsa-sha2-nistp256,ssh-ed255'}"
    }
}

PLAY RECAP
****************************************************************************
idrac3                      : ok=10   changed=4    unreachable=0    failed=0
     skipped=0    rescued=0    ignored=0
```

Code listing B.12: Output from Server_Setup.yml against two servers - Referenced in 6.4.3

```
[root@bachelor redfish-ansible]# ansible-playbook -i inventory/ playbooks/server_
    setup.yml

PLAY [Apply BMC settings]
***************************************************************************

TASK [Get server information]
***************************************************************************
ok: [idrac4]
ok: [idrac3]

TASK [Include host specific variables]
***************************************************************************

TASK [common : Include host specific Ansible Vault encrypted variables]
***************************************************************************
ok: [idrac3]
ok: [idrac4]

TASK [bios_idrac_settings : Set BIOS settings]
***************************************************************************
ok: [idrac3]
changed: [idrac4]

TASK [bios_idrac_settings : Create BIOS configuration job (schedule BIOS setting
    update)]
***************************************************************************
skipping: [idrac3]
changed: [idrac4]

TASK [bios_idrac_settings : Restart system]
***************************************************************************
skipping: [idrac3]
changed: [idrac4]

TASK [Include host specific variables]
***************************************************************************

TASK [common : Include host specific Ansible Vault encrypted variables]
***************************************************************************
ok: [idrac3]
ok: [idrac4]

TASK [idrac_settings : Set iDRAC settings]
***************************************************************************
changed: [idrac3]
changed: [idrac4]

TASK [idrac_settings : debug output]
***************************************************************************
ok: [idrac3] => {
    "msg": {
        "changed": true,
        "failed": false,
        "msg": "SetManagerAttributes: Modified Manager attributes {'NIC.1.VLanID':
            '1', 'Users.2.Password': '********', 'IPBlocking.1.FailCount': '3', '
            IPBlocking.1.FailWindow': '60', 'IPBlocking.1.PenaltyTime': '60', '
            SSHCrypto.1.HostKeyAlgorithms': 'ssh-rsa,rsa-sha2-512,rsa-sha2-256,
            ecdsa-sha2-nistp256,ssh-ed255'}"
    }
}
ok: [idrac4] => {
    "msg": {
        "changed": true,
        "failed": false,
        "msg": "SetManagerAttributes: Modified Manager attributes {'NIC.1.VLanID':
             '1', 'Users.2.Password': '********', 'IPBlocking.1.FailCount': '3', '
             IPBlocking.1.FailWindow': '60', 'IPBlocking.1.PenaltyTime': '60', '
             SSHCrypto.1.HostKeyAlgorithms': 'ssh-rsa,rsa-sha2-512,rsa-sha2-256,
             ecdsa-sha2-nistp256,ssh-ed255'}"
    }
}

PLAY RECAP
****************************************************************************
idrac3                     : ok=6    changed=1    unreachable=0    failed=0
     skipped=2    rescued=0    ignored=0
idrac4                     : ok=8    changed=4    unreachable=0    failed=0
     skipped=0    rescued=0    ignored=0
```

Code listing B.13: Excerpt of a GET request to the biosregistry resource - Referenced in 4.4.2

```
curl -sX GET  -u root:redfish -k https://192.168.0.123/redfish/v1/Systems/System.
    Embedded.1/Bios/BiosRegistry | python -m json.tool
#Excerpt:
    "AttributeName": "LogicalProc",
                "CurrentValue": null,
                "DisplayName": "Logical Processor",
                "DisplayOrder": 600,
                "HelpText": "Each processor core supports up to two logical
                    processors. When set to Enabled, the BIOS reports all logical
                    processors. When set to Disabled, the BIOS only reports one
                    logical processor per core.    Generally, higher processor
                    count results in increased performance for most multi-threaded
                    workloads and the recommendation is to keep this enabled.
                    However, there are some floating point/scientific workloads,
                    including HPC workloads, where disabling this feature may
                    result in higher performance.",
                "Hidden": false,
                "Immutable": false,
                "MenuPath": "./ProcSettingsRef",
                "ReadOnly": false,
                "ResetRequired": true,
                "Type": "Enumeration",
                "Value": [
                    {
                        "ValueDisplayName": "Enabled",
                        "ValueName": "Enabled"
                    },
                    {
                        "ValueDisplayName": "Disabled",
                        "ValueName": "Disabled"
                    }
                ],
                "WarningText": null,
                "WriteOnly": false
            },
```

Code listing B.14: GET request to confirm pending BIOS configuration changes - Referenced in 4.4.2

```
[root@bachelor redfish-ansible]# curl -X GET -u root:redfish -k https
     ://192.168.0.123/redfish/v1/Systems/System.Embedded.1/Bios/Settings| python -m
     json.tool
{
    "@odata.context": "/redfish/v1/$metadata#Bios.Bios",
    "@odata.id": "/redfish/v1/Systems/System.Embedded.1/Bios/Settings",
    "@odata.type": "#Bios.v1_1_0.Bios",
    "Id": "Settings",
    "Name": "BIOS Configuration Pending Settings",
    "Description": "BIOS Configuration Pending Settings. These settings will be
        applied on next system reboot.",
    "AttributeRegistry": "BiosAttributeRegistry.v1_0_3",
    "Attributes": {
        "LogicalProc": "Disabled"
    },
    "Actions": {
        "Oem": {
            "DellManager.v1_0_0#DellManager.ClearPending": {
                "target": "/redfish/v1/Systems/System.Embedded.1/Bios/Settings/
                    Actions/Oem/DellManager.ClearPending"
            }
        }
    },
    "Oem": {
        "Dell": {
            "@odata.context": "/redfish/v1/$metadata#DellManager.DellManager",
            "@odata.type": "#DellManager.v1_0_0.DellManager",
            "Jobs": {
                "@odata.id": "/redfish/v1/Managers/iDRAC.Embedded.1/Jobs"
            }
        }
    }
}
```

Code listing B.15: travisYML - Referenced in 7.2

```
---
# CI
# Based, but expanded upon:
# https://github.com/dell/redfish-ansible-module (Dell repository with sample
    playbooks)
# https://hobo.house/2019/08/05/how-to-setup-travis-for-quick-ansible-playbook-ci/
    (Will Foster)

language: python

# install ansible
addons:
   apt:
      packages:
      - python-pip
install:
   # install requirements
   - pip install -r redfish-ansible/tests/test-requirements.txt
before_script:
   # Dummy ansible vault file to pass the prompt for vault-password
   - mkdir -p ~/.ansible/vault && echo "test-password" > ~/.ansible/vault/vaultpass
       .txt
   # copy the modules and roles to Ansible default path
   - mkdir -p ~/.ansible/plugins/modules
   - cp redfish-ansible/plugins/modules/* ~/.ansible/plugins/modules
   - mkdir -p ~/.ansible/roles/
   - cp -r redfish-ansible/roles/* ~/.ansible/roles
   # change dir
   - cd redfish-ansible
script:
   # basic syntax check
   - ansible-playbook playbooks/*.yml --extra-vars "test_mode=true" -i /tests/
       inventory --syntax-check
   # lint-test, ommit warnings with -x 303,701,601,206,602,403,301,502,306
   - ansible-lint -x 503 playbooks/*.yml
   - yamllint ~/.ansible/roles/
```

Appendix C

Other

IPMI tools and trends

System administrators can use tools like ipmitool, ipmiutil, freeipmi, openipmi [23] to query information from the IPMI-based system over a command-line interface. The employer recommended the use of "ipmitool", but no reason behind the choice was given. To try to gauge which one of the tool for OOB management with IPMI is generally the most used in the IT administration world, Google Trends data was used. Google Trends measures the search-interest of user-specified terms in a specific time period. This "interest" is represented as a number from 0-100, where 100 is the most popular, and 50 is half as popular among the selected terms in the different time periods.

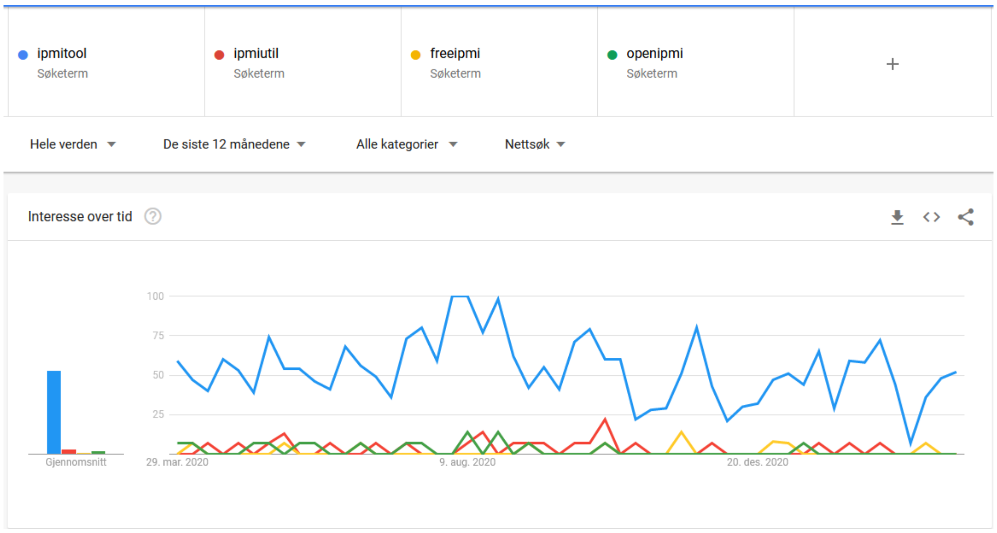


Figure C.1: Google Trends the last 12 months.

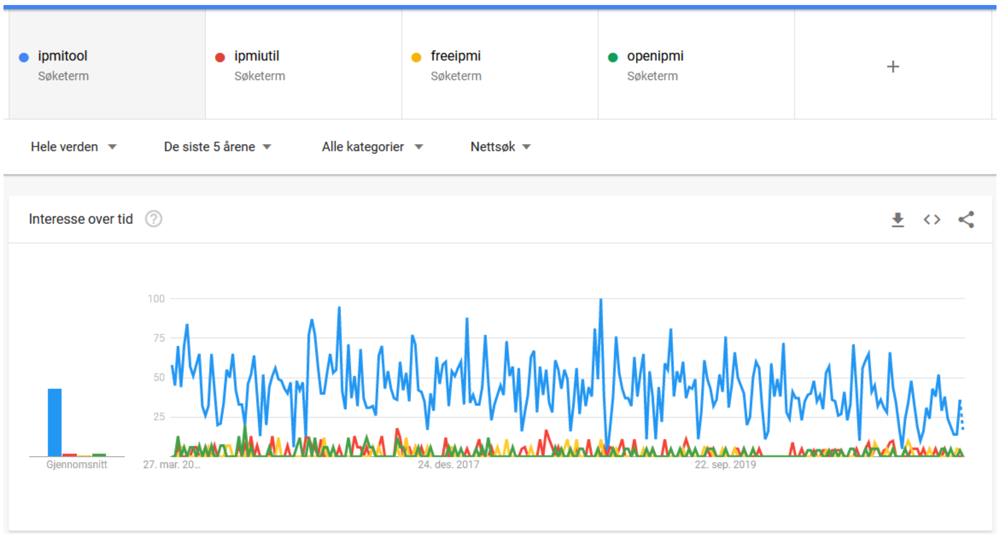


Figure C.2: Google Trends the last 5 years.

URL: https://trends.google.com/trends/explore?q=ipmitool,ipmiutil,freeipmi,openipmi
The data shows that 'ipmitool' has by far the highest search interest, averaging 31 (last 5 years) and 51 (last 12 months), while no other single tool had an average higher than 3 in neither time period. This could imply that 'ipmitool' also is the most used tool by administrators (by the ones mentioned) for OOB management with IPMI, as it would be common to google-search the term for either download-instructions or documentation. This is confirmed even more by adding the word "download" behind each tool-name. The term "ipmitools download" had an average search interest of 11 over the last 5 years, while all other tools had an average of 0. This could seem weird at first, 11/100 does not seem like a big number. If "ipmitools" was in fact the "go-to"-tool for OOB management, a higher average might be expected. This could however be explained by the fact that after the tool has been downloaded on a controller/master-node, it does not have to be downloaded again. This would reduce the reason for searching for the term for an extended time-period, which again would reduce the average search interest. The administrator that has to download the tool again might also remember the simple command to do it on his or hers operating system. What we can observe however, is that all other "{ipmi tool} download" terms has the average search interest of 0, supporting the theory that "ipmitool" is the most popular tool for OOB management with IPMI among the tools mentioned. This theory however does not account for any other tools which might not have been included in the "Software utilities for IPMI"[23] article which was used as basis for the overview of this comparison. Since IPMI is a specific standard the aforementioned tools are all ipmi related, and thus do not include monitoring and administration tools with similar applications/uses.

Appendix D

Project plan

Project plan

Server configuration with Redfish and Ansible

DCSG2900 Bachelor Thesis Bachelor of Science in Digital Infrastructure and Cyber Security

Magnus Walmsnæss Refsnes, Raymond Paradero Aardalsbakke, Gaute Hiis-Hauge and Nick Zakharov

01.02.2021

Table of contents

1 Goals and Constraints

1.1 Background

NTNU hosts the Idun cluster, which is a combination of computing resources to provide a testing environment for high performance computing (HPC) software. Idun is a high performance computer [1], meaning it is a complex, very fast computer, capable of processing computing tasks like no ordinary PC can. This cluster is composed of servers, which in turn is administered through a baseboard management controller(BMC) [2]. A BMC is a processor that monitors internal variables about the computer through sensors. The BMC also has remote capabilities, although a bit limited. Through this controller it can configure the different hardware of the servers.

The BMC is part of the Intelligent Platform Management Interface (IPMI)[3]. IPMI is an interface used to monitor a server's physical health, such as temperature, voltage, humidity and OS functions. It's a useful tool used by administrators to gain remote management capabilities for a server.

As it stands now; connecting a new server to the Idun cluster is manual labour. Everything from unpacking, connecting the cables to the manual BIOS configuration. While we can't do much about the unpacking part of the job, we can save time and make the lives of the IT staff a lot easier by automating the configuration using the Redfish standard [4] and Ansible. [5]

The redfish standard is designed to be easy to read by both humans and machines. It's a management standard that defines protocols, behaviors and other components to deliver an industry standard protocol.

Ansible is a software tool used to automate tasks. These tasks can range from simple scripts to start a job, all the way to cloud provisioning and configuration management.

1.2 Project goals

- A study of the viability of using Redfish and Ansible in a hpc environment for the setup and configuration of new server additions to the cluster.
- A prototype/proof-of-concept of secure automatic configuration of servers for a high performance computing environment using Redfish and Ansible.
- A finished bachelor thesis containing the process, findings, and results of the project, demonstrating the fulfillment of the learning outcome in the course DCSG2900 to a good degree.

1.3 Constraints

We are asked to use the open-source tool, Ansible, to work on the infrastructure. We will also be using DMTF's standard Redfish. During the time period from start to end of our report we are limited by regulations following the virus COVID-19, which in short limits our ability to meet and work physically. VPN connection when we are working away from campus is necessary to access the test-nodes set up for us in Trondheim.

2 Scope

2.1 Field of study

This project's field of study is the automation of the configuration management of servers making up the IDUN HPC-cluster at NTNU in Trondheim. It will also cover other areas in IT operations like operating systems, networking, and security.

2.2 Delimitation

The project is tackling multiple technologies, with the purpose of achieving the goals set by the project group. It is necessary to clarify the different restrictions that have been decided upon, which are set to limit the project scope to a reasonable degree.
Limitations:

- The use of Redfish will be a demo tailored to NTNU's needs and is not meant to cover all aspects of Redfish
- The demo created will be a proof-of-concept
- Ansible in conjunction with a Redfish-module will automate the configuration process
- While redfish could be used on switches and other network equipment which support it, this project will solely focus on server functionality.

2.3 Problem Statement

At the time of writing NTNU are locally and manually configuring servers and switches at the initial setup in the IDUN cluster. Subsequent updates to the configuration can be done remotely. The standard utilized for configuration is IPMI 2.0[6], which dates back to 2004, not counting revisions. This makes IPMI 2.0 a more than a decade old standard where changes are done through the BIOS. Out of bound management has become a time-consuming issue when dealing with multiple servers or switches. This is where Redfish comes in, which enables scalable configuration management.

However NTNU is not familiar with Redfish and wishes to explore its capabilities. That is why our group has been tasked with creating a proof-of-concept that should provide insight and a proposal for automating configuration management with the Redfish standard and Ansible. Ansible has been specified as the automation software of choice, partly because of its popularity and its supposed ease of use.

3 Project organization

3.1 Roles and responsibilities

The project group will work as the project owners(scrum) and leaders. Each group member has the responsibility to work on their given assignments for the project. Other individual roles are specified below.

Name: Einar Næss Jensen
Role: Client
Description: Communicates the grand vision of the project to the group. Provides technical insight.

Name: Ernst Gunnar Gran
Role: Project supervisor
Description: Supervises the project-group (not the project). Provokes thoughts about processes and decisions made by the group throughout the project, and fuels discussions.

Name: Magnus W. Refsnes
Role: Group leader
Description: The leader is responsible for the planning (trello) and holding meetings. The leader also is responsible for distributing work each week, though this is done in council with the other members of the group. The leader should attempt to resolve conflicts and disagreements within the group, and if the group's opinion of what direction to take is in disagreement then the leader is the deciding vote should the vote be even.

Name: Nick Zakharov
Role: Deputy project leader.
Description: The deputy project leader takes the responsibilities of the group leader should the group leader be absent. In addition the deputy project leader is responsible for setting up and maintaining the projects github repository.

Name: Raymond Aardalsbakke
Role: Secretary
Description: The secretary is responsible for writing a meeting summary during each meeting held, and maintaining the structure of google drive.

Name: Gaute Hiis-Hauge
Role: LaTeX expert
Description: The LaTeX expert is responsible for keeping the latex document updated with the project from the drive. The LaTeX expert is also responsible for creation of diagrams and charts.

3.2 Internal rules for the project

3.2.1 Group Rules

1. Meet at the scheduled time
2. In case of absence you must inform the group as soon as possible with a valid reason.
3. Every sunday from 13-16 is a mandatory meeting.
4. Update and maintain the trello board each sprint to reflect the current workload.
5. Each member is expected to spend on average 30 hours per week on furthering the bachelor project.
6. Each member is to inform the others if they are unable to meet a deadline for their workload in good time.
7. Members are to maintain civility when discussing the project.

3.2.2 Sanctions

- Should a member break three rules in a month then the group is to convene to figure out a resolution to the situation.
- Should this not improve the situation then the group is to convene with the supervisor of the thesis for possible resolvement or further actions.

4 Planning and reporting

4.1 Development model

We believe that a linear-sequential model is not a reasonable approach given the nature of this project. While the end result is clear, the path towards the end is more obscured. Our group needs to learn new technologies and frameworks, and plan and develop a solution for the client, as well as limit or expand the scope based on milestones reached during the project-period. We also might need to redesign or rewrite requirements several times as we get more knowledge on the subject at hand, so it is therefore natural for us to pick an agile methodology, with tools which the group is already familiar using, like scrum, kanban and pair programming. We believe that picking tools the team is already well familiar with, has experience, and is happy to work with, will have a huge benefit for the quality of our workflow.

Based on the agile principle of learning from experience, we want to make some adjustments based on our work together in the past. We will be doing short 'daily meetings' every *other* day to discuss our progress, and we will be borrowing the 'pair programming'-technique (originating from XP) to help, guide, and/or oversee each other's work when working together while communicating and streaming our screens on the 'Discord' application. A sprint session of 1 week fits us well given the meeting-interval with our supervisor, and the 'sprint review' meeting is set to be each sunday, followed by a 'sprint planning' meeting.

We are responsible for both defining the scope and keep progressing, and we are directly impacted by the results of the project (including the final report). We therefore view it as appropriate to be our own project owners, and also manage our own backlog. The supervisor and the client will therefore not have to be a part of the weekly sprint meetings. We will instead prepare questions to discuss with our supervisor the coming Tuesday, and the client will provide technical insight throughout the project in our bi-weekly meetings. The 'product backlog' will be organized as a kanban-board on the web-application 'Trello' to have a clear view of progress and each member's tasks and contributions throughout the sprint.The scrum master is decided to be the group leader.

4.2 Meeting Schedule and Decisions of Importance

We are having following meetings:

- A sprint review and planning meeting from 13:00 to 16:00 every sunday, in which the previous week's progress is gone through and discussed and the next week's workload is planned and distributed. In addition, questions for the supervisor and the client are prepared at these meetings.
- Brief meetings (10-15 minutes every other day, starting Mondays) to discuss progress or problems.
- A meeting with the project supervisor every Thursday at 16:00, the focus of these meetings will be to clarify questions and ask for guidance.
- Lastly, a meeting is held bi-weekly with the client at Tuesday 12:30.

Decisions of importance are either taken at the Sunday meeting, or if needed then any member can call for a council of the bachelor group at the earliest convenient time for all members. If the decision is urgent and cannot wait for all members to meet digitally by voice, then it can be held in the text-chat or in council with only three members. Should the meeting be held with only three members then a manifest of the meeting should be made and presented to the fourth.

If there is a difference in opinion then the disagreeing parts must present their case, and a vote is held by the complete group. Should there be a tie, then the group leader breaks the tie.

When necessary, group work sessions are planned with the other group members.

5 Organization of Quality assurance

5.1 Documentation, standards, and source code

The group has created a shared Google Drive[7] folder for articles such as meeting summaries, shared work and other documents relevant to the project. In addition a shared drive has been created with the client, where the client can share relevant information with the group.

The source code for the project will be kept in a private bitbucket repository. When using third-party libraries, the client has requested that we only make use of open-source projects. The project could be taken over for further development. It is therefore mandatory to comment all code, with separate documentation when deemed necessary to get a good comprehension of its function.

5.2 Configuration management

The draft of the report will be created in Google Drive as this is the environment the group is most familiar using. Using Google Drive also safeguards against loss of data due to computer issues as it is saved in a cloud, though the bachelor thesis will also be kept locally by members of the group in order to quickly restore the workflow in case of unforeseen events. In addition to the Google Drive document, a LaTeX Overleaf[8] document will be created and shared within the group. The report will be gradually translated over to LaTeX during the project-period.

We are using Trello for time and task management of the group. In Trello we have a few different tags being, *to do*, *in progress*, *to be reviewed* and *complete*. We are also keeping track of each other's time schedule and mutual meetings through the use of a shared Google Calendar.

Only the group is able to change, add or remove content from the platforms in use. All platforms are kept private and not shared outside of the group.

5.3 Risk Analysis

We have conducted a short risk analysis of our project, where we have identified the risks inherent with a project and found measures which we can take to mitigate these risks.

5.3.1 Scenario

This figure presents several risk scenarios which we either found possible to occur, or of such risk that there should be a few measures to either minimize the probability or reduce the consequence. Risk is calculated from the probability of a scenario occurring times the consequence of that scenario.

Fig. 1

ID	Scenario	P	C	R
1	One or more of the group is infected by corona or a similarly contagious disease causing them to be unable to meet with other members of the team.	2	3	6
2	Dissenting opinions within the group causes disagreements and arguments leading to a social fracture in the group which hampers the project.	2	2	4
3	The client does not provide the group with proper access to the environment or to the needed equipment causing us to be unable to provide a proof of concept.	1	4	4
4	Sudden departure of a member causes us to be unable to complete the project on time.	1	4	4
5	Poor time usage and procrastination of deadlines causes the project to be delayed and possibly not completed in time.	2	3	6
6	A member of the group fails to deliver the work which has been designated, or consistently delivers work of poor quality.	2	3	6
7	A member of the group repeatedly fails to meet at the designated meeting times.	2	2	4
8	Lacking knowledge of redfish and Ansible causes failure to complete the project.	1	3	3
9	Redfish and/or ansible is not compatible with the equipment given by the client, or is not able to do the tasks set by the client causing the group to be unable to complete the project.	2	4	8

5.3.2 Risk Matrix before measures

This matrix depicts the risk of the different scenarios before any measures are taken, with an orange and red risk seen as an unacceptable risk to take.

Fig. 2: Risk matrix before measures

Probability (p)	Consequence (C)			
	Low	Moderate	Severe	Critical
Low	🟩	🟩	🟩 8	🟨 3,4
Moderate	🟩	🟨 2,7	🟨 1,5,6	🟧
High	🟩	🟨	🟧	🟥
Very high	🟨	🟧 9	🟥	🟥

5.3.3 Risk Matrix after measures

This matrix depicts the risk of the different scenarios after the measures are taken, with all scenarios now being either yellow or green. This means they are all within acceptable levels of risk.

Fig. 3 Risk matrix after measures

Probability (p)	Consequence (C)			
	Low	Moderate	Severe	Critical
Low	🟩	🟩	🟩 1,3,5,8	🟨 4
Moderate	🟩 2	🟨 6,7	🟨 9	🟧
High	🟩	🟨	🟧	🟥
Very high	🟨	🟧	🟥	🟥

5.3.4 Measures:

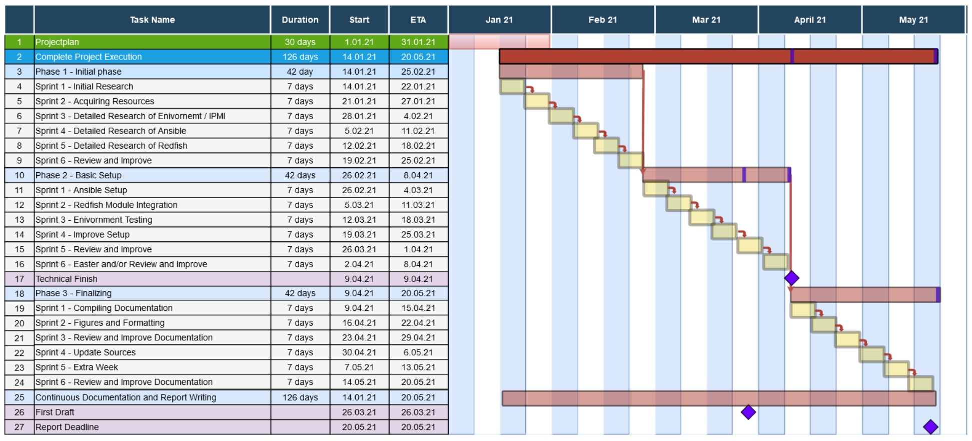
We have decided to focus the measures on things we are able to implement and which are realistic that we can handle, or is of such risk that not having a measure would be irresponsible. Using these measures we believe the remaining risk in the project is acceptable and mostly unavoidable.

Fig. 4: Measures

Measure	Scenario ID	Effect	Remaining Risk PxC=R
All meetings are held digitally, and the group is never gathered together all at the same time for the duration of the project.	1	Reduce probability to 1	1 x 3=3
The group handles disputes according to the rules within the pre-project.	2	Reduce consequence to 1	2 x 1=2
The group takes in use either skyhiig or VMs in order to provide a proof of concept.	3	Reduce consequence to 3	1 x 3=3
Deadlines are set on the workload each week, and the bar for asking for assistance or cooperation is set low. Trello is used to track progress.	5	Reduce probability to 1	1 x 3=3
The work done by each member is peer reviewed at the end of each week to make sure that the quality is up to standards. Repeated events of poor quality work will be discussed with the person responsible, before eventually being brought up with the supervisor.	6	Reduce consequence to 2	2 x 2=4
Changing the scope of the bachelor thesis to a study of other available and feasible technologies.	9	Reduce consequence to 3	2 x 3=6

6 Implementation Plan

Fig. 5: Gantt Chart



	Task Name	Duration	Start	ETA
1	Projectplan	30 days	1.01.21	31.01.21
2	Complete Project Execution	126 days	14.01.21	20.05.21
3	Phase 1 - Initial phase	42 day	14.01.21	25.02.21
4	Sprint 1 - Initial Research	7 days	14.01.21	22.01.21
5	Sprint 2 - Acquiring Resources	7 days	21.01.21	27.01.21
6	Sprint 3 - Detailed Research of Enivornemt / IPMI	7 days	28.01.21	4.02.21
7	Sprint 4 - Detailed Research of Ansible	7 days	5.02.21	11.02.21
8	Sprint 5 - Detailed Research of Redfish	7 days	12.02.21	18.02.21
9	Sprint 6 - Review and Improve	7 days	19.02.21	25.02.21
10	Phase 2 - Basic Setup	42 days	26.02.21	8.04.21
11	Sprint 1 - Ansible Setup	7 days	26.02.21	4.03.21
12	Sprint 2 - Redfish Module Integration	7 days	5.03.21	11.03.21
13	Sprint 3 - Enivornment Testing	7 days	12.03.21	18.03.21
14	Sprint 4 - Improve Setup	7 days	19.03.21	25.03.21
15	Sprint 5 - Review and Improve	7 days	26.03.21	1.04.21
16	Sprint 6 - Easter and/or Review and Improve	7 days	2.04.21	8.04.21
17	Technical Finish		9.04.21	9.04.21
18	Phase 3 - Finalizing	42 days	9.04.21	20.05.21
19	Sprint 1 - Compiling Documentation	7 days	9.04.21	15.04.21
20	Sprint 2 - Figures and Formatting	7 days	16.04.21	22.04.21
21	Sprint 3 - Review and Improve Documentation	7 days	23.04.21	29.04.21
22	Sprint 4 - Update Sources	7 days	30.04.21	6.05.21
23	Sprint 5 - Extra Week	7 days	7.05.21	13.05.21
24	Sprint 6 - Review and Improve Documentation	7 days	14.05.21	20.05.21
25	Continuous Documentation and Report Writing	126 days	14.01.21	20.05.21
26	First Draft		26.03.21	26.03.21
27	Report Deadline		20.05.21	20.05.21

The Gantt chart shows the outline of the general workflow of the entire project. Important dates are marked as milestones while the different sprints should work as smaller but more frequent milestones throughout the project.

Major milestones:

- Technical Finish - 9.04.2021:
    - All the required software and hardware should be completely configured, tested and working. This step is required because some documentation cannot be done until a finished proof-of-concept is made.
- First Draft - 26.03.2021:
    - An Initial draft consisting of documentation compiled so far. The First draft serves the purpose of being reviewed by the supervisor as a source of feedback.
- Report Deadline - 20.05.2021:
    - Hard deadline for the delivery of the final report, since no changes can be done after this date it is the end of the project timeline.

7 References

[1] Inside HPC - What is high performance computing? (-2021) Available at:
https://insidehpc.com/hpc-basic-training/what-is-hpc/ (Accessed: 31.01.2021)

[2] SearchNetworking - Baseboard Management Controller (Last updated May 2007)
Available at:
https://searchnetworking.techtarget.com/definition/baseboard-management-controller
(Accessed 31.01.2021)

[3] Techopedia - Intelligent Platform Management Interface (IPMI) (Last updated July 26, 2016) Available at:
What is Intelligent Platform Management Interface (IPMI)? - Definition from Techopedia
(Accessed 31.01.2021)

[4] DMTF - Redfish standard® (2014-) Available at
https://www.dmtf.org/standards/redfish/ (Accessed at: 31.01.2021)

[5] Open source - What is Ansible? (2020) Available at:
https://opensource.com/resources/what-ansible (Accessed at: 31.01.2021)

[6] Intelligent Platform Management Interface Available at:
https://en.wikipedia.org/wiki/Intelligent_Platform_Management_Interface (Accessed at: 10.02.2021)
[7] Google's own information page for Google Drive Available at:
https://www.google.com/drive/ (Accessed at: 12.02.2021)

[8] NTNU Wiki - Overleaf Available at:
https://innsida.ntnu.no/wiki/-/wiki/English/Overleaf (Accessed at: 11.02.2021)

Appendix E

Group agreement

NTNU

Vår dato

Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og **NTNU IT, Seksjon for IT-drift, servergruppa** (oppdragsgiver), og Nick Zakharov, Gaute Hiis-Hauge, Raymond Aardalsbakke, Magnus W. Refsnes (student(er)).

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.Januar 2021 til 20.Mai 2021

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle beståtte bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

   Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Ernst Gunnar Gran

Oppdragsgivers kontaktperson (navn): Einar Næss Jensen

Student(er) (signatur):

Nick Zakharov 21/1-21

[illegible] 21.01.21

[illegible] 21.01.21

[illegible] 21.01.21

Oppdragsgiver (signatur):

[illegible] dato 26.01.2021

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): ____________________________________ dato ____________