MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Provisioning of Monitoring Infrastructure for Traffic Flow Collection

Bachelor's Thesis

ONDŘEJ MOLÍK

Brno, Fall 2021

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Provisioning of Monitoring Infrastructure for Traffic Flow Collection

Bachelor's Thesis

ONDŘEJ MOLÍK

Advisor: RNDr. Daniel Tovarňák, Ph.D.

CSIRT-MU

Brno, Fall 2021


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

Declaration

Hereby I declare that this thesis is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ondřej Molík

Advisor: RNDr. Daniel Tovarňák, Ph.D.

Acknowledgements

I would like to thank RNDr. Daniel Tovarňák, Ph.D for his guidance and professional advice, which were exceedingly useful and crucial for the creation of this thesis. I am also really grateful to all my colleagues at CSIRT-MU, who provided me with valuable feedback and their cooperation. Further, I want to thank my family and friends for their support throughout my entire studies.

Abstract

This thesis aims to analyse, describe and implement the options of automatic provisioning and lifecycle management of the network monitoring infrastructure used by CSIRT-MU.

Firstly, the particular environment of the CSIRT-MU is introduced along with the technologies that are interacted with and the processes whose automation is to be evaluated or implemented. The primary sources that were used to determine the feasibility of automating the provisioning and lifecycle management are provided in the official documentation and the analysis of the behaviour of the deployed devices. The assessment of the provided options and requirements concludes that the current state of the vendor's software prohibits full automation of the provisioning process. Consequently, the practical result deals with activities that can be automated, e.g. automatic Ansible inventory building, backups, recovery and auditing of the deployed devices.

The work concludes with suggestions for further development that would allow for better automation, provisioning and management of devices and appliances in question.

Keywords

Flowmon, Netbox, Ansible, DevOps, S3

Contents

List of Figures

Introduction

The problem that is to be solved by this thesis is the efficient provisioning and lifecycle management of relatively large network flow monitoring infrastructure at CSIRT-MU (Cybersecurity Team of Masaryk University). This infrastructure is based on products from Flowmon Networks. Previous advances in this area were mainly made by colleagues at CSIRT-MU. Their activity was focused on specific subsections of the provisioning process, especially those parts that are shared by both MI (*Monitoring Infrastructure*) and other server infrastructure at CSIRT-MU. A focused effort towards complex automation of processes explicitly related to MI based on products of Flowmon Networks is yet to be made.

This thesis aims to comprehensively analyse all available options for automatic provisioning of network flow monitoring infrastructure used by CSIRT-MU. Based on preliminary knowledge of the involved environment and technologies, it is possible to say that more than a few aspects are not automatable in their current state. Therefore the result of this thesis is not a state-of-the-art automated solution to every aspect of the provisioning process. Instead, it is a set of Ansible scripts for the parts of the process currently suitable for automation and a comprehensive description of the obstacles that prevent the rest of the process from being automated.

This introduction is followed by chapter 1, in which I introduce the specific environment at CSIRT-MU and explain the reasons that lead to the need for automation in said environment as well as the tools that form the Ansible ecosystem and the basic principles of network flow monitoring using devices with Flowmon OS. In the chapter 2, the current provisioning process and different ways of configuring Flowmon OS are described and analysed. Based on these findings, parts of the provisioning and lifecycle management process that are deemed suitable for automation are chosen. The chapter 3 deals with methods and procedures that were used during the automation of parts selected in chapter 2. The last chapter concludes the entire thesis by evaluating things achieved and a listing of major obstacles that prevented certain parts of the provisioning process from being automated.

The result of this thesis consists of two parts, firstly the detailed analysis of the provisioning process, which is a part of its text, and secondly, a set of Ansible automation scripts for the sections of the provisioning process that were deemed automatable.

1 Background

This chapter describes the specific environment and circumstances at CSIRT-MU (Cybersecurity Team of Masaryk University) and previous achievements and findings relevant to the provisioning of MI (Monitoring Infrastructure).

The MI on the Masaryk University is based on a commercial platform developed by Flowmon Networks and is being maintained by CSIRT-MU. Both institutions have a longstanding close partnership, the result of which are above-standard options for more refined control and a deeper understanding of devices deployed (e.g., having superuser access to all systems) as well as a broader range of debugging options and closer integration with other related systems. On the other hand, this status occasionally leads to technical difficulties not shared by other customers of Flowmon and therefore are significantly harder to diagnose and resolve.

The scope of deployment at CSIRT-MU is among the larger ones, with approximately 30 devices forming the production part of MI. Managing this number of devices manually requires a significant amount of time, which creates a need for automation of specific tasks and processes to make the provisioning, maintenance, and lifecycle management reasonably time-efficient. The second uncommon aspect of MI maintained by CSIRT-MU is frequent testing of experimental software components, which creates processes that need to be automated.

Official documentation[1] does not directly deal with automatic provisioning of devices with Flowmon OS but contains several relevant documents that describe different ways to instrument these devices. Contents of these documents, as well as conclusions drawn from them, are discussed in depth in chapter 2.2.

Only known existing tooling for the outlined purpose was created in CSIRT-MU. Its development was not preceded by comprehensive research of all possible options but merely driven by the need to solve practical issues.

In order to achieve a complete and optimal solution, an initiative from Flowmon Networks would be required since there is currently no complete and stable interface that could be used for the complete configuration of a device with Flowmon OS. However, at the time of writing the thesis, they do not consider this lack of support a major priority since most of theirs customers do not have such a large-scale and dynamic deployment.

---

1. `https://portal.flowmon.com`

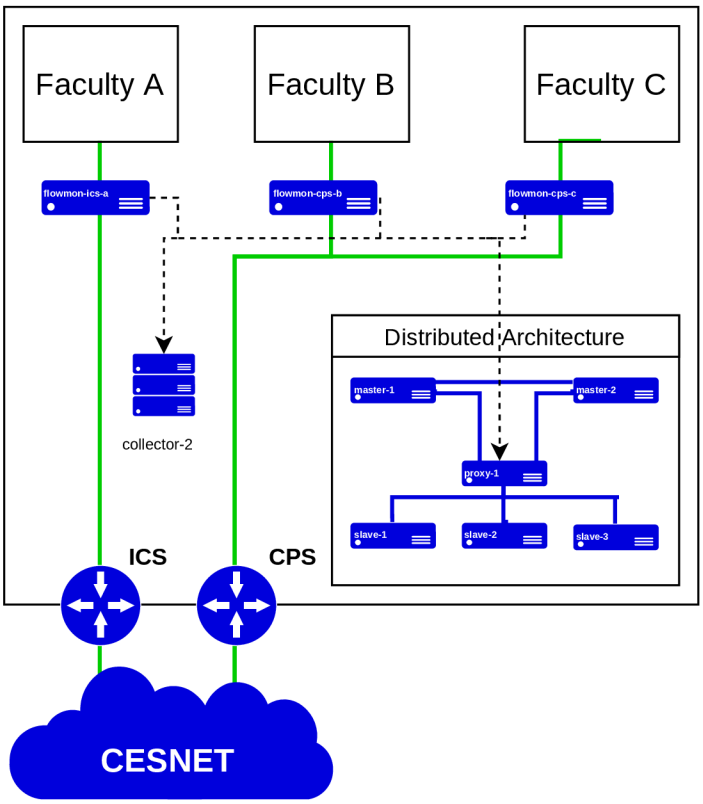

Figure 1.1: Simplified overview of MI at MU

1.1 Flow Monitoring

Flow monitoring is one of the possible ways to monitor network traffic, unlike tools such as Tcpdump[2] that store the entire captured traffic, the flow monitoring stores only metadata and headers of the captured packets [1]. Flow monitoring is generally more space and CPU efficient than the full packet capture tools.

1.1.1 Network Flow

One flow record is usually created from multiple packets that have at least the following attributes in common: source IP address, source port, destination

2. Tcpdump is an open-source tool for capturing and filtering network traffic in PCAP format.

IP address, destination port. The record also stores two timestamps that note the beginning and the end of the flow and the number of packets transmitted, and their total size in bytes [2]. (i.e. one direction of a TCP session will be recorded as one flow which is usually equal to one record).

1.1.2 IPFIX

The naive description in the previous chapter is codified as NetFlow specification version 10, commonly known as IPFIX (Internet Protocol Flow Information Export), originally defined in RFC3917 by Cisco Networks [3]. This format is used for all network flow records at CSIRT-MU. Fields that are collected include [4]:

- Source and destination IP.
- Source and destination port.
- Type of Service.
- Number of packets.
- Total bytes transferred.
- Start and end timestamps.

These basic fields can optionally be extended if both the probe and collector are configured properly. Flowmon heavily utilises this extend-ability (as described in the chapter 2.4).

1.1.3 Network Collector

A network flow collector receives and stores network flow records that are being sent from networks probes. Flow data is being transported in IPFIX format via UDP (*User Datagram Protocol*) or TCP (*Transmission Control Protocol*), if TCP is used optional TLS (*Transport Layer Security*) encryption can be enabled. In order to simplify the occasionally needed debugging, each probe has assigned a different 'Listening port' to which it sends data. On Flowmon Collector received data are stored as a separate file for each five-minute interval in a separate sub-directory for each of the ports, as shown in Listing 1.

```
# probeName ~ name of the flow source
# portName ~ name of the listening port
data_dir="/data/nfsen/profiles-data/live"
date +$data_dir/probeName_portName/%Y/%m/%d/nfcapd.%Y%m%H%m
```

Listing 1: Flow storage directory structure

Table 1.1: The most important detection methods used at CSIRT-MU.

Method	Description
SCANS	various TCP/UDP/ARP scans
RDPDICT	dictionary attacks on RDP
SSHDICT	dictionary attacks on SSH
BLACKLIST	communication with hosts on a blacklist

Collectors usually include components that allow for viewing and analysis of stored flows records. In Flowmon OS, these components are called FMC (*Flowmon Monitoring Center*) and Flowmon ADS (*Anomaly Detection System*). Figure 1.2 displays these two modules as well.

ADS utilises various detection methods to recognise abnormal events in captured flow data. Flowmon ADS offers three categories of methods, firstly those that utilise thresholds, secondly behavioural ones and lastly methods that use a combination of both [5]. The majority of methods used at CSIRT-MU belong to the first category, they only check that a certain type of traffic did not over-exceed the manually configured threshold.

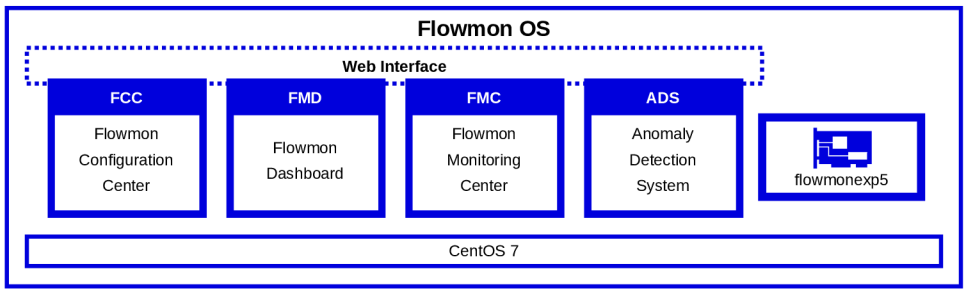

Figure 1.2: Simplified structure of Flowmon OS software modules

1.1.4 Network Probe

A network monitoring probe is a server-like device that is placed in the location of one or more network links and utilises TAP (*Traffic access point*) to capture network traffic that is being transferred over said link. Captured traffic is then analysed by an exporter process which takes raw network traffic that is being sent to one interface of the probe and creates NetFlow or IPFIX records that represent the captured traffic. These flow records can either be stored locally or sent to a remote collector. In the case of

probes that Flowmon Networks manufactures, the exporter program is called `flowmonexp5` and can be extended via dynamically linked libraries stored in `/usr/lib64/flowmonexp5`, its configuration is stored in `/etc/flowmon`. It is important to state that no official documentation suggests direct modification of files in said directory. Rather, the exporter should be configured via the web interface (*'FCC > Listening ports'*) as shown in Figure 2.2.

On Figure 1.3 is shown a sample deployment of one network probe, that monitors the 10-gigabit links between one location and upstream network. Dotted lines represent IPFIX that is being transported over TCP to several collectors. Each solid line represents one 10 gigabit fibre, two of these fibres form one network link of the same speed.

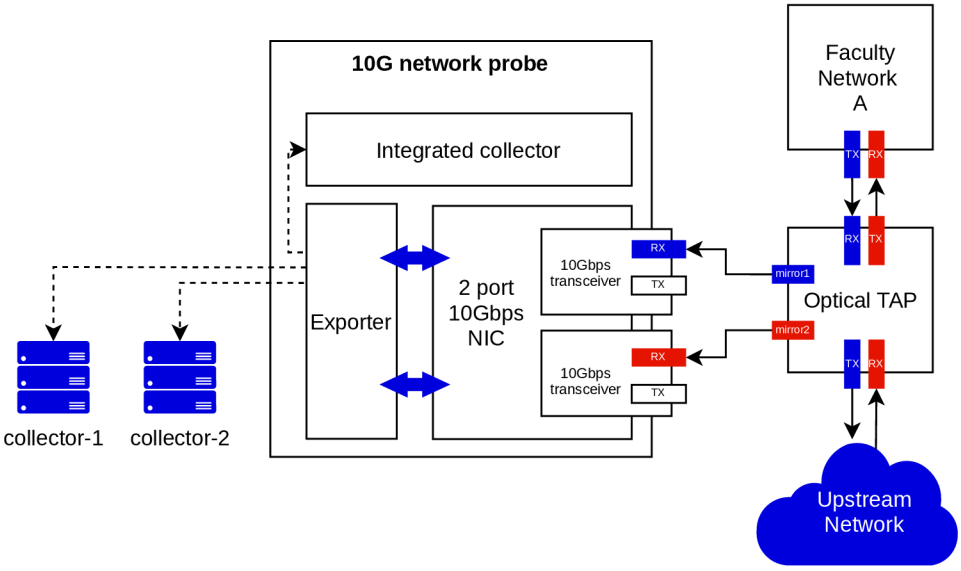

Figure 1.3: One probe monitoring one optical link

Optical Transceiver Sometimes also called 'Optical Module'. Is an adapter that is used by high-speed network cards to allow the connection of different terminations and types of optical fibres and metallic TP (*Twisted Pair*) cables. There are several different form factors of optical transceivers the most frequently used types at CSIRT-MU are QSFP+ (*Quad Small Form-factor Pluggable*) and SFP+ (*Small Form-factor Pluggable*). Similarly to serial communication, each optical transceiver has

two connector slots for fibres, receiving (RX) and transmitting (TX) one. For purposes of network monitoring, only the receiving slot can be used. This means that two transceiver slots are required for monitoring of one link (i.e. two 10 gigabit network card ports are required for monitoring of one 10 gigabit network link).

Optical TAP It takes two fibres as input (one RX and one TX) and mirrors the light in said fibres into two almost identical pairs. Light intensity is being split in a certain ratio (usually 30:70) with the majority of it continuing on the monitored link and a smaller portion being sent to the probe. This is done optically, therefore a power outage in the location of the probe or an issue on the network probe itself will not result in disconnection of the monitored link (contrary to the metallic TAP, failure of which would result in failure of monitored link as well). For outlined reasons, only optical TAPs are deployed in the production part of MI at CSIRT-MU.

1.2 DevOps and Ansible

Before the description of Ansible and its components, a brief introduction to DevOps and infrastructure automation. There are several main motivations for having the entire provisioning process of any infrastructure automated with scripts stored in a Git repository. The first of them is versioning which allows to quickly revert to the last working configuration. The second is reproducibility of deployments, which saves time that would be spent on manual and repetitive labour, furthermore it helps prevent human errors during repetitive tasks. A fully automated provisioning process can be deployed via (*Continuous integration / Continuous delivery*) systems, however, this is not the case of MI at CSIRT-MU.

Ansible is an open-source automation platform developed and maintained by Red Hat inc.. It allows for provisioning and management of multiple servers via SSH (*Secure Shell*) [6].

1.2.1 YAML

YAML (*YAML Ain't Markup Language*) is a data serialisation language. It is more human-readable than JSON (*JavaScript Object Notation*) since it does not require the usage of quotes strings and uses python-like indentation to signal nesting. While JSON files are valid YAML files, YAML offers additional features such as comments and self-referencing using anchors and extend

operators. In Ansible YAML is used for all automation scripts. The official file suffix is `.yaml`, but `.yml` is often used as well [7].

1.2.2 Jinja2

Jinja2 is a python-based templating language developed by The Pallets Project [8]. It allows for manipulation and filtering of the data gathered by Ansible modules. Every YAML file included in an Ansible playbook or a role can be templated via Jinja2, including Ansible Inventory definitions. Additionally to the filters and tests that are available in any standard Jinja2 environment, Ansible provides so-called 'lookup plugins'[3], the available lookup targets are `file`, `url`, `password` and others. Said plugins can be added as a part of Collections. To list all plugins currently available `ansible-doc -t lookup -l` command can be used.

1.2.3 Collections

Ansible collections can be thought of as libraries, that contain modules, roles, playbooks and plugins which are specialised for the provisioning of a certain set of services (e.g. Netbox, iDrac). They are relatively old Ansible components however, they've become more prominent after `2.10` release[4] which split to 'core' modules into more collections such as `ansible.builtin` , `community.general` and `ansible.posix`. Third-party manufactures and developers can provide collections for their products. The most relevant for thesis are following core collections: `ansible.builtin`, `ansible.posix`, `community.general`. Collections that are not part of the 'core' are listed in `requirements.yml` (see Structure of Attachment).

1.2.4 Roles

Roles are YAML prescriptions for the provisioning process of one service (i.e. openssh-server) or a smaller stack of services (i.e. Netbox server). They provide a more abstract declarative interface in form of variables while keeping the particular order of steps that need to be made to achieve the desired state hidden inside themselves. Each role has a metadata section that specifies its dependencies, the supported target platforms as well as the maintainer. Even though all roles include default values (`defaults/` sub-directory), these values should almost always be changed or overridden before the execution of the role.

---

3. `https://docs.ansible.com/ansible/latest/plugins/lookup.html`
4. `https://www.ansible.com/blog/ansible-3.0.0-qa`

1.2.5 Plugins

Plugins allow for third party functionality to be added to Ansible. There are several different kinds of them, the most important type for this thesis being the lookup, caching and inventory plugins[5]. Inventory plugins allow dynamic creation of inventories from various databases (i.e. Netbox).

1.2.6 Playbooks

A playbook is the basic unit that is executed when provisioning a system via Ansible. Each playbook file can contain multiple 'plays', each 'play' is a list item in the root of YAML document as shown in Listing 2 and represents a set of tasks, roles that are applied to the specified inventory of hosts. It is possible for multiple playbooks to be executed against the same inventory. An ideal playbook, that deploys a service, should be idempotent, e.g. its second execution against the same host should yield only `OK` statuses and change nothing on the host.

```
---

- name: Basic Ansible play example
  hosts: an_inventory_group
  tasks:
  - name: print my hostname
    debug:
      msg: "{{ invetory_hostname }}"
  roles:
    - role: idrac
```

Listing 2: Basic ansible play example

1.2.7 Inventories

Inventory is one or more files with definitions of group(s) of hosts that can be provisioned and managed by Ansible. There are several different ways to define them, the most simple is `ini` configuration file, which is suitable for quick testing. More advanced inventories are written as YAML files and can utilise one or more plugins to pull hosts from remote asset management systems such as Netbox, in the environment of CSIRT-MU

5. `https://docs.ansible.com/ansible/latest/plugins/plugins.html`

state of hosts does not change on a day to day basis and therefore is safe to store in cache, details about caching are described at the end of this chapter. Hosts pulled that are part of dynamic inventory can be further manually grouped into different subgroups based on their attributes (i.e. hostname, device type or platform). Location of inventory definition is modifiable via `defaults.inventory` option in the `ansible.cfg` configuration file. For the purpose of this thesis, both static and dynamic inventories are utilised and all of their definitions are stored in `inventories/` sub-directory.

1.2.8 Vault

Ansible Vault is a component that can be used to encrypt files or values contained in YAML files with symmetric AES (*Advanced Encryption Standard* formerly named Rijndael) cipher-suit with 256 bits long key. This allows for to securely commit all configuration files to git repositories. Key for the Vault is contained in a file is usually omitted from the repository by `.gitignore`.

1.2.9 Facts

By default at the beginning of every playbook execution, `gather_facts` module is called, the results of its execution are variables that contain various information about the target system. All available information can be listed by executing `ansible -m setup somehost` in `ansible_playbook_dir`.

1.2.10 Miscellaneous

Additional parameters settings that can be specified in `ansible.cfg` (see chapter A) include caching of Facts and Inventories.

Caching of dynamic inventories leads to faster execution times of playbooks, allows for better debugging (inventory files are available in `./misc/inventories`) and reduces the number of requests that Netbox has to handle. Facts cache (stored in `misc/facts`) not only makes the execution of playbooks faster allows the system administrator to query them for additional information (see chapter 3.8 for examples). While in this thesis only `jsonfile` caching plugin is used, there are other options. A complete list of caching plugins that are available in a particular Ansible installation can be listed by `ansible-doc -t cache -l`.

While Ansible provides several ways to test the automation scripts, the most comprehensive one, Ansible Molecule[6] is not suitable for a proprietary

6. `https://github.com/ansible-community/molecule`

system that does not officially support it, such as Flowmon OS. The main is the reason is that it applies the automation scripts to a predefined virtual environment, this environment would in the case of Flowmon OS be a fresh VA (*Virtual Appliance*). VA do not necessarily behave in the same way as a physical network probe, since it has neither previous configuration and data nor iDrac interface, simulating the real problems/conditions would require effort disproportionate to the benefits.

Consequently, the testing process of Playbooks included in this thesis was limited to the following two-part process. To verify that syntax of playbooks and subordinate roles is correct: `ansible-playbook -syntax-check someplaybook.yml` is used. The `-dry-run` option is unusable in this case since it does not execute any of the tasks, including those that retrieve additional variables or facts, consequently the tasks that reference those would always fail.

Additionally, Ansible-lint[7], a tool that is not part of the default Ansible installation was used. It verifies that the resulting Ansible scripts adhere to best-practices and suggests possible improvements as well as fixes to common mistakes (i.e. using `when: previous_task.changed` instead of `notify: [ a_handler ]`).

Due to these limited testing options and further problems described in chapter 2, are tools for fully automatic Ansible deployment (e.g. Ansible AWX[8]) out of the scope of this thesis.

7. `https://github.com/ansible-community/ansible-lint`
8. `https://github.com/ansible/awx`

1.3 Technologies and Services

This chapter and its sub-chapters contain descriptions of services and protocols used for MI provisioning and by the MI itself. Figure 1.4 shows a simplified view of the interaction of these services with one device and one administrator. Please note that the unidirectional arrows originate from the subjects of their annotations and point towards the objects of said annotations. Each service description also contains an explanation of specifics that are introduced by Flowmon OS to the aforementioned standardized protocols.

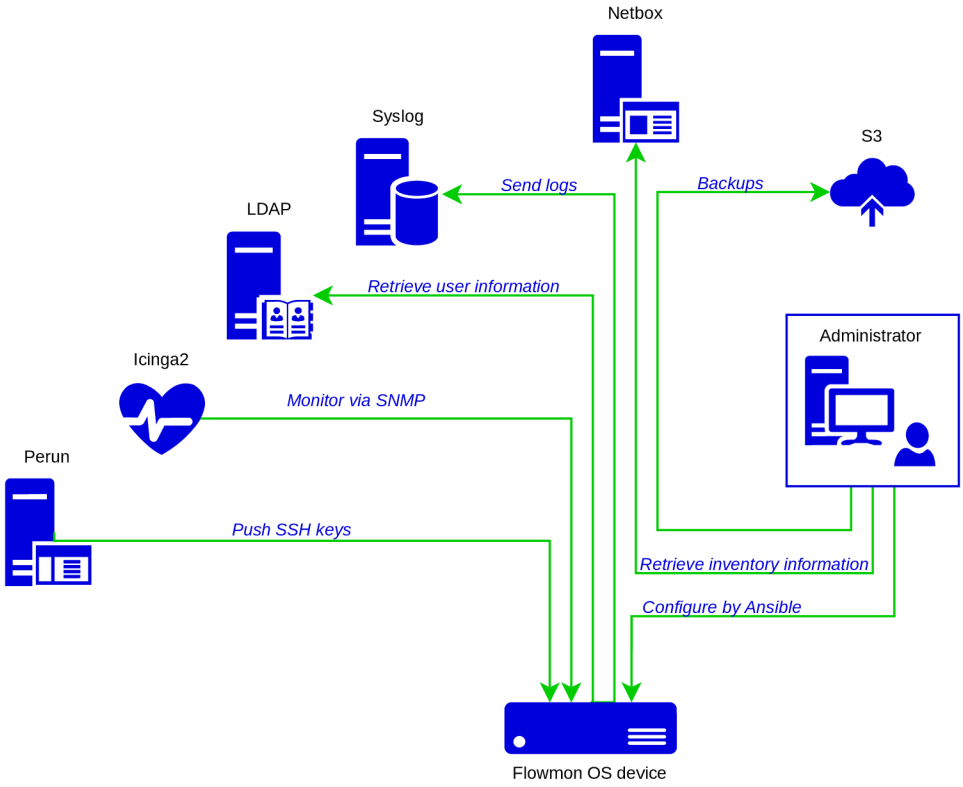

Figure 1.4: Overview of the service dependencies

1.3.1 LDAP

LDAP (Lightweight Directory Access Protocol) is a protocol that allows to access and manage user and other information on a directory server. Its latest version 3 is defined in RFC4511 [9].

Flowmon OS v11 allows for user accounts to be authenticated against an LDAP server, these user accounts work only for the web interface and are not mapped to the underlying operating system. Due to limitations of the used LDAP server, it is not possible to use advanced features such as user role mapping, therefore these features are not discussed any further. Moreover, there is a problem with Flowmon OS LDAP functionality itself, it does store hashed administrator user passwords locally. This, unfortunately, allows the users, that had their access removed by the LDAP server, to keep access to devices in question. Whether a user is an administrator is determined by the boolean value of `isAdmin` attribute, which can be inspected in the exported XML configuration file (more information in subsection 2.2.5), this value is set to `true` when a user has administrator rights to at least one of then modules (FMC, FCC, ADS). A feature request, suggesting that this behaviour should be changed, was sent to the Flowmon support team during the writing of this thesis, ETA for its implementation was not specified in the reply. The current workaround is described in chapter 3.8.

1.3.2 Perun

Perun[9] is an IDM (*Identity Management*) system, that controls SSH access to Flowmon devices at CSIRT-MU. It manages `~/.ssh/authorized_keys` file of `root` and `flowmon` users. In order for it to work, option `PermitRootLogin` in `/etc/ssh/sshd_config` needs to be set at least to `forced-commands-only`. The internal structure of users, facilities, resources is managed by my colleagues at CSIRT-MU and not in the scope of this work, provided values are discussed in chapter 3.2.1. Adding a new machine or changing the FQDN (*Fully Qualified Domain Name*) of an existing device requires manual modification of the host list in the web interface since Perun and needs to be done manually by the manager of the facility. SSH access to probe is not necessary for day-to-day operations, that is the main reason why it is managed by custom tools, separately from web users. While Perun has many different modules that allow management of other things than SSH, only `sshkeys-generic` and `sshkeys-root` modules are used for MI provisioning at CSIRT-MU.

9. `https://perun-aai.org`

1.3.3 Syslog

Syslog (*System Logging Protocol*) is a messaging protocol used for logging in UNIX-like systems [10]. There are two specifications that define the format of the messages, the older is RFC3164 [11] the newer one, which is also used by Flowmon OS, is RFC5224 [12]. Syslog protocol is implemented by many programs, the most common ones are `rsyslog`[10] and `syslog-ng`[11]. Both Flowmon OS and the central log collection endpoint at CSIRT-MU use `syslog-ng`. Although TLS encryption is supported both by the protocol and `syslog-ng`, it cannot be configured on Flowmon OS (see chapter 2.1.4 for more information).

The centralised log collection on MI was configured for the first time as a part of this thesis. The main reason for introducing it to MI was that some of the problems encountered resulted in an increased amount of logs, which were automatically rotated before they could be analysed. The second reason is that it allows to quickly verify which probes manifest the signs of the same problem, without the need to check each device either manually via SSH or by an Ansible ad-hoc command (see Additionals and Snippets). The central log collector is maintained by my colleagues at CSIRT-MU, who provided me with the credentials and other information necessary.

1.3.4 Netbox

Netbox is an open-source application that is designed to document networks and devices deployed in them. It is being developed and maintained by DigitalOcean, Inc.[12]. Netbox developers provide well documented and comprehensive Ansible collection[13] that can be used to add, remove and modify all attributes of the devices and networks stored there. This collection utilises RESTful API (*Representational State Transfer Application Programming Interface*) that is provided by each Netbox instance. At CSIRT-MU mainly its capabilities related to asset management are utilised since only certain segments of the MU network are under direct control of CSIRT-MU. For reasons that will become apparent, this thesis mainly benefits from the possibility of creating Ansible inventories based on data stored in Netbox (see chapter 3.1). This allows for quick execution of fact-gathering or diagnostic tasks without the need to keep a separate list of hosts and their state. Due to the fact that the production instance that is currently used by CSIRT-MU, all

10. `https://www.rsyslog.com`
11. `https://www.syslog-ng.com`
12. Cloud service and infrastructure provider from the USA founded in 2011.
13. `https://netbox-ansible-collection.readthedocs.io`

information stated about Netbox relates strictly to version `2.9.3`. Each MI device is added to Netbox upon its purchase, see Figure B.1 and Figure B.2 for further reference.

1.3.5 Icinga2

Icinga2[14] is the monitoring system used by CSIRT-MU, its development originally started as a fork of Nagios[15] in 2009. Icinga is used to check the availability of network services such as SSH, HTTP(S) and other information that is exposed via predefined SNMP (*Simple Network Management Protocol*) objects, all objects are listed in *Supported SNMP Objects - Flowmon v11.1*. Flowmon OS also allows to add custom objects, at CSIRT-MU this is used only marginally, details are described in chapter 3.2.3. As this thesis deals only with the provisioning of the MI, it is presumed that Icinga was already configured to match the configuration of addresses of the devices in MI.

1.3.6 S3

In the environment of Masaryk University, S3-compatible storage is provided by an in-house Ceph cluster, which is managed by another team outside of CSIRT-MU. This thesis aims to utilise it as secure storage of assets like configuration, licenses, certificates and keys. There are several Ansible collections that allow for the interaction with S3 storage, since Amazon.com, Inc. was the starting force behind the S3 protocol I have chosen their collections as the most suitable, particularly useful was the `amazon.aws.aws_s3` module. Before S3-compatible storage was chosen as the most fitting for this purpose, the following other options were considered.

Passbolt[16] is the current password manager at CSIRT-MU however, its API enforces the usage of GPG keys for authentication, which while being secure, results in making access to the API difficult. Moreover, it provides very limited structure, i.e. no folders, tags or any other way that could be used for predictable and machine-readable access to stored secrets. As of March 2021, there is no Ansible lookup plugin available and the reliance on GPG-based authentication makes it impossible to use `ansible.builtin.uri` module for password retrieval. Implementation of a custom plugin is out of the scope of this thesis since it would require a substantial amount of time to properly implement and would need further support and maintenance.

14. `https://icinga.com`
15. `https://www.nagios.org`
16. `https://www.passbolt.com`

Netbox, contrary to Passbolt provides a powerful API and each asset stored in has its own predictably name, (e.g. *'Flowmon Probe 20000 SFP+'* results in *'flowmon-probe-20000-sfp'*). However, during the course of this thesis, version 3.0.0 was released, this release removes the option to store secrets and storing any of the aforementioned assets as a custom attribute in the 'Config context' section is not advisable, since they contain confidential information.

2 Analysis and Design

Firstly, it is important to disclose, that the Flowmon OS does not differentiate between a probe and a collector, the software installed on the device is exactly the same as well as the chassis. The purpose of the Flowmon device is determined by its additional hardware components and software configuration, collectors have a large redundant (RAID6) disk array, while the probes have high-speed network cards (10G, 40G or 100G) [14]. Other system specifications such as CPU and memory remain in the same order of magnitude.

The process description as well as the following analysis and implementation focuses mainly on the system and flow collection configuration of Flowmon OS in non-distributed mode. Other components (i.e. DA and ADS) are mostly mentioned in situations where their presence in the specific environment of CSIRT-MU results in some kind of restriction for the other parts of MI. Additionally, they do not offer any other options of provisioning and their configuration changes frequently after they are provisioned therefore there is no real use in codifying it.

2.1 Provisioning Process

This chapter describes the manual provisioning process of a Flowmon probe. The same process can be used to provision a Flowmon collector, the only difference is that the 'Monitoring ports' section is to be left unchanged on the collector and one 'Listening port' with the corresponding profile have to be created for each subordinate probe (see chapter 2.1.3). The entire configuration process is divided into sub-chapters that follow.

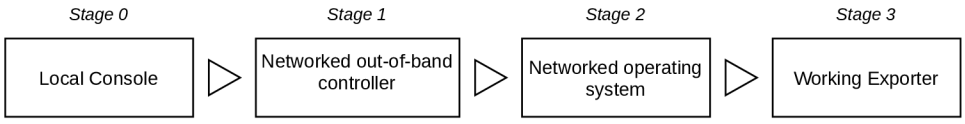

Figure 2.1: Provisioning stages

From the practical point of view of the system administrator, the provisioning process has four basic checkpoints as displayed on Figure 2.1. These checkpoints represent the completeness of the process and are going to be used as a reference for better orientation throughout the rest of the thesis.

Local console Probe is correctly wired, but not ready for any kind of remote access, on-site configuration is required for the out-of-band controller or OS (*Operating system*) to be networked.

Networked out-of-band controller OS and the firmware out-of-band controller itself can be completely re-installed or configured remotely without on-site presence.

Networked system System administrator can SSH directly to the probes OS without needing an out-of-band controller.

Working exporter `flowmonexp5` is running and has correctly set up targets and additional plugins (see chapter 2.4). Flow data is arriving at all specified targets. An analogy for a collector would be running `nfcapd` process, which is receiving flow data from all sources.

2.1.1 iDrac Configuration

The first step is to configure iDrac, this is done via local console because no other configuration option is available at the moment, the administrator can enter its configuration menu during the boot process, by pressing `Ctrl + E` when the following prompt is shown: *'Press CTRL + E for Remote Access Setup within 5 sec'*. No particular structure of the menu is described in this thesis since the versions of the controller used at CSIRT-MU vary widely and their navigation is intuitive for a person with some technical background. However, the following things need to be always changed or verified:

- Configuration of network parameters on the interface itself. Subnets are specific for each physical location, free IP address is found and assigned in Netbox. Network mask and gateway is selected in a similar way.
- Changing of the default password to `root` service account.
- Hostname is chosen accordingly to internal rule (hostname of probe management interface with `-idrac` suffix).
- DNS (*Domain Name System*) servers used by MI are *ns.muni.cz*, *ns2.muni.cz* which are used because they can be directly accessed from the private network, where MI is placed.
- NTP (*Network Time Protocol*) servers are the same as DNS servers.

2.1.2 System Configuration

The second step is system configuration. As mentioned in the following chapter there are several different ways to change the configuration of the

Flowmon OS device however, the only method that can control all officially supported options is FCC (*Flowmon Configuration Center*), therefore it is the current method of choice.

Management Interface (see Figure B.3), a typical Flowmon OS-based device has two management interfaces, in this case, only one interface is utilised as there is no need for a second network, the configuration is selected based on data from Netbox. The management interface is used for the following purposes: serving of web interface and API, access to supporting systems that are configured in the following steps, transport of flow records to remote collectors and retrieval of update packages.

DNS Servers (see Figure B.4), in the environment of CSIRT-MU the same servers as in iDrac are used. It needs to be taken into account, that given DNS servers need to be accessible from the private network in which is the device deployed, which along with security concerns disallows the usage of public DNS servers.

Hostname (see Figure B.5), it is entered in two parts: hostname of the device in question and the domain to which this device belongs. There are no known caveats or special requirements. In larger deployments, it is advisable to have a rule for choosing the hostname, which takes into account both the physical location of the probe and the network links, which it monitors (e.g. `flowmon-LOCATION-LINK.someorg.example`).

Time zone (see Figure B.6), each flow record contains two timestamps (as discussed in chapter 1.1.2), which are important for further analysis and detection methods. Therefore it is important that all devices that export flow records have the correct timezone and accurate time. Choice of NTP (*Network Time Protocol*) servers at CSIRT-MU is the same as the one used for iDrac Role.

Email (see Figure B.7), the chosen SMTP server is used for the three following purposes. Firstly notification to administrator(s), while the manufacturer's documentation does not include a comprehensive list of all possible alerts and their triggers/causes, the following are among the most common ones: inaccessible exporter target, license expiration and a significant decrease in monitored data. Secondly alerts from Network Collector detection methods. Thirdly the delivery of FMD (*Flowmon Dashboard*) reports, which are part of FMC that is not actively used at CSIRT-MU and is therefore not discussed any further.

Proxy (see Figure B.8), devices that are part of MI are usually placed private network that is not directly routed to the Internet, usage of HTTP (*Hypertext Transfer Protocol*) proxy allows these devices to check and download update packages from manufacturers servers.

Syslog (see Figure B.9) configures the forwarding of the system logs to a remote Syslog target. Caveats such as the exclusion of `flowmonexp5` are described in detail as a part of chapter 1.3.3.

LDAP settings are not captured or displayed in this thesis since they are not relevant for the automation of the provisioning process and would need to be completely censored for security reasons. Due to external factors, no user role mapping is utilised in the environment at CSIRT-MU.

2.1.3 Flow Configuration

The third step of the process covers configuration of flow export and flow collection, which is the primary functionality of MI.

Flow export setup consists of only one section, that section being the configuration of the exporter itself, the relevant menu is labelled as *'FCC > Monitoring Ports'*. It consists of the 'Global settings' tab and several 'Monitoring port' tabs (see Figure 2.2), the exact count of the latter type depends on the count of monitoring interfaces of the particular probe. Devices that are only licensed as collectors do not have this item in the FCC menu.

At CSIRT-MU exclusively IPFIX format is used, therefore no change in default settings of "Export protocol" are made or discussed (true for both global and per-interface variants of this setting). However, flow data transport is configured in several different ways due to the reasons outlined in the following paragraphs. The prevalent flow transport option is TCP with TLS encryption enabled, this combination is shown in Figure 2.3.

Change in any of the shown options, followed by clicking the 'Save' button results in the restart of all `flowmonexp5` processes on that monitoring interface which usually results in a small loss of data, from the interval before the `flowmonexp5` starts again. The global configuration is only used for the configuration of the optional plugins that add custom IPFIX fields, the options shown in Figure 2.4 are common to all probes and have been agreed upon by IH (*Incident Handling*) team at CSIRT-MU, Flowmon collectors are configured by default to store these attributes.

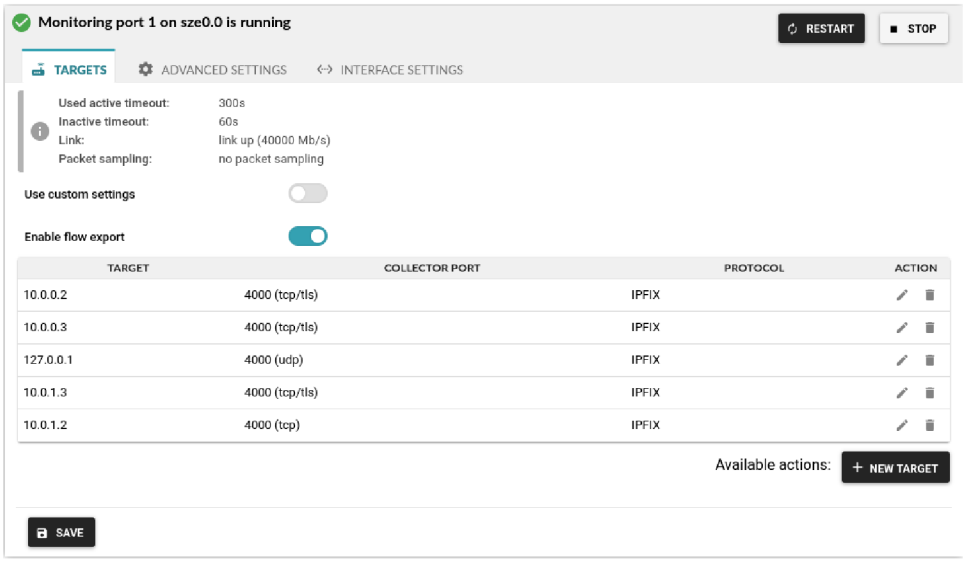

Figure 2.2: Exporter targets of one monitoring port in FCC

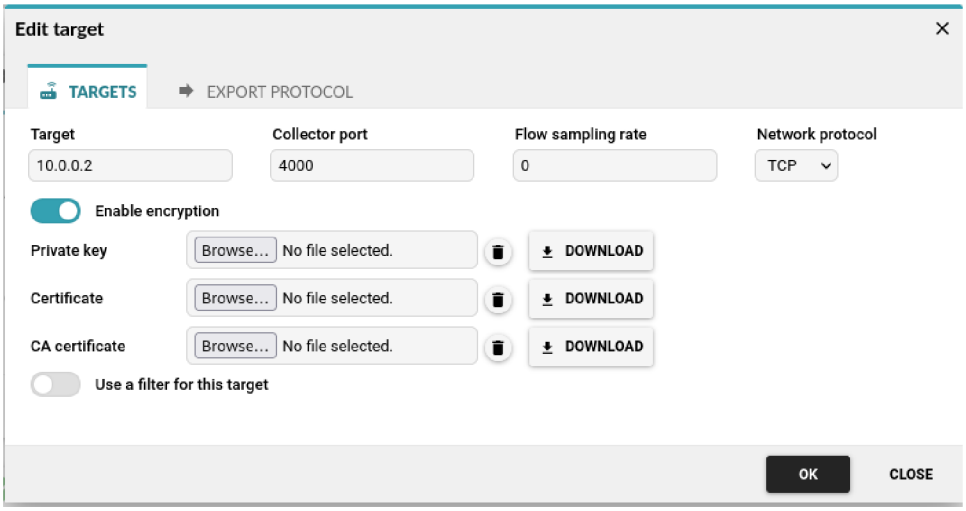

Figure 2.3: Configuration of one exporter target in FCC

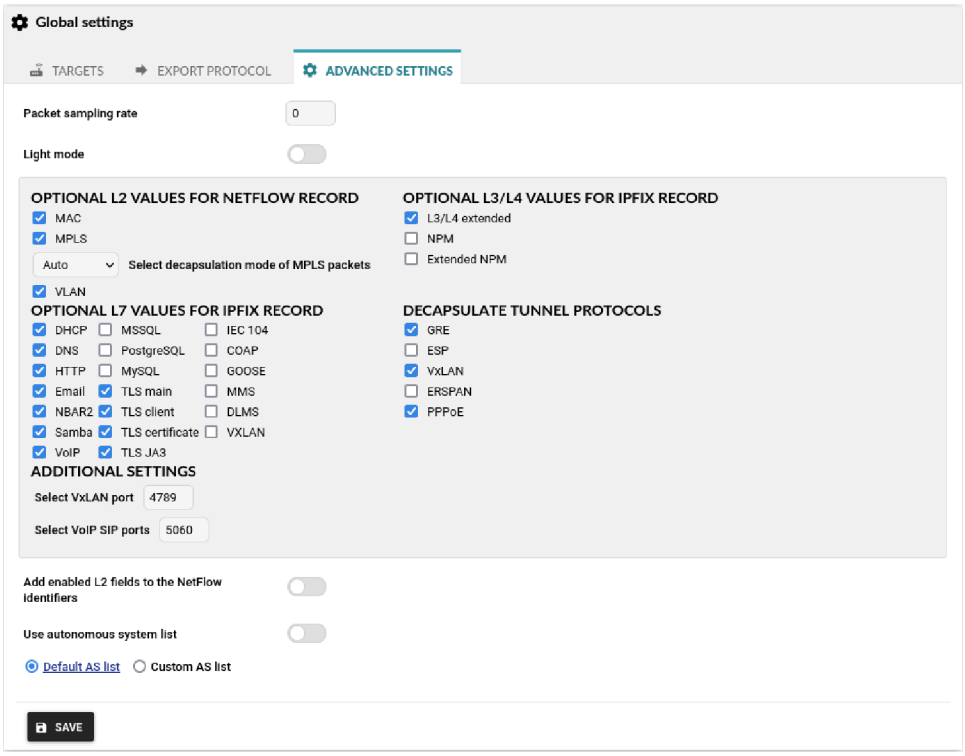

Figure 2.4: Exporter advanced options in FCC

There are two parts to the configuration of flow collection, the first one being *'FCC > System > FMC settings > Listening ports'*. As was previously stated, the collector is to be configured both locally on the probe and remotely on the central collector. Two main differences are that the local collector will be configured to use UDP for IPFIX transport as it needs fewer resources and serves as a baseline, which can be used to verify that no data is lost during the communication between the probe and the collector. The remote collector is be configured to use TCP with TLS. This choice was made with the intention to increase the security of transported flow data as well as the reliability of transport.

The second is *'FMC > Profiles'*, each new flow source needs to have manually enabled profiling. 'profiling' means that an automatic profile (`Sources/name_of_flow_source`) is created in FMC. These automatic profiles can be aggregated and have filter expressions applied to them. Filter expressions are evaluated via `nfdump` consequently the same syntax is used, according to the manual page of `nfdump`, it is similar to `libpcap` syntax in

more commonly known `tcpdump`[1], complete syntax reference is in Chapter 5 of User Guide which is present on each Flowmon OS device[2].

2.1.4 Custom Configuration

The fourth stage of the provisioning process of Flowmon OS-based devices at CSIRT-MU includes several steps that are not part of the official documentation, but many of them were suggested by the Flowmon support team as workarounds for features that have not been implemented yet. This section is precluded by upload of a custom package to FCC, this GPG (*GNU Privacy*) signed package adds SSH (*Secure Shell*) public keys of selected employees of CSIRT-MU to `/root/.ssh/.authorized_keys`. Without this modification, some of the following steps would be possible, as the `flowmon` user account is supposedly only able to perform the actions listed in *Flowmon Solution Maintenance* [15].

All of the following modifications were previously implemented manually or via various BASH scripts, while the process of applying said modifications was rewritten in Ansible, the assets that are being deployed were left mostly unmodified.

The first step is to install components that allow centralised management of SSH keys. Since the package mentioned in the first paragraph includes only people that take part in the provisioning and maintenance of MI, other users are managed via Perun. The original manual process of configuration will not be described here, since it is not applicable to Flowmon OS version 11, Ansible role is described in chapter 3.2.1.

The second step is an addition of a custom SNMP check. The script and its dependencies itself were adopted from the previous configuration which was created by Bc. Tomáš Plesník, its migration from NRPE (*Nagios Remote Plugin Executor*) to SNMP was done by my colleague Bc. Martin Gregorík. Its deployment was automatised with an Ansible role, which is shown in chapter 3.2.3.

The last step of the deployment process is to change the default passwords to both `flowmon` and `root` user account. This is done by `passwd` utility or `ansible.builtin.user` module.

1. `https://www.tcpdump.org/manpages/tcpdump.1.html`
2. This manual cannot be referenced by a publicly available link.

2.2 Provisioning Options

Based on a rigorous review of all official documentation available at Flowmon support portal[3] and long-term practical experience, it is possible to say, that there are seven ways to provide or export configuration to or from a device running Flowmon OS. Before the full analysis of each of them, it is important to state, that Flowmon provides backwards compatibility of exported XML configuration and API interface only for each of the major releases (e.g. 9, 10, 11). Therefore, everything listed here is completely true only for the current release, Flowmon OS version 11.1.

2.2.1 FCC

As has been already shown FCC (*Flowmon Configuration Center*) is a part of the web interface of each Flowmon OS-based device, it allows for the configuration of all officially supported settings. The only exceptions that need to be changed via console are system user passwords, the configuration of the optional out-of-band controller and custom patches to configuration files. This makes it by far the most complete configuration interface that the Flowmon OS offers. However it is not possible to automate it in any reasonable way, therefore it does not require any further debate.

2.2.2 API

On versions of Flowmon OS prior to `11.01` API documentation was published as a PDF (*Portable Document Format*) file on the above-mentioned support portal. Since the latest release, there is OpenAPI version 3[4] compliant swagger interface available on path `/restapi-doc/fos/` of each devices web interface. Nevertheless, it still does not support the modification of the options from FCC, which are an inseparable part of the Provisioning Process. Operations that are currently supported by the API can be divided into the following categories based on their use-cases from the point of view of this thesis.

Information retrieval, these endpoints allow to get CPU, RAM, current disk space usage as well as exporter targets and states. Unfortunately, the API access has the authentication done via OAuth 2.0 bearer tokens[5] which can be only created with valid user credentials on `/resources/oauth/token` path and have expiration immutably set to 24 hours after their creation. This along

3. `https://portal.flowmon.com`
4. `https://swagger.io/specification/`
5. `https://datatracker.ietf.org/doc/html/rfc6750`

with the fact that there is no possibility of limiting a token to read-only access to specific API scope, prevents the usage in monitoring systems, for example, Icinga2.

User management can be fully done via methods that have already been implemented. However, it is not desirable to manually add or delete users using this API in a permanent deployment of this scale, as a centralised user database such as LDAP is much more suitable.

Analysis, the majority of the endpoints belongs to this category, their functionality is predominantly related to FMC and FMD. This makes them irrelevant to the provisioning process since the analysis and configuration of alerts, dashboards, etc... is neither in the scope of this thesis nor the responsibility of the infrastructure team at CSIRT-MU.

Provisioning, the only relevant thing that can be provisioned via API are channels[6] though, without the option to configure 'Listening ports' in a similar manner, the practical usefulness of automating the creation and management of said channels is severely limited if non-existent.

2.2.3 Sysconfig

Sysconfig is aconfiguration util, that can be used in the console of Flowmon OS-based devices. It is merely a BASH script that modifies system configuration files containing things such as hostname, NTP servers and others (see The interface of the sysconfig utility). Based on its source code (found in `/root/sysconfig`) it is possible to ascertain that it mainly relies on selecting relevant lines of configuration files via `*grep` and then replacing their contents via a `sed` regular expression. Selectors that are used are not documented or subjects to any guarantee of backward compatibility. This does make any attempt on doing the same thing via Ansible modules impractical if not impossible.

On the other hand, its usage for first-time provisioning of a new Flowmon OS is directly suggested in the short quick-start guide that is packed with each device[7]. It provides a subset of functionality that is a part of FCC however, it is predominantly focused on bootstrapping a device to the point where it can be deployed to a remote location (see 'Stage 0' in Provisioning stages). While it bypasses the problem that is posed by lack of iDrac or its equivalent

6. Channel is an abstract group of one or more sources from 'Listening ports' with optional filter expression applied.
7. Unfortunately a digital copy does not exist and the guide itself is not available in any library.

in cheaper models of Flowmon OS-based probes [14], it does not help with flow exporter or collection configuration.

Based on these findings it is safe to conclude that it is unsuitable for any kind of automation since its interface is optimised for human interaction and cannot be reasonably scripted.

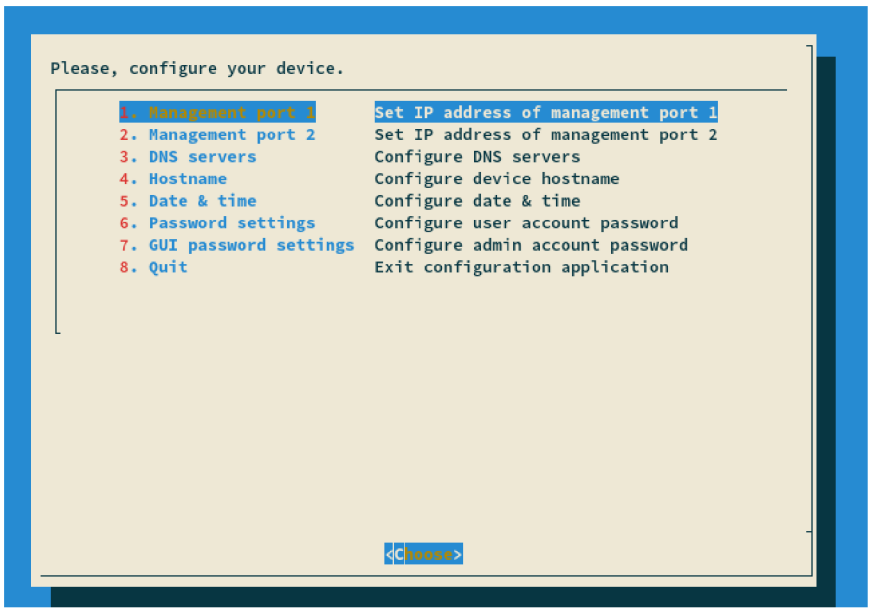

Figure 2.5: The interface of the sysconfig utility

2.2.4 Index.php

As the name suggests, this program exposes several methods, which are used by the FCC web interface, as command-line tools. Some of the methods that are provided are documented in the *Flowmon Solution Maintenance*, all of the methods that are meant to be used by the administrator can be listed via `php /var/www/shtml/index.php Cli:list`. In cases where `index.php` and API have overlapping functionality, API is generally the preferred option, since it provides structured and well-documented output, above mentioned duplicities are listed but not discussed any further. There are four classes of methods: `FmdModuleJobManagerCliPresenter`, `InstallCliPresenter`, `DaCliPresenter` and `CliPresenter`. The first two provide no command-line functionality, the third is only relevant on the devices that are part of DA, the fourth one is by far the most useful one. Still only selected methods

from this class will be analysed further, since the others are either read-only, undocumented or irrelevant to the provisioning process.

`SetConfigurationXML` was provided in previous versions of Flowmon OS and its documentation (version 10). In the current (version 11.1) it was removed from both the maintenance [15] and the recovery manuals [16], but remains available and is being utilised by the Flowmon support team.

`GetConfigurationXML` was recently removed from the official maintenance guide [15] as well as help output in the command-line tool itself. The function is not likely to be entirely removed, since the analysis of its code on a live device revealed that it is merely a command-line wrapper for backend functions used by the web interface.

`DownloadUpdatePackages` method downloads all packages and plugin updates that are available for the device with its current license.

`SyncTimeNTP` method forces immediate synchronization of time using current NTP settings in FCC.

`GetEmailSettings` retrieves current email settings from FCC, useful for verification that the sender is set to a correct value (mistakes tend to be common due to changes in probes assigned location).

`DeleteUser` duplicates an API endpoint `DELETE /rest/fcc/users/{id}`.

`GetExporters` duplicates API endpoint `GET /rest/fcc/exporters`.

`ChangePassword` is duplicitous with API endpoint `PUT /rest/fcc/users`. While the API documentation in version `11.01.07` (see chapter 2.2.2) does not list 'Password' as a supported attribute, it is possible to change it, a bug report was sent to the Flowmon support team as a result of this discovery.

`GetAllUsers` is duplicitous with an API endpoint and not included in the above-mentioned list, but is utilizable in ad-hoc commands since does require additional authentication.

2.2.5 XML Configuration

As the methods in the previous chapter suggest, each Flowmon OS device allows for the import and export of the entire configuration of all modules

into one XML (*Extensible Markup Language*) file, this can be done via methods shown in the previous chapter. The complete structure of said XML file is not subject to any part of the official documentation, the only existing resource, other than the file itself, is *Profile Import Using XML*.

Theoretically, the import of a customised configuration file should allow for automation of the entire Provisioning Process. However, the aforementioned lack of documented schema and the fact that the official documentation explicitly discourages modification of any other section than 'Profiles' ('We strongly recommend not to modify any other sections but Profiles [17]'), makes the usage of configuration import for the purpose of provisioning a new device impossible. Furthermore, each configuration file contains section(s) that are highly specific of its origins, such as licenses, modification timestamps and TLS key-pairs. Also, the individual sections of the configuration file are dependent on each other, while these dependencies are not artificial (i.e. ADS data feed needs the FMC profiles, which it includes), it makes it impossible to migrate configuration of only one module (i.e. ADS) without overriding or amending existing configuration of the other ones. For example, the import of *'Flowmon ADS > Email reports'* is dependent on *'Configuration Center > System - Users'*, but users have pulled from LDAP and therefore should not be manually overridden.

2.2.6 Official Backups

The chapter addresses the configuration and usefulness of backups to 'External storage' which were introduced in release `11.0.05` [15]. These backups are not universally supported yet and have certain disadvantages such as that the external storage only supports SMB (*Server Message Block*) protocol. Furthermore, flow data are an inseparable part of the backups, which would understandable and appropriate for the instances of Flowmon OS that serve as collectors, but it is completely unnecessary for probes, which in the majority of cases store just the most recent data, as they only have one terabyte solid-state disks [14]. Furthermore, in the environment of CSIRT-MU and other middle to large scale deployments, probes are usually configured to send flow data to more than one collector. Therefore creating a backup of data stored in a built-in collector is unnecessary.

As stated in *Flowmon Solution Maintenance* [15] restoring probe configuration and data of Flowmon OS appliance using this method requires already partially restored settings, so that the appliance is networked ('Stage 2' from Provisioning stages), has access to the 'External storage' endpoint where the backup is stored and has a valid license.

Additionally, in order to store the initial backup of all 24 currently active probes in the production environment of CSIRT-MU, up to 24 terabytes of storage would be required. Another approximately 8 to 10 terabytes of storage would be required for the initial backup of each of the production collectors. This estimation doesn't account for any redundancy of the backup storage and the exact numbers are dependent on quotas settings in *'FCC > Resource'*. A further complication is posed by the fact that flow data backup cannot be automatically rotated and deleted after a certain time interval. As per *Flowmon Solution Maintenance* [15] only configuration and database data are subject to automatic expiry.

Because of these problems, it was decided that the backup via XML configuration export is the most fitting and sufficient solution for the time being.

2.2.7 iDrac

All of the physical Flowmon OS-based devices that are deployed at CSIRT-MU are based on chassis manufactured by Dell Inc. and have iDrac (*Integrated Dell Remote Access Controller*) installed [14].

Dell provides an official Ansible Collection for interactions with several of their remote management modules, it is called 'OpenManage'. The aforementioned collection can be used for the updates of firmware and configuration of the iDrac interface. The practical usefulness of these features in the environment of CSIRT-MU is disputable since, in order to be able to use it, one has to have the management interface already connected to the network (see 'Stage 1' in Provisioning stages). However, it is useful for verification of the non-critical settings such as NTP and DNS. The re-installation of the host operating system is by far the most useful feature available for purposes outlined in this thesis. It requires the ISO (*Optical Disk Image*) to be exposed on an NFS or CIFS network share, while this adds more complexity to the setup, it can prove itself useful in cases where the workstation of administrator is connected via a slow or unreliable network (e.g. when on the home office).

2.3 Evaluation

Based on the options listed in chapter 2.2 and the needs outlined in chapter 2.1, I conclude that it is impossible to create a zero-touch deployment scenario, which would be suitable for reality and continue to work on future releases of Flowmon OS. The biggest obstacles to fully automated deployment are the lack of a programmable interface that could configure flow export or

flow collection and the fact that the initial configuration of Flowmon OS networking needs to be done manually. However, it is possible and useful to automate the following sub-processes of provisioning and MI lifecycle management.

Recovery, for when the operating system of a device is damaged and needs to be restored to its factory state, in from 'Stage 2' to 'Stage 3' (see Figure 2.1), note that it is a different process than 'restore' which just restores Flowmon OS configuration from the XML file.

Deployment, as has been already stated, it is not possible to automate the entire process, but it is possible to automate large parts of the first and fourth step of the Provisioning Process. More details are in chapter 2.1.4 and chapter 3.2.

Facts, this use-case is not a part of the above-described provisioning process, but it significantly contributes to the effort of keeping rather large property evidence up-to-date. It also helps during annual license renewal or manual verification of the current state of the MI.

Backup, because of the scale and dynamicity, the infrastructure team at CSIRT-MU frequently encounters crashes, having several versions of configuration at hand, significantly reduces the amount of time and work that is required to remedy these outages.

Restore, in case of migrating a device with an undamaged Flowmon OS-based device to another role/location or immediately after a recovery has been performed. As per *Flowmon Solution Maintenance* only license export can be performed in this automatable way.

Exporter, adding a custom exporter process, while this is not a part of the previously described Provisioning Process and it is not officially supported. It is occasionally needed at CSIRT-MU and unlike the main provisioning process, it provides an option for me to demonstrate, how can something be fully automated with Ansible.

As outlined in the introduction of each Ansible component, dynamic inventories are also useful for ad-hoc commands, which due to the manual nature of the provisioning and maintenance are an important tool, these snippets are also included in the results even though they are not written in pure Ansible.

3 Automation

Before executing any of the playbooks or ad-hoc commands, use BASH script `utils/install.sh` or equivalent commands to install all the dependencies. All of the playbooks that follow are configured to use the same HTTP proxy shown in Figure B.8.

3.1 Inventory

As already said in chapter 1.3.4, the main point of interest in Netbox lies in the `Device` entity. Upon inventory creation, devices from Netbox are divided into four distinct groups, the first division is done by device names, where all with the name matching the `flowmon-.*` pattern are considered to be probes and `collector-.*` collectors. Both probes and collectors are divided by their state into *Active* and *Staged* sub-groups, resulting in `flowmon_(active|staged)_(probes|collectors)` inventory groups. Devices with any other 'Status' value should not be connected to network, therefore their presence in the inventory is not required. The current inventory can be listed via command shown in Listing 8, attributes that are included in the inventory are shown in Table 3.1

The majority of Ansible variables in this work is provided at the inventory level either in the inventory file itself or as `group_vars/host_vars`. While variables in Ansible can be provided at almost any level (i.e. tasks, roles, plays, imports), the inventory is the most suitable one, since it allows to easily override their values on a per-host basis while keeping each unique value written only in one place.

Table 3.1: Netbox attributes exposed as variables in the inventory

Netbox attribute	Ansible variable
primary_ip.address	ansible_host
display_name	inventory_hostname
status.value	status
platform.value	platform
primary_ip	primary_net

3.2 Deploy Playbook

This playbook handles the things described in the first and fourth steps of the chapter 2.1 with the addition of a patch to `syslog-ng` configuration. The playbook consists of one play and utilises three independent roles and some additional tasks. The target inventory is all probes and collectors that are marked in Netbox as active and not a part of DA.

Functionality that is described in the fourth step (subsection 2.1.4), was decomposed into 'perun' and 'monitoring' roles along with several sets of independent task files. The functionality from the first step (subsection 2.1.1), is done by a standalone role.

As a consequence of the fact that SSH is used for the configuration and communication of the distributed ADS (*Anomaly detection system*) and the documentation at support website[1] does not include further explanation of the inner logic of this process. It is not possible to manage SSH keys of DA devices via Perun as on other devices. This issue is resolved by the explicit exclusion of the static `flowmon_collectors_da` group in the play.

3.2.1 Perun Role

This role is heavily inspired by a preexisting role written by Kamil Andoniadis, it was rewritten to comply with Ansible version `2.10` and be compatible with Flowmon OS as well as CentOS 7. It was also extended by a script that stores BASH history of each SSH key managed by Perun Role in a separate file. The aforementioned scripted was written by Marián Rehák and Mgr. Martin Macháč.

The role performs the following step, firstly it adds an additional repository from which the Perun-related packages are installed. Secondly, the SSH public added to the `authorized_keys` file of the `root` user and configuration of the `sshd` is modified according to the requirements of Perun. Lastly, the configuration and BASH history script is stored.

The information needed for configuration was provided by my colleagues and includes two separate so-called facilities for collectors and probes as well as the address and SSH public key of the Perun Role instance used by CSIRT-MU. Due to the obvious security concerns, this information is not included in the attachment.

1. portal.flowmon.com

```
perun_facility: "MI_Probes" # MI_Collectors
perun_user: flowmon

perun_slaves: []
perun_default_slaves:
  - perun-slave-base
  - perun-slave-process-sshkeys-generic
  - perun-slave-process-sshkeys-root

perun_server:
  host: hostname-of-the-perun-instance
  key: public-ssh-key-of-the-perun-instance
```

Listing 3: Default variables of the Perun role

3.2.2 iDrac Role

This role automates the first step of the provisioning process that was introduced in the chapter 2.1. The tasks that are described in the process are automated using `dellemc.openmanage` Ansible collection. The role itself does not retrieve any information from the Netbox instance, rather it expects all required values to be supplied via variables (see Listing 4). Although the above-mentioned process includes the network configuration of the iDrac interface, the role does not change it, since in order to be able to connect to the iDrac interface, the network needs to be already configured.

It is important to be aware of the fact that this role can cause the loss of access to the out-of-band controller if configured improperly. Generally, it is ill-advised to remotely change iDrac configuration of multiple devices at the same time, without close supervision. This is because the iDrac is the last resort before having to physically go to the server room (see 'Stage 0' in Figure 2.1). Moreover, based on personal experience it is exceptionally rare for a device to require a remote change of its iDrac during its production lifespan of 5 years[2].

The modules utilized in this role require `share_name` argument to be specified for unknown reasons. The argument can be either a network share or local path, since the usefulness is debatable the value is always set to `playbook_dir`.

2. Planned production lifespan of a Flowmon device at CSIRT-MU is 5 years, because after that Dell no longer provides NBD (*Next Business Day*) hardware repairs.

Because of this issue, the role is tagged with `never` which disables them unless tag `idrac` is explicitly added during playbook execution, i.e. `ansible-playbook -tags=idrac deploy.yml`.

```
idrac_connection:
  ip: "192.168.1.1"
  user: root
  password: root

idrac_hostname:
  name: flowmon-abc-xyz-idrac
  domain: mgmt.example.com

idrac_new_pass: root
idrac_dns: &ns-muni [ "147.251.4.33", "147.251.6.10" ]
idrac_ntp: *ns-muni
```

Listing 4: Default variables of the iDrac role

3.2.3 Monitoring Role

This role is not required for a basic degree of monitoring functionality on Flowmon OS, but its modifications improve the usefulness of the existing features. It applies the two patches that have already been introduced in chapter 1.3.5 and chapter 1.3.3. The role does not accept or require any variables, since there are no customizable aspects to the deployment of both patches.

The first patch is the additional SNMP check named `check_nfcapd.sh` added to `/etc/snmp/scripts`. It was included mainly for legacy and continuity reasons. It counts flow records that are stored in `/data`, as shown on Listing 1. The deployment task filters all available sub-directories with flows and adds to check only those that have been modified in the last twelve hours. This prevents the check from failing because of orphaned directories that are no longer active but have not been deleted yet.

The second patch, fixes one of the two major setbacks were encountered during the deployment of centralized log collection. The problem is that the logs produced by `flowmonexp5` processes as well as other crucial parts of Flowmon OS are not forwarded by default. This fact was verified by the Flowmon support team and is not mentioned in the User Manual that is available in each devices web interface. All affected log files can be listed via the follow-

ing command: `ls -l /data/components/*/logs`, the results also include various ADS and collector components. Considering the importance of said processes, Flowmon support team was consulted and they recommended adding the shown in Listing 5 to the existing content of `/etc/syslog-ng.d/`. This advice is in strong contrast to their policy, which says that their devices are 'appliances' and should not have such modifications applied to them. The support team also accepted a feature request to have this behaviour changed in the default configuration. The patch adds only logs of `flowmonexp5`, others were not deemed useful in the specific environments of CSIRT-MU. The second setback is that Flowmon OS does currently support neither TLS encryption of Syslog traffic nor TLS client authentication. This has been verified by the Flowmon support team and consequently resulted in a feature request, which might be implemented in the Flowmon OS 12. Due to this limitation, IP whitelist is the only form of access control that could have been implemented on side of the centralised log collector.

```
source s_flowmonexp {
  file("/data/components/flowmonexp/log/flowmonexp.log"
        follow-freq(1));
};

log {
  source(s_flowmonexp);
  destination(d_remote);
};
```

Listing 5: Syslog-ng configuration patch

3.3 Facts Playbook

The purpose of this playbook and the included role of the same name is to retrieve information that can be used for lifecycle related operations, such as purchase planning, property database updates, license renewal and auditing of the MI. Unlike the inventory and updates the records in Netbox or stores the raw retrieved information in `misc/info`. Either of the previously mentioned options can be disabled by `--skip-tags=online` or `--skip-tags=offline` parameter.

```
{
  "hwid": ".serial-number.random.string",
  "targets": [
    {
      "id": 1,
      "exportProtocol": "ipfix",
      "ip": "147.251.14.53",
      "port": "4007"
    }
  ]
}
```

Listing 6: Structure of the additional facts

Although much of the wanted information can be retrieved by multiple methods that are described in chapter 2.2, the chosen policy was to use Facts whenever possible, and API in the other cases.

Ansible module `ansible.builtin.uri`, that is used to interact with the API, is configured not to validate certificates of the API endpoints, since the network probes use self-signed TLS certificates. Trusted certificates cannot be used due to the fact that both probes and their DNS records are not exposed to the Internet, therefore certification authority cannot verify them. Everything gathered is stored in Netbox as 'Local Config Context' attribute of each device, the structure of this attribute is shown in Listing 6. Additionally to the previously mentioned patch, device 'platform' attribute is modified. Note that this modification expects a platform with predefined 'slug' to be already present in Netbox, the main purpose of is to keep track of deployed Flowmon OS versions without excessive manual labour.

This information can either be viewed as any other data in Netbox or queried from cache (see chapter 1.2.10), examples of macros that use the cached facts can be seen in Listing 8.

The tag `override` has to be used to upload the retrieved information, the role itself just retrieves it, additional tasks in the playbook upload to given Netbox instance. Tasks that are performing the interactions with the API are executed on administrators workstation instead of the remote host, via usage of `delegate_to: localhost` directive.

3.4 Backup Playbook

This playbook creates a backup of the license and configuration of all active devices in the MI. Object versioning is enabled on the bucket where the backup is stored, server-side encryption that is a part of some S3 services, is not supported by this particular Ceph instance. Due to the sensitive nature of the contents of the configuration file, it needs to be encrypted before it is uploaded to S3. After extensive research, I was not able to find any Ansible or Jinja2 plugin that would allow to encrypt a string or file. Because of this a workaround shown in Listing 7 was implemented. In order to ascertain that the workaround is functioning correctly `SHA-256` are generated before encryption and verified before import.

The license is included in the XML configuration file, but because of convenience, it is also exported separately. While the license can be exported just by copying `/etc/flowmon/license` file, it documentation does not state if can be imported in the same manner, the exported license file can be imported via FCC.

```
- name: encrypt the configuration
  ansible.builtin.command: >
    gpg
      --armor
      --symmetric
      --batch --yes
      --cipher-algo AES256
      --passphrase {{ gpg_pass }}
  changed_when: no
  no_log: yes
  args:
    stdin: "{{ unencrypted_input }}"
  register: encrypted_results
```

Listing 7: Configuration encryption task

3.5 Restore Playbook

Restore playbook performs the steps as the Backup Playbook, but in reversed order, the encryption and decryption process of both playbooks is done on each respective device to minimize the exposure of the sensitive configu-

ration. The only addition is that it initiates the restart of the device. Once the configuration is retrieved the user is asked whether he wants to proceed by `ansible.builtin.pause` module. The playbook comes with a similar limitation as the official backups have, the device needs to be networked (see chapter 2.2.6 for more information) before the restoration can begin. Once the configuration was imported and the device restarted the success of the operation is verified by an HTTP request to `/rest/version` endpoint.

3.6 Recovery Playbook

This playbook performs complete system recovery as described in *Flowmon Recovery Guide* [16]. At the beginning of the execution, there are 60 seconds of delay during which the administrator is informed that the process will completely wipe the device if it is not canceled. The only limitation to the full automation of this process is that the Ansible collection that is used for interaction with iDrac is operating system agnostic, and therefore does not allow to login and execute commands in the system console automatically. As a consequence of there is no option to configure the network interface automatically, thus at the end of its execution a connection string is provided, it is meant to be manually pasted into the `/etc/sysconfig/network-scripts/ifcfg-eth0` file, via iDrac or local console. The information need to create this string is retrieved during inventory creation. Based on the practical experience, this configuration file is safe to be modified without causing any failure of the official configuration tools (an exception to the conclusion made in 2.2.3). The playbook utilizes `dell.openmanage.os_deployment` module which unlike the modules used in iDrac Role role requires the values of `share_*` variables to point to a NFS or CIFS version 2 network share with the recovery disk image.

3.7 Exporter Playbook

This playbook sets up a secondary exporter process with customisable configuration. There are two pre-configured plays in the exporter playbook that work on Netcope network cards (see explanation in the following paragraph), each of them deploying one of the two examples on separate inventory. The configuration of the `flowmonexp5` process itself can be easily modified by extending or overriding the `exporter_config` list variable and adding `--skip-tags=override` parameter during execution. The customization can allow the playbook to work on other network card models.

The first example adds a plugin that exports the version of QUIC (*Quick UDP Internet Connections*), which is being developed by Flowmon and was deployed at MUNI for testing. The plugin adds a custom IPFIX field to the records. This field is not supported by Flowmon Collector, therefore the exporter target is a special collector instance, which was deployed outside of this thesis. Due to licensing issues, to code of the plugin itself was replaced with a placeholder and is not part of the attachment.

The second example is an exporter instance that exports biflow instead of uniflow[3]. Due to the fact that Flowmon does not support officially support biflow export, it can only be deployed on probes that allow all channels to be joined and monitored by one `flowmonexp5` process, at CSIRT-MU this restricts the selection to network probes with 2-port 100G Netcope[4] network interface. This play only executes `biflow` role against `flowmon_biflow_probes` inventory group which contains all probes that are eligible for biflow export. The aforementioned inventory group is currently static since probes with compatible network interface configuration are no longer being sold and thus the inventory is not expected to change in the future. Information on how to modify the configuration of the Netcope card was provided to me by RNDr. Petr Velan, Ph.D..

3.8 Additionals and Snippets

What follows is a list of macros and scripts that do not fit the traditional layout of Ansible playbooks or roles, but can be considered one of the more important improvements to day to day operation and maintenance of monitoring infrastructure. All paths in the following figures are relative to 'playbook_dir'. Usage of the majority of these commands is suggested in *Flowmon Solution Maintenance* [15].

3. Biflow records are a more advanced type of IPFIX, that stores both directions of communication as one record [18].
4. Netcope Technologies a.s. is Czech network card manufacturer, that originated from the same start-up as Flowmon Networks.

```
# Delete user account from all active probes
ansible all
  -a 'php /var/www/shtml/index.php \
      Cli:DeleteUser -login="johndoe"'

# Execute the self-health-check on the selected device(s)
ansible flowmon-cps-ics \
  -a "/usr/sbin/SHC/self-health-checker.sh"

# List all existing user accounts on all active probes
ansible all \
  -a 'php /var/www/shtml/index.php \
      Cli:GetAllUsers'

# Check all probes have correct sender in SMTP configuration
ansible all \
  -a 'php /var/www/shtml/index.php \
      Cli:GetEmailSettings'

# Show all inventory hosts and groups
ansible-inventory --graph

# List all exporter targets on all active probes
ansible-playbook --tags=override -v facts.yml

# Retrieve HWids from cached facts
for i in misc/facts/flowmon-*; do
  $i > jq '.ansible_nodename + " " + .ansible_board_serial';
done | column -t | cut -d'"' -f2
```

Listing 8: List of additional scripts and macros

4 Conclusion

Before the conclusion it needs to be disclosed, that during the course of writing this thesis, Flowmon Networks a.s. was acquired by Kemp Technologies, which was subsequently bought by Progress Software Corporation, therefore some of the products were renamed and the documentation cited in the thesis might have been moved to a new location.

With regard to all previously stated facts, it is apparent that the foundation laid by the automatically generated inventories is objectively the most beneficial practical result of this thesis. The automation scripts created, resolve mainly the issues related to custom components, auditing of the active devices and marginal parts of the Provisioning Process. The fundamental part of FCC configuration still requires a significant amount of manual work that needs to be done by the administrator. Consequently, there are several areas where the development by Flowmon Networks a.s. would significantly improve the automation capabilities.

The first area of possible improvement is the configuration handling, especially a standardised way to provision new Flowmon OS appliances. According to unofficial information, the most likely platform to be used is Terraform[1]. Though, the acquisition of Kemp Technologies by Progress Software Corporation would suggest the usage of Chef[2] since it also belongs to the same company.

The second area is the monitoring of the device. While this area is in lesser need than the aforementioned provisioning, there is certainly room for improvement. The most obvious would be to implement the two Syslog related feature requests created during the course of this work. One step further would the addition of an endpoint with Prometheus[3] compatible metrics, which would include the information presently exposed by SNMP objects (see *Supported SNMP Objects - Flowmon v11.1* [13]) as well as statics that are currently only present in logs of `flowmonexp5`. Parsing of the abovementioned logs has already been implemented by Bc. Tomáš Biloš in *Správa inteligentných senzorov v cloudovom prostredí* [*online*] [19]. The implementation of the endpoint itself could be done via 'text-collector' plugin in 'node

1. Terraform is software that deals with infrastructure provisioning, but opposite to Ansible should not be used for configuration management.
2. Chef is a tool that deals with a similar set of problems as Ansible, but uses agents running on each node rather than SSH to distribute configuration.
3. Prometheus is a monitoring system and time-series database developed by CNCF (*Cloud Native Computing Foundation*.

exporter'[4], which takes plain-text files from a given directory and exposes their contents as Prometheus metrics on an HTTP endpoint. Another way to implement it would be by using pushgateway[5] software, which serves the same purpose but does it via RESTful API instead of a directory.

The third and last suggested improvement is the addition of the option to use S3-compatible storage for the official backups. The main reason for it is the additional features that it offers such as versioning and automatic expiration (see chapter 1.3.6).

4. Node exporter is software that exposes system metrics in Prometheus format, developed by CNCF.
5. `https://github.com/prometheus/pushgateway`

Bibliography

1. RAČEK, Matúš. *Platforma pro dotazování nad kyberbezpečnostními daty* [*online*]. 2021. Available also from: Dostupn%C3%A9%20z%20WWW%20%3Chttps://is.muni.cz/th/brm68/%3E. Diplomová práce. Masarykova univerzita, Fakulta informatiky, Brno. Supervised by Daniel TOVARŇÁK.
2. LUCAS Michael, 1967-. *Network flow analysis*. San Francisco: No Starch Press, 2010. Available also from: http://www.loc.gov/catdir/enhancements/fy1009/2010015790-d.html. Includes index.
3. HOFSTEDE, Rick; ČELEDA, Pavel; TRAMMELL, Brian; DRAGO, Idilio; SADRE, Ramin; SPEROTTO, Anna; PRAS, Aiko. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys & Tutorials*. 2014, vol. 16, no. 4, pp. 2037–2064.
4. ZSEBY, Tanja; CLAISE, Benoît; QUITTEK, Juergen; ZANDER, Sebastian. *Requirements for IP Flow Information Export (IPFIX)* [RFC 3917]. RFC Editor, 2004. Request for Comments, no. 3917. Available from DOI: 10.17487/RFC3917.
5. *Science of Network Anomalies* [online]. Brno: Flowmon Networks a.s., 2021 [visited on 2021-10-11]. Available from: https://www.flowmon.com/en/blog/science-of-network-anomalies.
6. GEERLING, J. *Ansible for DevOps: Server and Configuration Management for Humans*. Leanpub, 2015. ISBN 9780986393419. Available also from: https://books.google.cz/books?id=oLuVjgEACAAJ.
7. *YAML Ain't Markup Language (YAML™) Version 1.2. 2009* [online]. Oren Ben-Kiki, Clark Evans, and Ingy döt Net [visited on 2021-04-16]. Available from: https://yaml.org/spec/1.2/spec.html.
8. *Jinja - Jinja Documentation* [online]. The Pallets Project, 2007 [visited on 2021-04-16]. Available from: https://jinja.palletsprojects.com/en/2.11.x/.
9. SERMERSHEIM, Jim. *Lightweight Directory Access Protocol (LDAP): The Protocol* [RFC 4511]. RFC Editor, 2006. Request for Comments, no. 4511. Available from DOI: 10.17487/RFC4511.
10. *Syslog* [online]. [N.d.] [visited on 2021-11-15]. Available from: https://www.paessler.com/it-explained/syslog.

11. LONVICK, Chris M. *The BSD Syslog Protocol* [RFC 3164]. RFC Editor, 2001. Request for Comments, no. 3164. Available from DOI: 10.17487/RFC3164.

12. GERHARDS, Rainer. *The Syslog Protocol* [RFC 5424]. RFC Editor, 2009. Request for Comments, no. 5424. Available from DOI: 10.17487/RFC5424.

13. *Supported SNMP Objects - Flowmon v11.1* [online]. Brno: Flowmon Networks a.s., 2019 [visited on 2021-03-29]. Available from: https://portal.flowmon.com/s/contentdocument/0695p00000cn8A9AAI.

14. *Flowmon-Probe-Specification.aspx* [online]. Brno: Flowmon Networks a.s., 2020 [visited on 2021-03-30]. Available from: https://www.flowmon.com/getattachment/d1833e35-16ff-4926-9dad-75b0cb589cfe/Flowmon-Probe-Specification.aspx.

15. *Flowmon Solution Maintenance* [online]. Brno: Flowmon Networks a.s., 2019 [visited on 2021-04-16]. Available from: https://portal.flowmon.com/s/contentdocument/0690N00000GWJZmQAP.

16. *Flowmon Recovery Guide* [online]. Brno: Flowmon Networks a.s., 2018 [visited on 2021-11-24]. Available from: https://kemp-artifact-packages.s3.amazonaws.com/Flowmon/Flowmon/11.1.9-Stable/docs/flowmon-recovery-guide.pdf.

17. *Profile Import Using XML* [online]. Brno: Flowmon Networks a.s., 2019 [visited on 2021-04-15]. Available from: https://portal.flowmon.com/s/contentdocument/0690N00000GWJaBQAX.

18. BOSCHI, Elisa; TRAMMELL, Brian. Bidirectional Flow Measurement, IPFIX, and Security Analysis. 2006.

19. BILOŠ, Tomáš. *Správa inteligentných senzorov v cloudovom prostredí* [*online*]. 2021. Available also from: Dostupn%C3%A9%20z%20WWW%20%3Chttps://is.muni.cz/th/oeemc/%3E.

A Structure of Attachment

The attachment contains one Git repository, which consists of a typical Ansible structure:

- `inventories`
  - `host_vars` – variable files for single hosts
  - `group_vars` – variables files for host groups
  - `*.yml` – inventory definitions
- `misc`
  - `facts` – cache of facts
  - `info` – raw information retrieved by facts role
  - `inventory` – cache of dynamic inventories
  - `.vault.key` – password to `ansible-vault`
- `roles`
  - `facts`
  - `idrac`
  - `monitoring`
  - `perun`
- `utils` – scripts to run tests, install dependencies and etc...
- `tasks` – repetitive tasks sequences imported at playbook level
- `ansible.cfg` – configuration of Ansible runtime
- `requirements.yml` – Ansible dependencies
- `*.yml` – Ansible playbooks

B Attachments

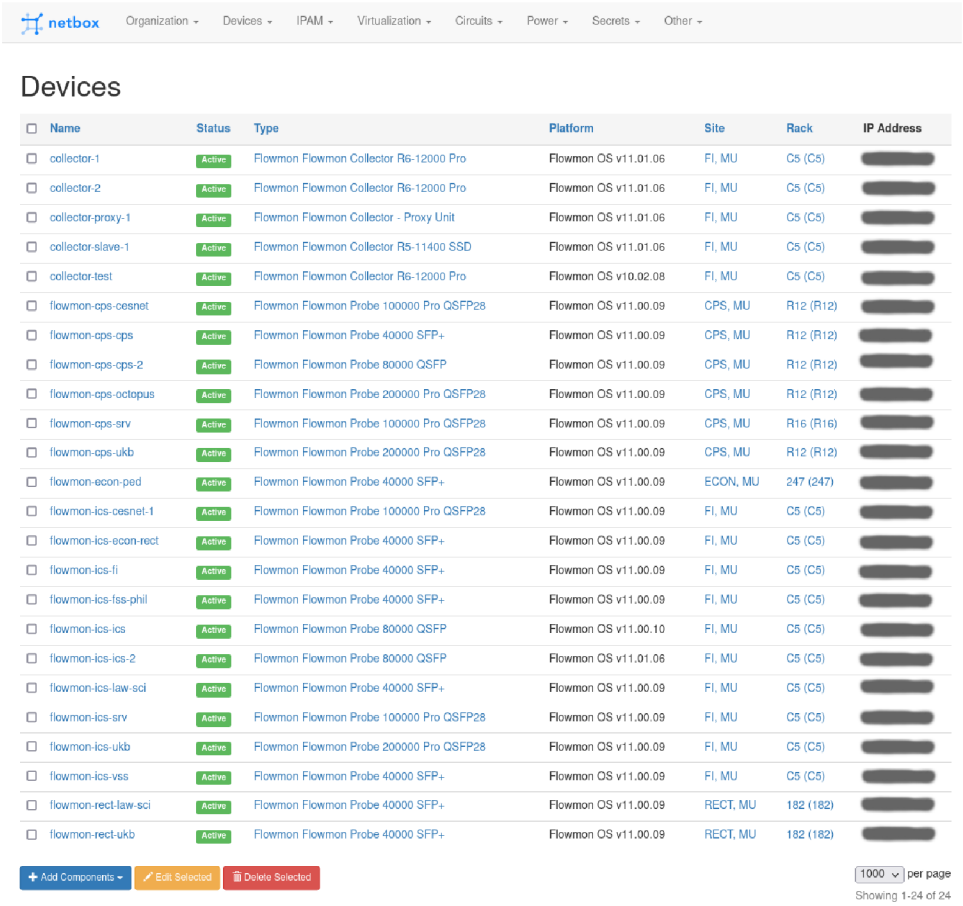

Figure B.1: List of MI devices in Netbox

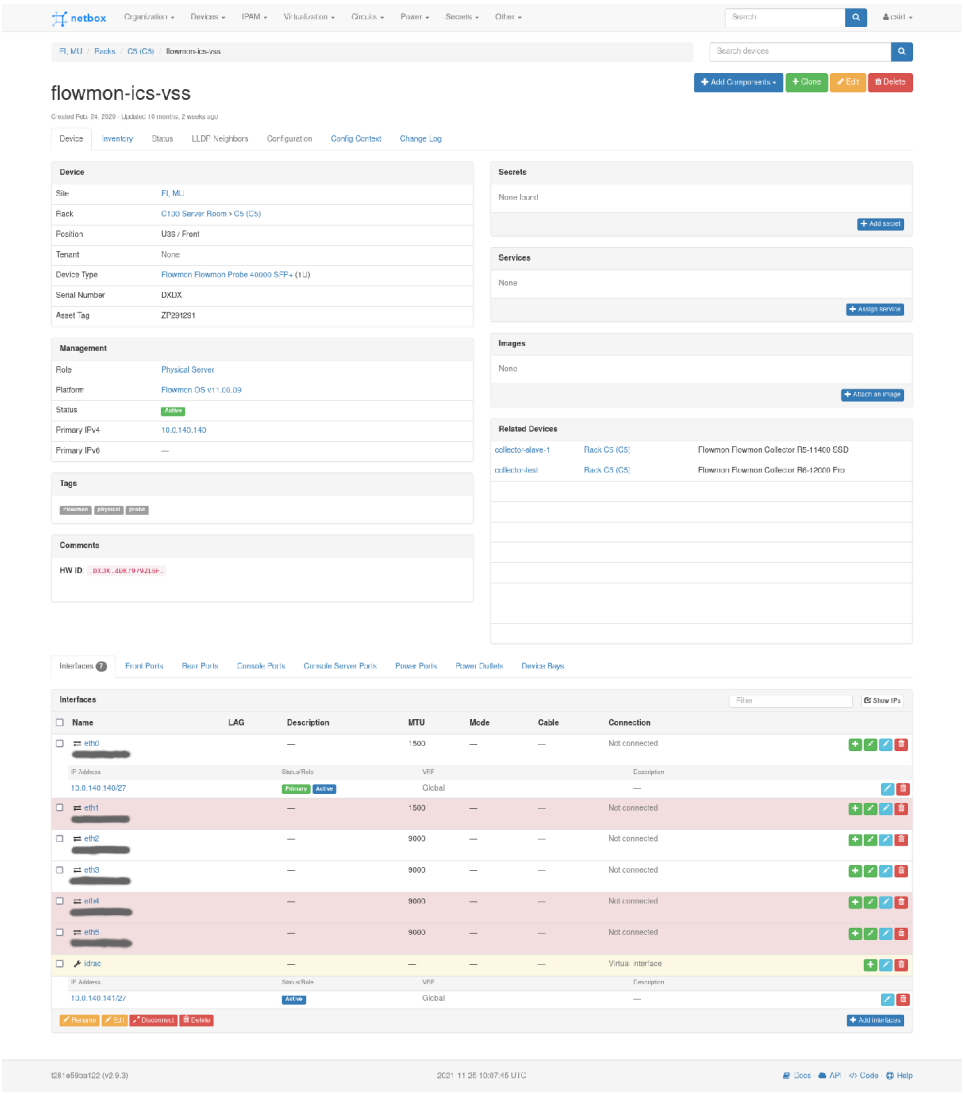

Figure B.2: A network probe device in Netbox

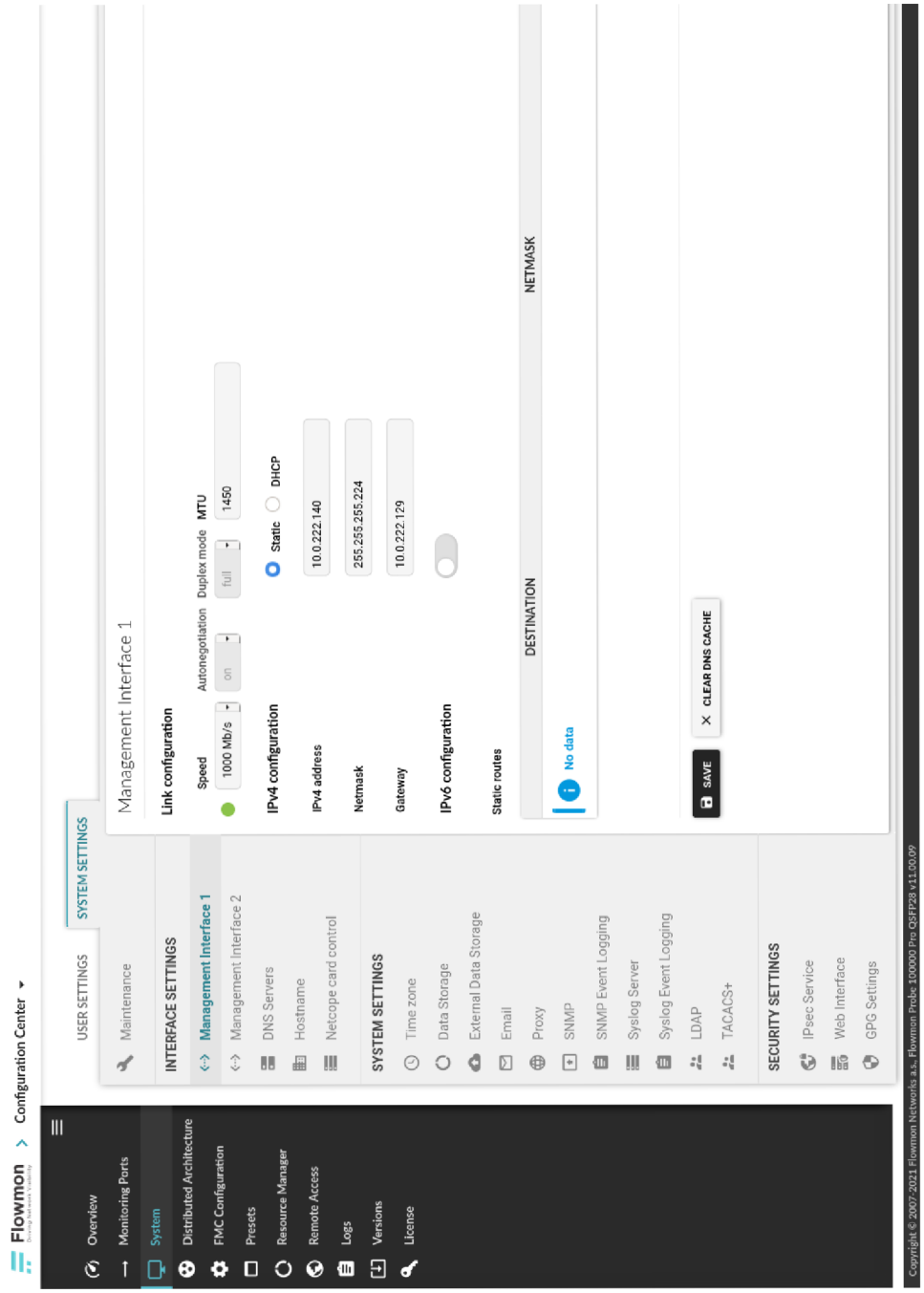

Figure B.3: Networking configuration in FCC

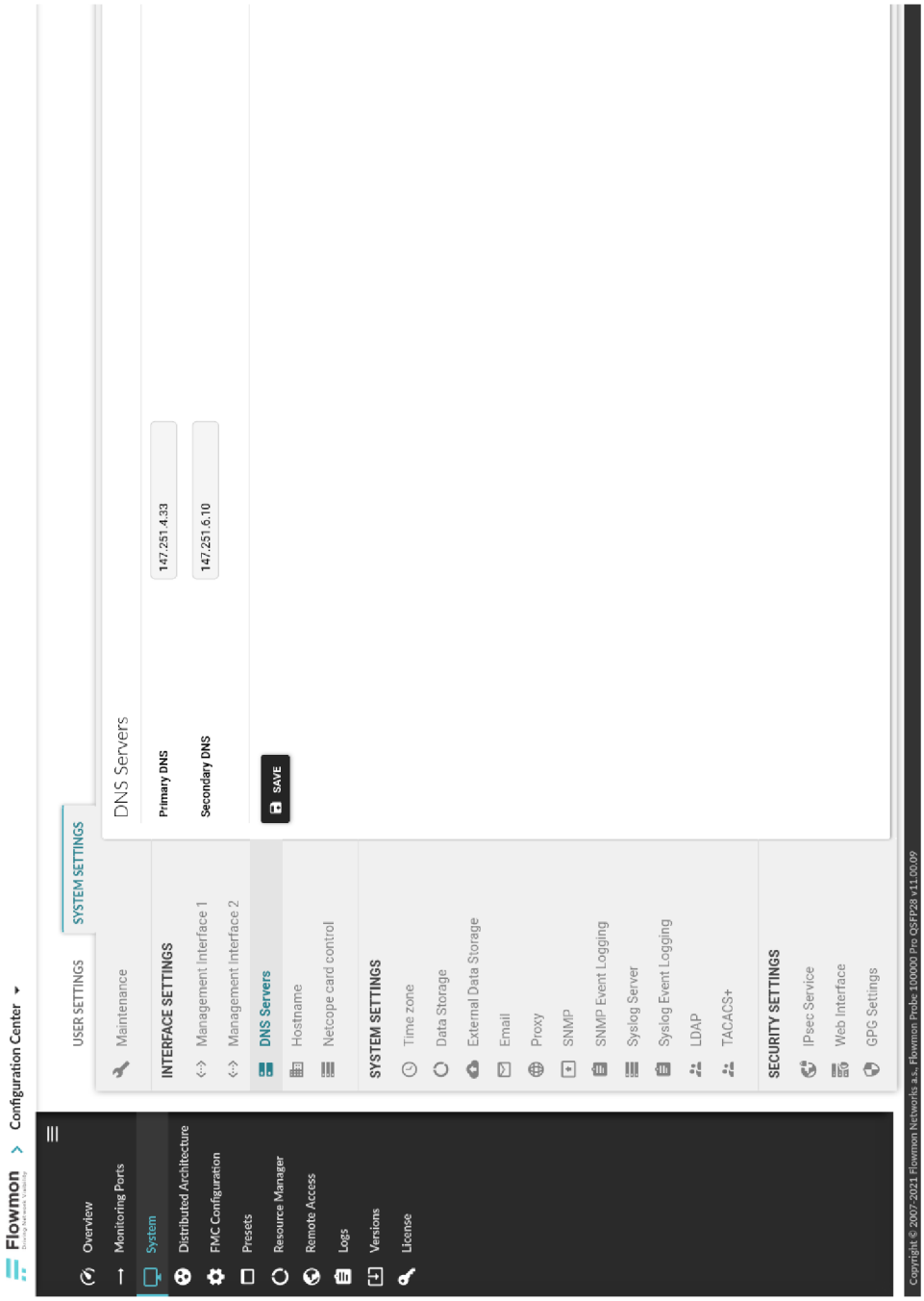

Figure B.4: DNS configuration in FCC

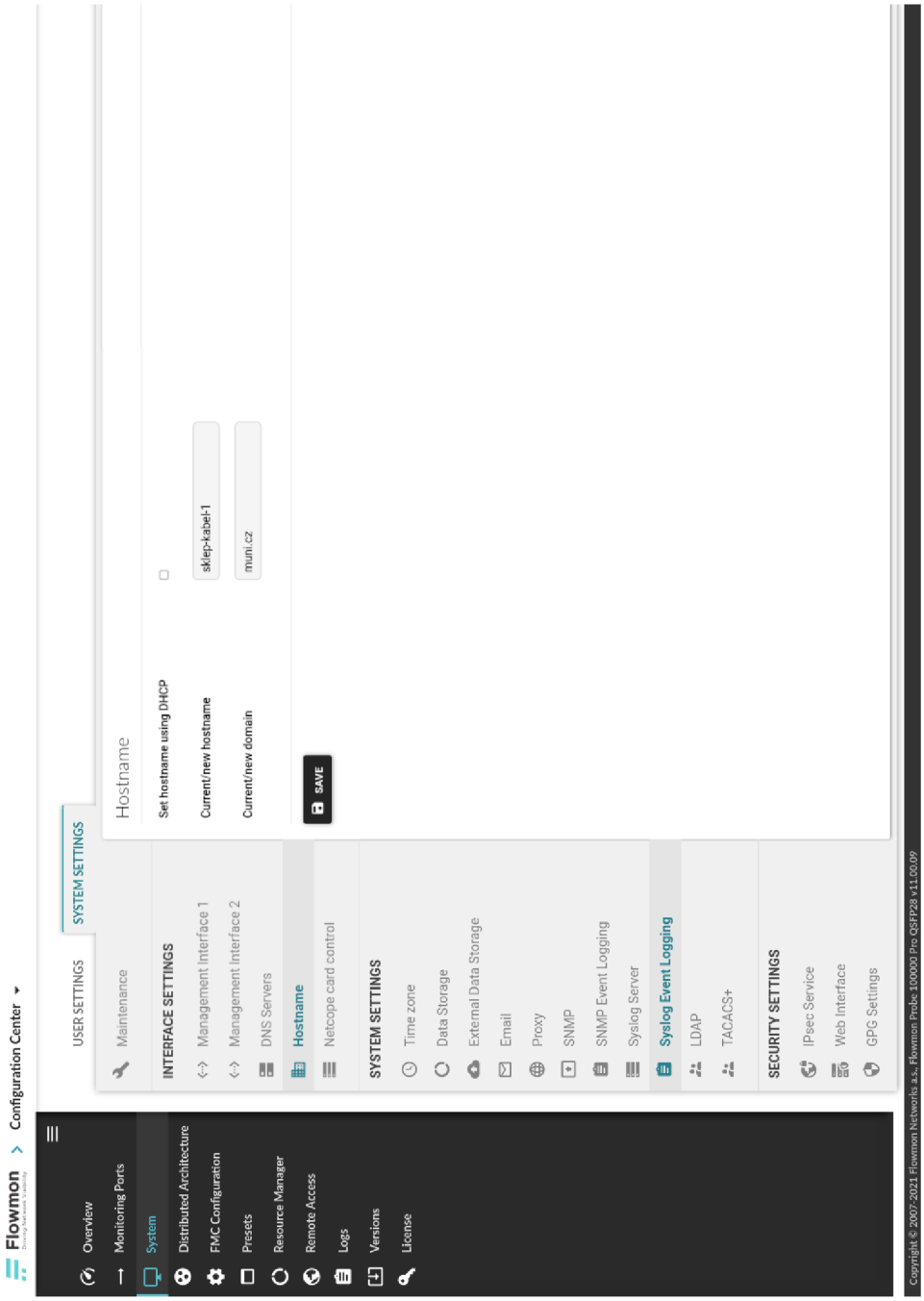

Figure B.5: Hostname configuration in FCC

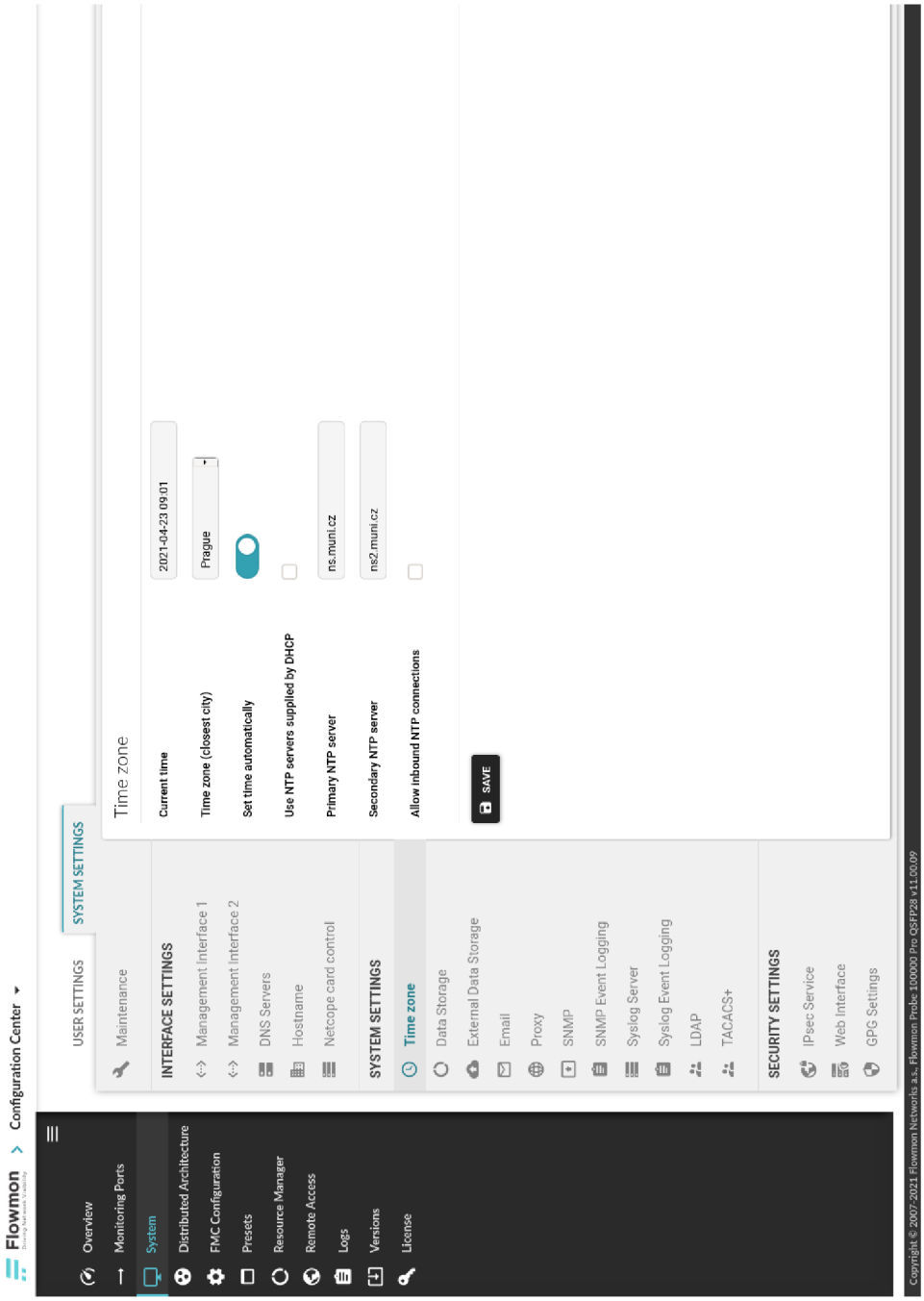

Figure B.6: Timezone configuration in FCC

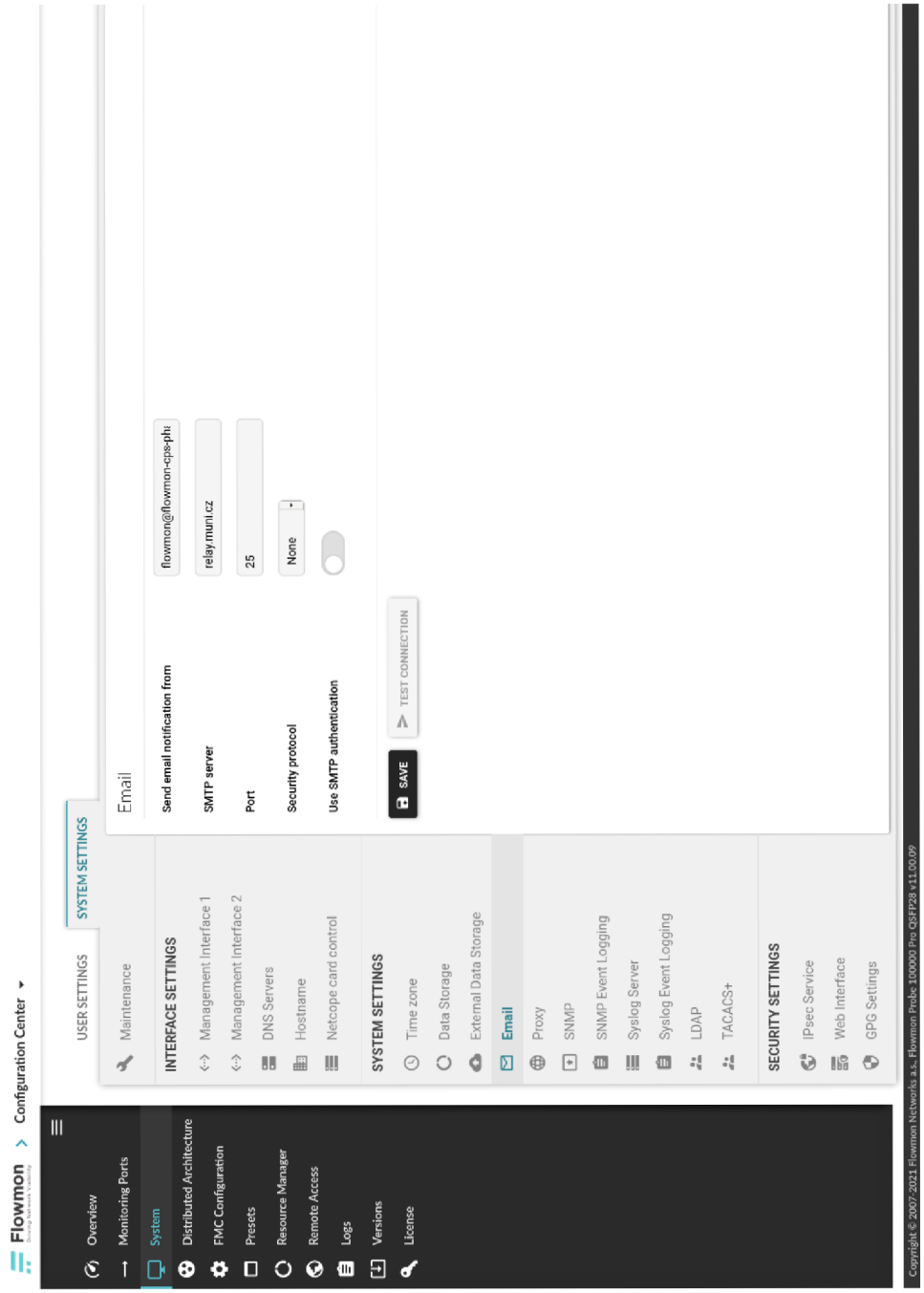

Figure B.7: Email configuration in FCC

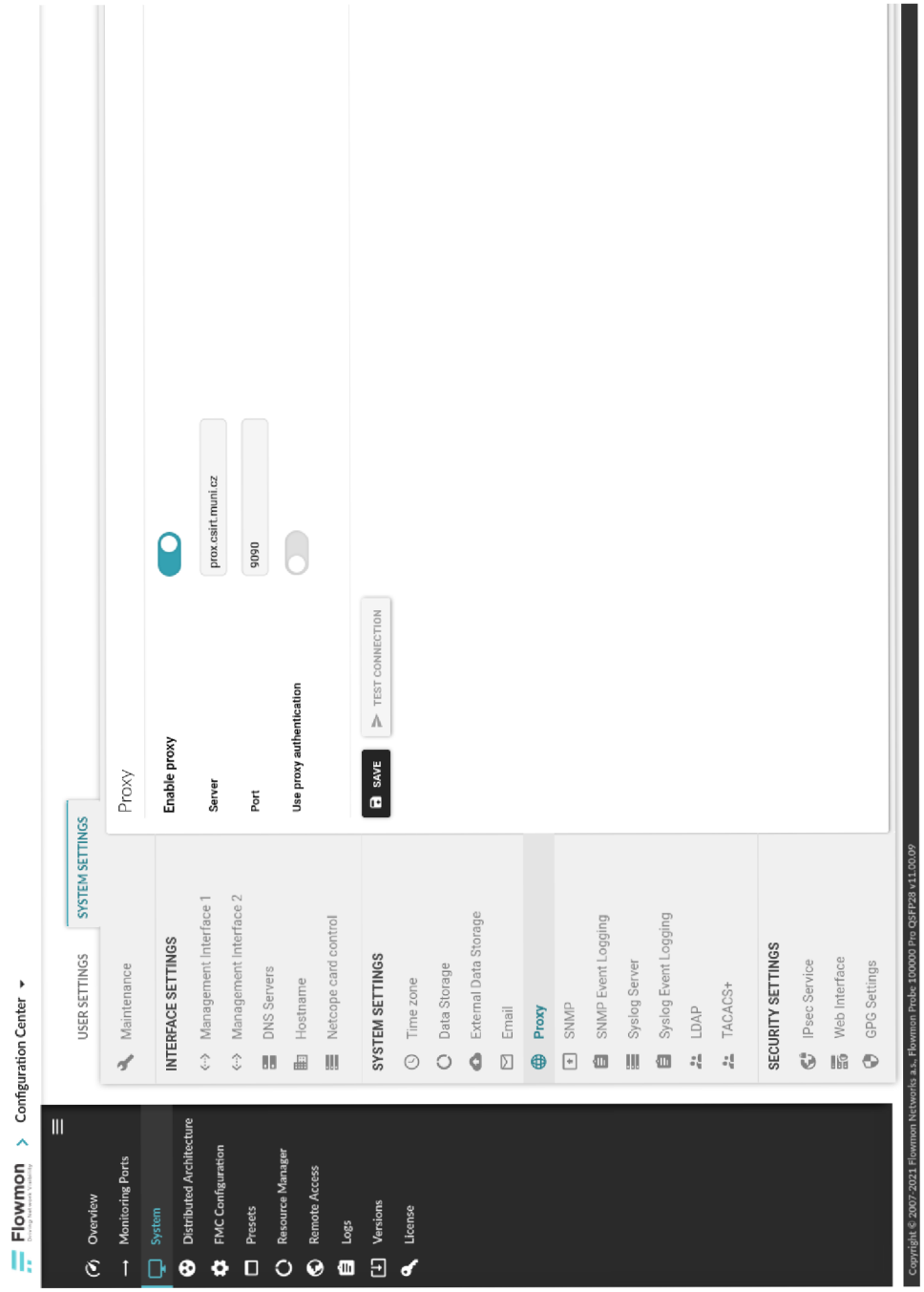

Figure B.8: Proxy configuration in FCC

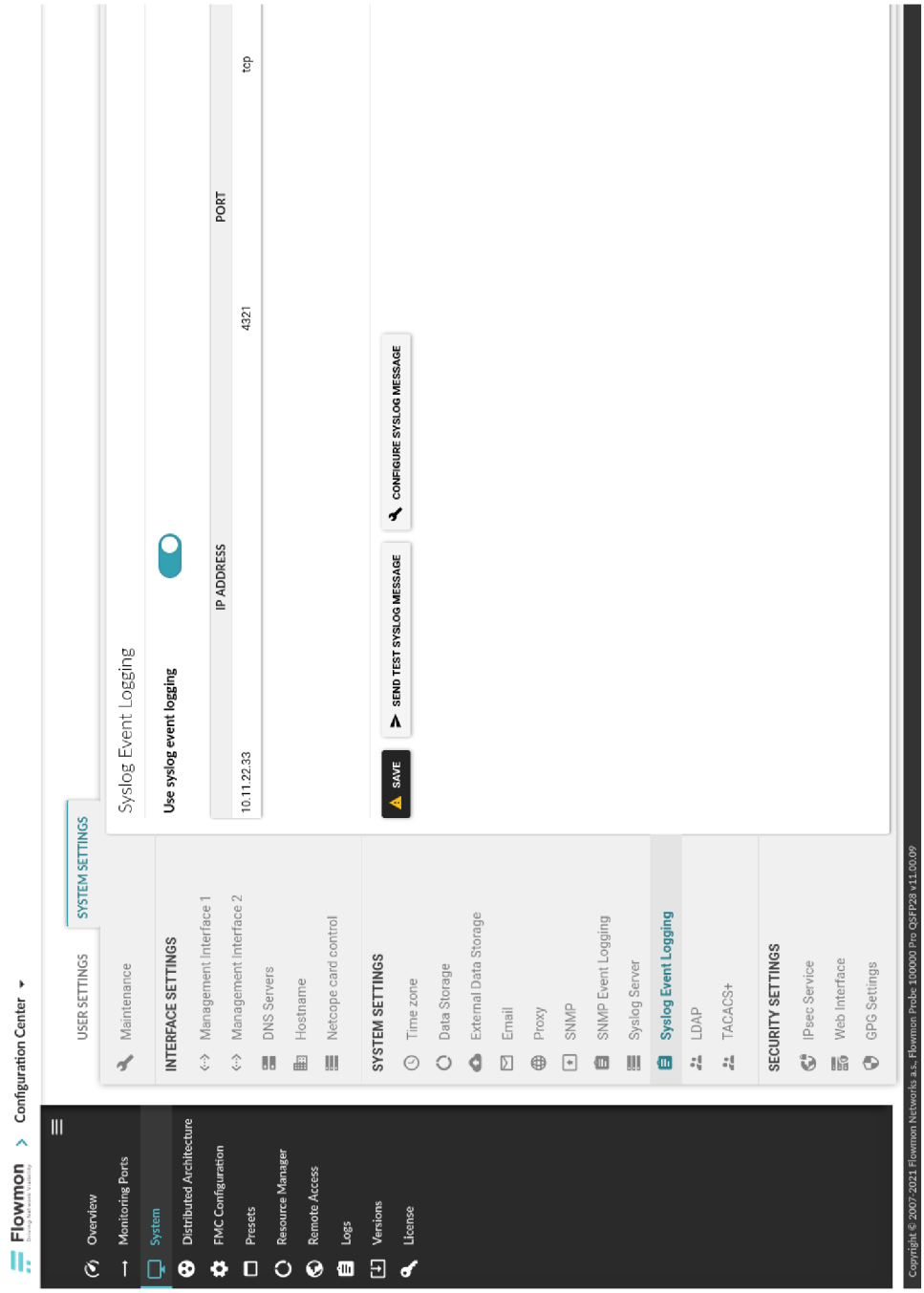

Figure B.9: Syslog event logging configuration in FCC