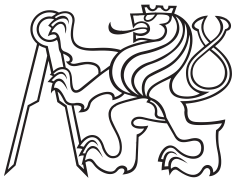
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Telecommunications Engineering

Monitoring System Based on Telemetry in Network CESNET3

Bc. Ladislav Loub

Supervisor: Ing. Jan Kubr, Ph.D.
June 2023

Acknowledgements

I would like to thank my supervisor Ing. Jan Kubr, Ph.D., for all the help and valuable advice while writing this thesis. Secondly, I would like to thank the CESNET company for inciting this topic and allowing me to use all the necessary resources needed for this thesis. Enormous thanks belong to my family for supporting me throughout my studies and helping me with everything that was in their capacity. Namely my mother, Jana Loubová, my father, Ing. Ladislav Loub, and my brother Mgr. Tomáš Masař, Ph.D.. The last thanks belong to my girlfriend Tereza for her patience and support during writing this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

Abstract

The thesis focuses on designing and deploying a monitoring system based on Model-Driven Telemetry technology. This technology is relatively recent. The RFC 9232, partly describing it, was released in may 2022. The thesis is the continuation of my bachelor thesis, in which I have researched Model-Driven Telemetry and its possible usage in the future.

The goal of the system is to monitor operational state data from network devices managed by CESNET. Primary information is data concerning the condition of the devices and statistics about their network interfaces.

The system consists of several parts to be able to fulfill requirements. The parts are: collectors for assembling data, a time-series database for storing them, visualization tools for displaying the data in a user-friendly way and, finally, management of the whole system. The colletors are deployed Telegraf container instances in Docker, InfluxDB is used as a time-series database, Grafana and Nagios are used as visualization tools, and Ansible is used for the management of all the systems.

The result of this work is a system that serves as monitoring of the CESNET network. The thesis could serve as inspiration for other service providers that are considering deploying similar monitoring system.

Keywords: Model-Driven Telemetry, Streaming Telemetry, monitoring system, YANG, SNMP

Supervisor: Ing. Jan Kubr, Ph.D.
Department of Computer Graphics and Interaction,
Resslova 307/9,
12000 Praha 2

Abstrakt

Cílem této práce je návrh a nasazení monitorovacího systému založeného na technolgii Model-Driven telemetrie. Tato technologie je poměrně nová, kdy RFC 9232, které ji částečně popisuje bylo vydáno teprve v květnu 2022. Tato prace navazuje na moji bakalářskou práci, ve které jsem zkoumal telemetrii a její možné nasazení do budoucna.

Navržený monitorovací systém bude mít za úkol sběr stavových dat ze síťových zažízení provozovaných sdružením CESNET. Sbírané informace budou primárně ohledně stavu zařízení samotných a jejích síťových rozhraní.

Systém se sestává z několika částí. Tyto části jsou kolektory pro sběr dat, časová databáze pro uložení dat, visualizační nástroje pro zobrazení statistik v přívětivé formě a správa celého systému. Kolektory jsou nasazeny pomocí softwaru Telegraf jako kontejnery v systému Docker. Časová databáze je použita InfluxDB. Pro vizualizaci jsou použity nástroje Nagios a Grafana. Celá správa systému je udělána pomocí nástoje Ansible.

Výsledkem práce je monitorovací systém, který slouží pro dohled sítě ve sdružení CESNET. Práce může sloužit jako inspirace pro ostatní poskytovatele síťových služeb.

Klíčová slova: Model-Driven telemetrie, Streamovaná telemetrie, monitorovací systém, YANG, SNMP

Překlad názvu: Monitorovací systém založený na telemetrii v síti CESNET3

Contents

Figures

Tables

Chapter 1

Introduction

This thesis directly follows my bachelor thesis, "Network traffic monitoring using Model-Driven Telemetry" [16]. I have focused on a new way of monitoring network devices named Model-Driven Telemetry. The main reason for researching this technology was the problem that started to appear during the bachelor thesis. All currently used monitoring systems in CESNET are SNMP based and reaching their limits. SNMP stands for Simple Network Management Protocol. It is a protocol used to manage and monitor devices on a network. SNMP is widely used in network management systems to collect information about network devices such as routers, switches, servers, and printers [22]. One system would need a significant architecture overhaul to suffice current trends. The other is paid software, which would require considerable investment. In that order, it was decided to explore options of Model-Driven Telemetry. Based on the results, the decision was made to continue on this path and deploy a functional monitoring system.

Since finishing my bachelor's thesis, some things regarding Model-Driven Telemetry technology have changed. When I started researching it, it was only in the early draft [23], but since then, it has matured into full RFC 9232 [24]. This RFC is not for any particular technological solution but attempts to specify telemetry and all related terms and technologies. In later chapters, I will review the changes this RFC brings, where the term telemetry will be explained.

My bachelor thesis aimed to research and deploy a Model-Driven Telemetry collector. This collector was designed for testing purposes but not for production monitoring. Deployment of the collector was managed by the bash script, which installed all needed parts. Since the goal of the thesis was not any production system, the script constantly deployed an entirely new monitoring stack. The collector's purpose was to measure the differences between SNMP and Model-Driven Telemetry.

The testing offered valuable performance data on differences between SNMP and Model-Driven Telemetry. Based on the results, it was decided to continue with the MDT[1] path. Nevertheless, as with most new systems, it must be backward compatible. The restriction then must be included in the system's design. The final system must be able to collect data using SNMP and

[1] MDT (Model-Driven Telemetry)

simultaneously using MDT.

In the testing setup, the collector consisted of four parts. The data collection part was handled by Telegraf and Pipeline tools. This data was then saved into the InfluxDB database. The last part was the visualization of collected data. The Grafana software was used for this purpose. All tools were available as open-source software, so no purchases was necessary. This thesis' primary analysis task will be to evaluate the tools used in the testing setup for the new monitoring system. The biggest challenge will probably be the scalability of the collector. The collectors must be able to handle traffic from hundreds to thousands of devices in the network. Also, the monitoring system must be easy to deploy and automated as much as possible.

On the previous collector, I measured the impacts of SNMP and Model-Driven Telemetry in the old network CESNET2 with Cisco CRS platform devices. Even though these devices had relatively low RAM[2], implementing a monitoring system based on Model-Driven Telemetry was decided to be beneficial. Recently the old CESNET network is being upgraded to newer devices. These new devices have abundant memory, so MDT usage makes more sense. This technology is becoming a preferred way and gets much more updates on the manufacturer side.

The new CESNET3 network currently consists inclusively of Cisco devices. Regardless, the collector must be able to handle data from multiple vendors. To achieve this requirement, the data collecting part of the system will receive data using SNMP and Model-Driven Telemetry. Since these two technologies are quite different, data normalization will be needed. This task can be achieved by writing a script or finding another possible tool. Incoming data using SNMP and MDT are different in structure. The system must have means to unify similar data structures based on the administrator's description.

[2]One of the disadvantages of MDT compared to SNMP is its higher usage of RAM.

Chapter 2

Analysis

2.1 Current monitoring technologies

Network monitoring is one of the essential parts of keeping the network operational and providing services for customers with adequate quality. Throughout time there were many methods and technologies used for monitoring.

One of the well-known technology today is SNMP. This protocol went through three major versions, which filled in missing features or fixed its issues. But as network management moves more and more toward programmability, the SNMP can no longer provide adequate options.

The new way of monitoring is called Model-Driven Telemetry or Streaming Telemetry. Precise details and explanations of these terms will be in the next chapter. The situation progressed so far that some monitored data are no longer available using SNMP.

Another technology that monitoring systems can use is CLI[3]. The system can initiate an *SSH*[4] session to the network device and execute basic show commands. This way is very inconvenient for any machine processing as data received by this method can have various forms and formats. The CLI access is still used mainly by network engineers, but even this access is supposed to be very rare, and the network should be managed from the central management system rather than from the command line.

Next, technology also considered part of device monitoring can be *syslog*. The primary use of syslog is sending device logs, but in terms of state monitoring, it lacks in various ways. Syslog messages are limited in terms of the information they can convey. While syslog messages can contain basic information such as severity level and timestamp, they do not provide detailed context about the event that triggered the message. The messages are typically sent in a fixed format without any deeper structure, which limits their usefulness for monitoring and analysis. The last disadvantage

[3]Command Line Interface (CLI) is a method of interacting with a network device using text commands typed into a command line interface instead of a graphical user interface (GUI).

[4]Secure Shell (SSH) is a network protocol used for secure remote access to devices over unsecured networks. SSH provides a secure, encrypted communication channel between two devices.

is the scalability of syslog. As the number of network devices and syslog messages increases, it can become challenging to manage and analyze the data generated by syslog. This can result in slower response times, increased storage requirements, and other performance issues.

The current trend is to make the network manageable as much as possible programmatically. The way to achieve this is by using the YANG models. It's essential to explore these new technologies as more vendors implement them. Also, we can observe more common ground between vendors as most of them use at least OpenConfig YANG models. The more significant vendors, such as Cisco, Juniper, Huawei, Arista, and Alcatel-Lucent, all have YANG model usage options.

2.2 Network devices

The essential part is the identification of network devices from which we are going to collect data. As I've already mentioned, all the devices are network devices, such as routers and switches. The collection of data from servers is not the goal of this system. Other monitoring systems are in place in CESNET for this purpose. The majority of machines are from Cisco. Although there is mostly only one vendor, devices still need to be differentiated into groups.

I'm going to use several groups. The CESNET3, CESNET2, DWDM and Other. The goal of this thesis is the monitoring of devices in group CESNET3, but I will be designing this system to be able to cover the rest of the groups as well.

The **CESNET3** network devices group consists only of Cisco devices. This group is also considered as the IP/MPLS layer of the network [5]. All devices in this group are new and are currently being installed as the next generation of the CESNET network for customers. The core routers in this group are Cisco 8000, ASR9000, and NCS540 series. These devices are running the IOS-XR operating system with the current version 7.5.2. The second part of the network is aggregation switches, which are Cisco Catalyst 9000 series. These are running the IOS-XE system with version 17.06.03. The last part of the network is OOB [6] routers that are ISR1100 series, also running the IOS-XE operating system at version 17.06.03. The OOB routers are used to manage the CESNET3 network. The whole CESNET3 network is designed on private IP addresses, limiting access to the devices.

The **CESNET2** group could be considered as older devices that are currently in use. It is also part of the IP/MPLS layer, and its design can be seen in figure 2.1. These devices are being replaced with new ones. Some of the

[5] Internet Protocol Multi-Protocol Label Switching (IP/MPLS) is a routing system that enables fast data switching from one node to the next based on labels. Commonly, the packets in IP networks need to consider the destination addresses to control the function of the routers, which takes time. IP/MPLS finds an alternative to this process by switching packets based on the label [6].

[6] Out-of-Band (OOB) refers to a separate management network that is used to control and monitor devices such as routers, switches, and servers.

old devices probably aren't going to be replaced but will be incorporated into the new network. These devices are, in some cases, limited in functionality regarding connection with this newly designed monitoring system. Some of them are not Model-Driven Telemetry capable, and SNMP remains the only option for monitoring.

Devices in this group are routers Cisco CRS series with IOS-XR 6.6.3 version. I tested Model-Driven Telemetry on some of these devices during my bachelor thesis [16]. Other devices are Alcatel-Lucent routers Nokia 7750 SR with TiMOS-C-16 operating system. These devices have only the SNMP option and are not MDT-capable. However, it doesn't matter, as all Cisco CRS and Nokia 7750 routers are being replaced. The devices in this network, which have some level of probability of staying active, are various portions of Cisco Catalyst switches. Many of these run only on the IOS operating system and are only SNMP capable.

The next group is **DWDM** [7]. The DWDM group consists of devices used in the optical part of the network. To better understand this layer, figure 2.2 visualize both IP/MPLS and DWDM layers. The lowest layer of figure represents physical layer. In case our case this layer is DWDM system. Above this system is then built IP/MPLS layer on figure represented by "Logical Layer (data network)". This layer represents routers and switches in network. The top layer of figure is users layer that represents services provided by whole network to customers.

The DWDM layer serves as an underlay and, in ISO/OSI [8] terms, as the network's physical layer. This part of the network is also relatively new in CESNET. It consists of Cisco NCS 2000 devices. Unfortunately, these devices don't support Model-Driven Telemetry yet. But according to Cisco presentations, this feature is being considered and probably will be released in the future. As of now, these devices support SNMP.

The last group is named **Other**. The probability of these devices being monitored by this system is relatively low. It consists of various laboratory switches, WiFi controllers, access points, etc. The monitoring capabilities of these devices can vary and will be considered only with the request to be monitored. The fallback solution is considered SNMP.

2.3 Telemetry term explanation

Since May 2022, RFC 9232 Network Telemetry Framework has been available, which gives more information about used terms. The purpose of this RFC is not any technological implementation but an overview of different parts that can be considered Network Telemetry. In recent times it's possible

[7] Dense Wavelength Division Multiplexing (DWDM) is a technology used in optical fiber networks to increase the network's capacity by allowing multiple data signals to be transmitted simultaneously over a single fiber optic cable [19].

[8] ISO/OSI, also known as the OSI model, is a conceptual framework for understanding computer network communication protocols. It stands for Open Systems Interconnection reference model.

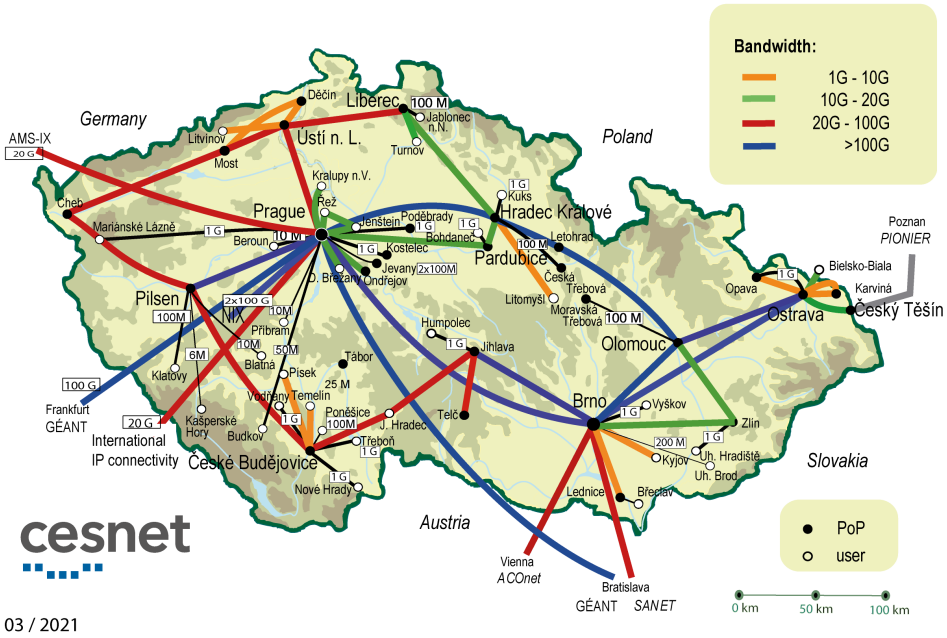


Figure 2.1: CESNET2 topology (Source: [1])

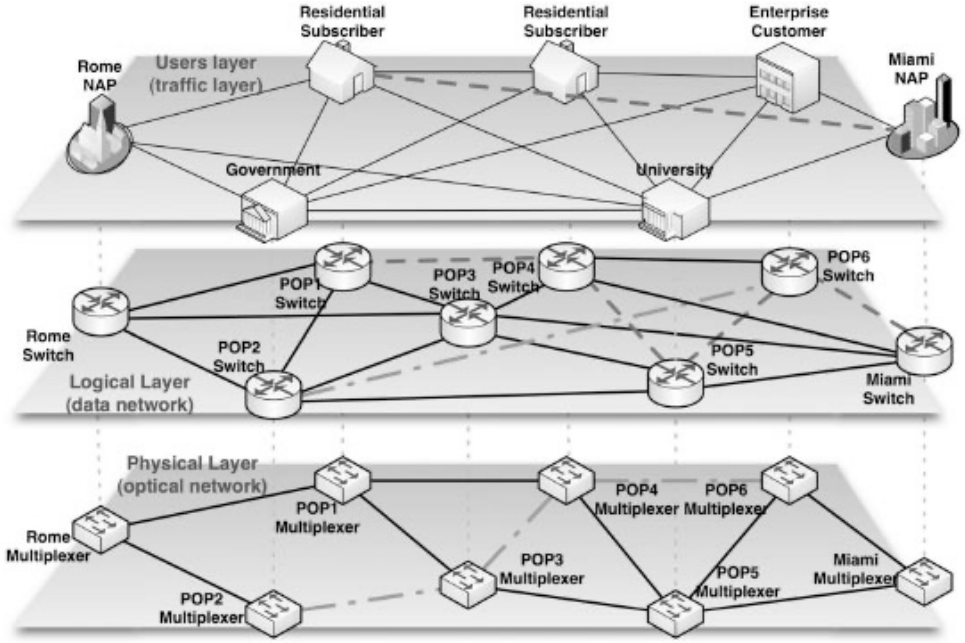


Figure 2.2: DWDM and IP/MPLS layers (Source: [4])

to see many uses of the term telemetry, which are entirely valid uses. In section 2.1. Telemetry Data Coverage is specified: "Any information that can be extracted from networks (including the data plane, control plane, and

management plane) and used to gain visibility or as a basis for actions is considered telemetry data." [24]. According to this definition, almost all data pulled from a device can be considered as Network Telemetry data.

This thesis result won't be using technologies such as Flow Telemetry [27] or Sflow Telemetry [20], which focuses on exporting flow data from network devices [9]. Another not used term is In-band Telemetry [28] as that also revolves around data flow and not state monitoring of devices.

On the other hand, valid terms used in this context are Model-Driven Telemetry, Streaming Telemetry, and Network Telemetry. Model-Driven Telemetry is based on YANG models. Streaming Telemetry describes when devices stream data to the collector without the need of a received event. Another term is Event-Driven Telemetry which is a subset of Streaming Telemetry. The difference is that in an Event-Driven scenario sends data based on events in the network, such as when one interface changes state. Streaming Telemetry transmits this data continuously [10]. Next is Open Telemetry [9]. This is almost synonymous with Model-Driven Telemetry, only that YANG models are developed by OpenConfig.

The context in which the term telemetry is used in this thesis is simply about **collecting state data** from network devices (such as switches and routers) using YANG data models. The term Model-Driven Telemetry (MDT) is then a type of telemetry that uses a data model to define the structure of the data that is collected and transmitted from a network device. The primary goal of Model-Driven Telemetry is to replace the SNMP. The reason for this goal is to make state monitoring of devices more automated and simplified.

Visualization of the telemetry stack can be seen in pictures 2.3 and 2.4. The first one provides a basic overview of layers of Telemetry. Raw data are modeled on the device into YANG data format. Then are exported using transport protocol to the collector based on the device's configuration.

The second figure visualizes the protocols used. This figure needs explanation since it visualizes two parts in Transport, Encoding, and Protocol sections. The NETCONF protocol, XML encoding[11], SSH, and TCP transport are part of the Model-Driven Configuration rather than Model-Driven Telemetry. It's the element of Closed-loop automation named by Cisco. The main principle is that Model-Driven Configuration should be able to change device configuration based on data received by Model-Driven Telemetry, and vice-versa MDT should be configured using Model-Driven Configuration.

[9] These technologies are also commonly referred to as Netflow data [18].

[10] Further explanation of differences between Streaming Telemetry and Event-Driven Telemetry is explained in my bachelor thesis [16, Sec. 2.3].

[11] Extensible Markup Language (XML) is a markup language and file format for storing, transmitting, and reconstructing arbitrary data. It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable [30].

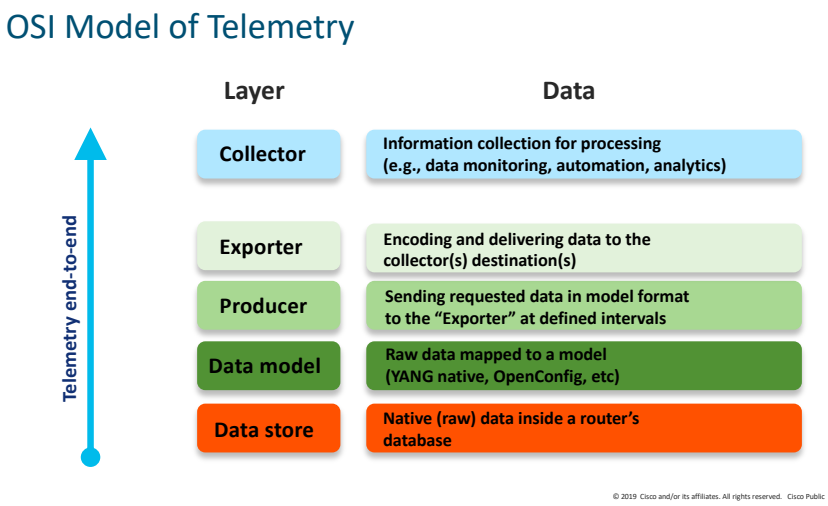


Figure 2.3: OSI Model of Telemetry (Source: [10])

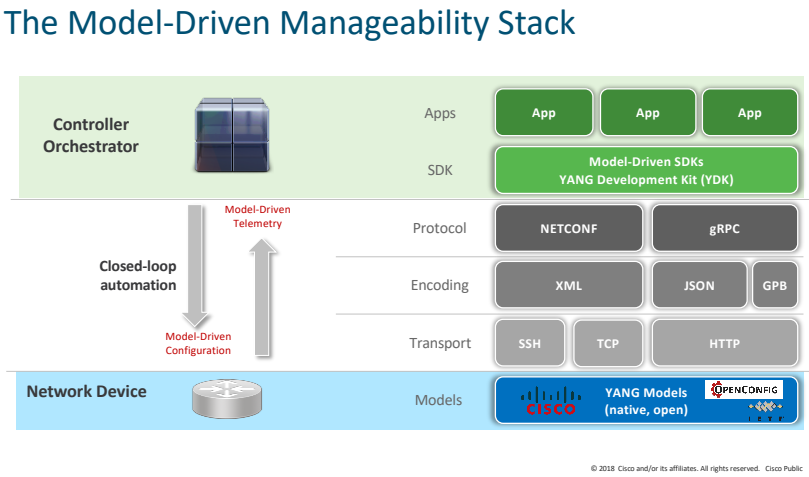


Figure 2.4: The Model-Driven Manageability Stack (Source: [10])

2.4 Automation tools

One of the vital requirements of monitoring system management will be its automation options. The system will handle a large number of devices, and the option to configure everything by hand won't be a possibility. Also, it's much easier to share the management of the system between administrators if it's well-automated and documented. For that reason, an appropriate automation tool must be chosen.

The job of the automation tool will be deploying and configuring Model-Driven Telemetry data collectors on servers. The second task will be the configuration of network devices. Data needed for the automation tool to work will be fetched from the network documentation system. Three tools were considered for this task: Puppet, Terraform, and Ansible.

The first option, Puppet, is an open-source tool that uses declarative language to define the desired state of infrastructure resources and applies changes to achieve that state. This means that you determine how you want your infrastructure to be configured, and Puppet will handle the changes required to make it happen. Puppet has an agent-based architecture, which means that a lightweight agent is installed on each node being managed, allowing it to be easily handled at scale. Some of the disadvantages could be its steep learning curve, as it may require a significant investment of time and resources to become proficient in using the tool effectively [13] [21].

The second option is Terraform. It is also an orchestration and provisioning tool that is open-source. With Terraform, it is possible to define infrastructure resources in code, create version-controlled configurations, and manage infrastructure as if it were a software application. The best use of Terraform is for provisioning software in the cloud, which is different from this designed monitoring system [15][14]. Detailed functioning of Teraform can be found on their website [29].

The last option is Ansible, an open-source automation tool that enables users to automate IT infrastructure tasks. It is agentless and manages nodes with SSH protocol. Complete documentation and more details are available here [11]. Ansible is easy to set up and use, with a low learning curve, making it an ideal tool for small and medium-sized organizations without a dedicated IT infrastructure team.

The decisive factor in choosing an automation tool was its simplicity to use. In that regard, the Ansible seems to be the tool for this purpose. Ansible is also already used for the deployment of some parts of the configuration of network devices in the CESNET3 network. It would be much easier to use the same tool for compatibility later. Also, many system administrators in CESNET use Ansible as an automation tool, which means a much larger knowledge base and possible experience sharing.

2.5 Network documentation by Netbox

The best option for the automation tool was decided Ansible system. Since this system is primarily used by CLI, the GUI for managing this tool would be useful. For this purpose, it's best to use the already existing system, currently deployed and under development in CESNET.

Netbox is a network documentation system that has complete documentation of all network devices from the CESNET3 network [12]. It's also being extended to hold all network devices which CESNET manages as a Service provider. All of this means that this system already has support within CESNET and is easily extendable. Complete documentation of this tool is available on the website [26].

Netbox is also an open-source system, which is free and has an active community around the world, that is still working on new features and bug fixes. The great benefit of it is its comprehensive API, which covers all data that are stored within Netbox. Only minor changes need to be made to use this system as GUI for Ansible.

The system is currently filled with more than 400 devices seen in the Netbox dashboard in figure2.5 and many more pieces of information regarding the network state. Every device has its management IP address stored. The view of the device will be extended by Custom fields with information about the data collector and service TCP port.

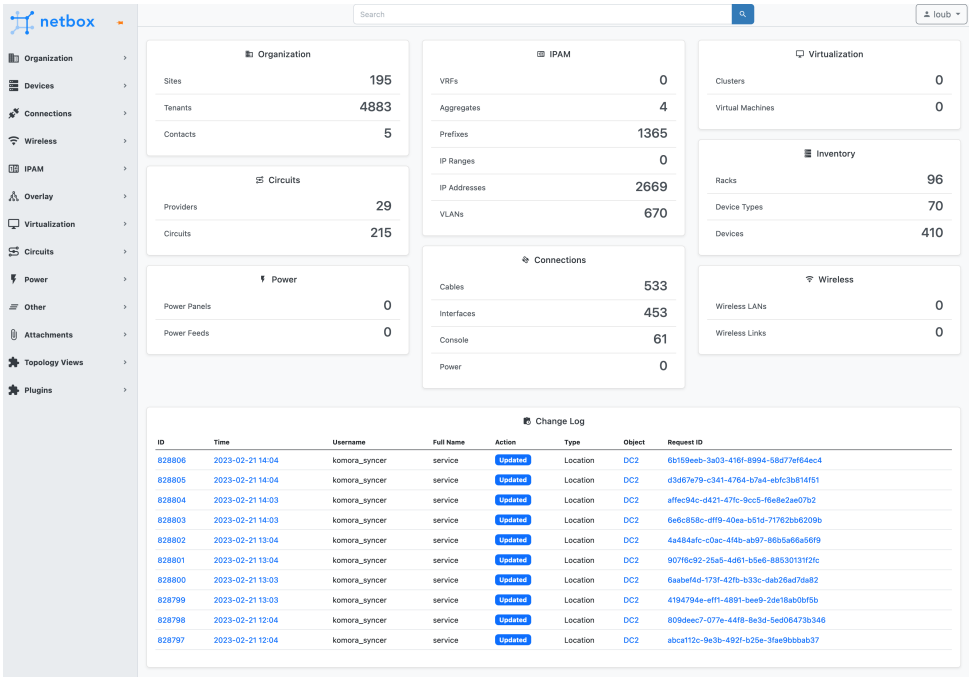


Figure 2.5: View of the Netbox system

[12] Due to ongoing migration of network is possible that not all devices are immediately added to Netbox, with all parameters.

2.6 Network management by NETCONF

The Network Configuration Protocol could be considered as a new approach to maintaining networks and deploying configurations. As shown in figure 2.4 in section 2.3, network management done by modeling language is a significant part of the new approach. The NETCONF protocol was published as early as December 2006 in RFC 4741 [8] by IETF (Internet Engineering Task Force) [13]. NETCONF stands for Network Configuration Protocol and is used to manage network devices such as routers, switches, and firewalls.

Although CLI configuration can be somewhat automated, its templatization is challenging and time-consuming. Also very different among various vendors. The advantage of using NETCONF is that it provides a programmatic way of configuring network devices, which can help reduce errors and increase efficiency compared to the CLI option.

NETCONF uses XML (Extensible Markup Language) to format messages between clients and servers. A client can send a request to a server to modify, retrieve, or delete network configuration data. The server responds to the request with a confirmation or error message. NETCONF is often used in conjunction with YANG, which is a data modeling language used to describe network configuration data. NETCONF and YANG provide a powerful and flexible framework for managing network devices. More information on YANG will be provided in section 2.8.

The parts of the NETCONF protocol can be seen in figure 2.6. "The NETCONF protocol can be conceptually partitioned into four layers:

- The Content layer consists of configuration data and notification data.
- The Operations layer defines a set of base protocol operations to retrieve and edit the configuration data.
- The Messages layer provides a mechanism for encoding remote procedure calls (RPCs) and notifications.
- The Secure Transport layer provides a secure and reliable transport of messages between a client and a server." [17]

In the second figure 2.7 can then be seen encapsulation of the mentioned layers as an SSH connection is used.

NETCONF protocol is already in use for some parts of the configuration of CESNET3 network devices. That provides me with a basic framework that I will extend. The protocol will be used for the configuration of Model-Driven Telemetry on devices.

[13] Even though NETCONF has been known since the year 2006, I still consider it as a new approach since many providers still didn't fully adopt it and many configuration changes are done by CLI.

Noteworthy to mention in this context is the RESTCONF protocol. NETCONF and RESTCONF are two protocols used for managing network devices. The very simplified description of RESTCONF is that it's a NETCONF protocol, but instead of SSH as transport, it uses HTTP or HTTPS [14] [5, pp. 190].

While both protocols are designed to provide a programmatic way of configuring network devices, they differ in several key ways. NETCONF uses XML to format messages between a client and server, whereas RESTCONF uses JSON. Although both protocols are used for managing network devices, they are often used in different contexts. NETCONF is typically used in traditional network environments, while RESTCONF is often used in cloud-based environments that rely heavily on APIs and microservices.

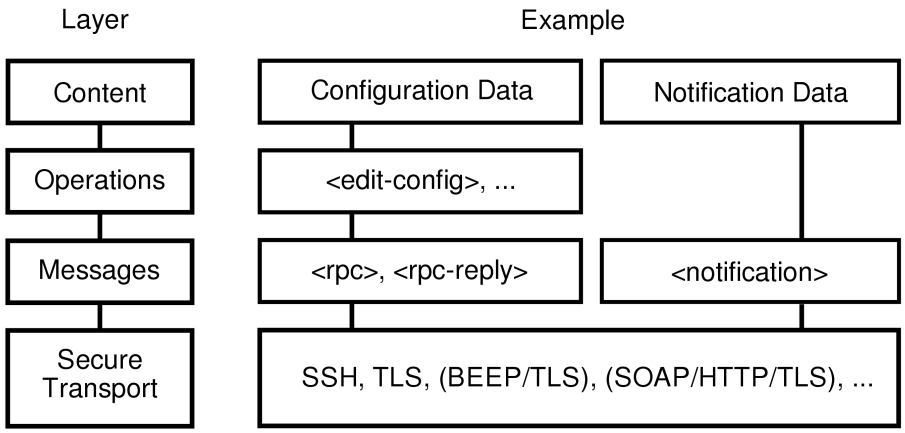


Figure 2.6: NETCONF protocol layers [17]

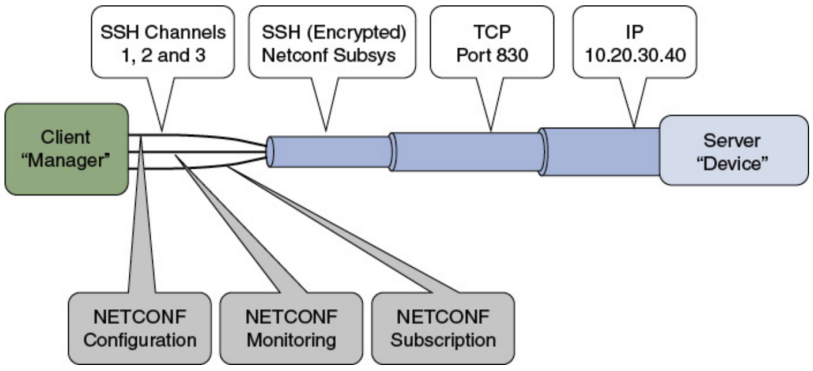


Figure 2.7: NETCONF Client-Server connection [5, Fig 4-2, pp. 162]

[14] Hypertext Transfer Protocol Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP). It uses encryption for secure communication over a computer network and is widely used on the Internet [12].

2.7 gRPC Network Management Interface

If we are discussing new approaches to the management of networks, one technology cannot be forgotten. That technology is gNMI. The gNMI represents gRPC Network Management Interface and is similar to NETCONF and RESTCONF regarding network configuration.

The gNMI is defined only by the OpenConfig community in comparison to NETCONF, with the definition in RFC by IETF. It is an open-source protocol developed by Google. The gNMI relies heavily on gRPC.

The gRPC is an open-source, high-performance remote procedure call (RPC) framework originally developed by Google. It uses Protocol Buffers as the data serialization format and provides a simple and efficient way to connect and call methods on remote servers across distributed systems. One of the key advantages of gRPC is its performance. Because it uses binary serialization and HTTP/2, it is more efficient than traditional web APIs that use text-based formats like JSON or XML. This means that gRPC can handle a higher volume of requests and responses with lower latency, making it ideal for building high-performance and scalable applications.

The gNMI interface utilizes YANG models as well. It consists of a few similar operations to the ones offered by NETCONF/RESTCONF. The options from gNMI: CapabilityRequest, GetRequest, SetRequest, and SubscribeRequest correspond to hello, get/get-config, and edit-config from NETCONF [5, pp. 215]. With regards to Model-Driven Telemetry, gNMI is primarily used in dial-in mode. Dial-in mode is when the collector initiates a TCP connection to the network device. The opposite method is called dial-out.

2.8 YANG models

YANG is a modeling language used for describing data models and management protocols for network devices. It stands for "Yet Another Next Generation" and was developed by the IETF to define data models for network configuration and management in RFC 6020 [3].

YANG is a human-readable, machine-parsable language that allows network administrators to define the data structure used to configure and manage network devices. The structure of the YANG definition file can be seen on listing 2.1 [16, pp. 21].

YANG models consist of a hierarchical tree structure of data elements that define the components and attributes of network devices, such as interfaces, IP addresses, routing protocols, and other network services. The hierarchy of the YANG model *Cisco-IOS-XR-pfi-im-cmd-oper* can be seen in figure 2.2 [16, pp. 33].

Overall, the YANG modeling language plays an essential role in network automation and orchestration, enabling the development of standardized and interoperable network management systems.

```
module Cisco-IOS-XR-pfi-im-cmd-oper {
  namespace "http://cisco.com/ns/yang/Cisco-IOS-XR-pfi-im-cmd-oper";
  prefix pfi-im-cmd-oper;

  import Cisco-IOS-XR-types {
    prefix xr;
  }
  include Cisco-IOS-XR-pfi-im-cmd-oper-sub2 {
    revision-date 2017-06-26;
  }
  ...
  organization
    "Cisco Systems, Inc.";
  contact
    ...
  description
   ...
  revision 2017-06-26 {
    description
      "Change identifiers to be more readable.";

   ...

  container interfaces {
    config false;
    description
      "Interface operational data";
    container interface-xr {
      list interface {
        key "interface-name";
        leaf interface-name {
          type xr:Interface-name;
        }
      }
    }
    ...
  }
}
```

Listing 2.1: YANG module Cisco-IOS-XR-pfi-im-cmd-oper

```
module: Cisco-IOS-XR-pfi-im-cmd-oper
  +--ro interfaces
     +--ro interface-xr
        +--ro interface* [interface-name]
           +--ro interface-name xr:Interface-name
           +--ro mac-address
           | ...
           +--ro arp-information
           | ...
           +--ro ip-information
           | ...
           +--ro encapsulation-information
           | ...
           +--ro interface-type-information
           | ...
           +--ro data-rates
           | ...
           +--ro interface-statistics
           | ...
           +--ro l2-interface-statistics
           | ...
           +--ro interface-handle? string
           +--ro interface-type? string
           +--ro state? Im-state-enum
           +--ro line-state? Im-state-enum
           +--ro encapsulation? string
           +--ro encapsulation-type-string? string
           +--ro mtu? uint32
           +--ro last-state-transition-time? uint32
           +--ro speed? uint32
           +--ro crc-length? uint32
           +--ro duplexity? Im-attr-duplex
           +--ro bandwidth? uint32
           +--ro max-bandwidth? uint32
           +--ro keepalive? uint32
           +--ro parent-interface-name? xr:Interface-name
           +--ro loopback-configuration? Im-cmd-loopback-enum
           +--ro description? string
           +--ro transport-mode? Im-attr-transport-mode
           +--ro fast-shutdown? boolean
           +--ro if-index? uint32
```

Listing 2.2: Tree view of YANG module Cisco-IOS-XR-pfi-im-cmd-oper

2.9 Virtualization tools

Since the most probable usage of the system will be that every network device will have its collector instance, it's necessary to explore viable options. Similar

tools in this context are often named Docker, Kubernetes, or LXC.

Docker is OS-level virtualization software that allows one to install and deploy multiple containers [7]. Containers are lightweight, portable, and isolated environments that contain all the necessary components to run an application, including the application code, libraries, and dependencies. These containers are enclosed from the host OS and allow connection through several interfaces.

Kubernetes is a container orchestration platform that automates containerized applications' deployment, scaling, and management. It provides a way to manage and coordinate many containers across multiple nodes, ensuring that the suitable containers run on the right nodes at the right time.

LXC is a lightweight containerization technology that provides a system-level interface for managing containers.

In summary, Docker is a containerization platform that allows developers to create and manage containers, Kubernetes is a container orchestration platform that provides advanced features to automate the deployment and management of containers at scale, and LXC is a lightweight containerization technology that provides a system-level interface for managing containers. Each technology has its own strengths and weaknesses, and the choice of which one to use depends on the specific use case and requirements.

Part of my bachelor thesis was testing the deployment of Model-Driven Telemetry collector in Docker containers compared to hosting it directly on a server. Testing proved that using Docker mostly doesn't matter performance-wise, so I've decided to use it for collectors in this monitoring system. Also, I'm more familiar with its structure and configuration.

2.10 Database and collector selection

Basically, every monitoring system needs a database to function properly. Requirements are always to store and visualize data with its history. The best way is to use a time series database for this purpose.

A time series database is a specialized database designed to store and retrieve data organized by time. In a time series database, data is typically stored in a series of rows, with each row representing a specific time period. Each row may contain multiple data points, such as temperature readings or data throughput, and these data points are associated with a particular timestamp.

One of the key benefits of using a time series database is that it is optimized for time-based queries, allowing for efficient and fast retrieval of data over specific time periods. Time series databases also often include specialized features such as data compression and automatic data aggregation, which can help to reduce storage costs and improve performance. Some popular time series databases include InfluxDB, Elasticsearch, and OpenTSDB.

Analysis of different databases was also done in my bachelor thesis [16, Ch. 3]. The comparison was made between InfluxDB and Elasticsearch. Both are popular time-series databases used for storing and analyzing time-series

data. While both databases are designed for similar use cases, they have some significant differences in terms of their architecture, query language, and features.

The choice between InfluxDB and Elasticsearch will depend on the specific needs of the project. If we are primarily dealing with time-series data, InfluxDB may be the better choice, while Elasticsearch may be a better fit if we need to search and analyze a wide variety of data types.

InfluxDB has been chosen in this particular case as the system will be dealing with a large quantity of time-series data. Also, InfluxDB was used and tested during the bachelor thesis, so I'm familiar with its configuration.

For the data collector tool, two options were considered. I analyzed both in my bachelor thesis. Since then, Pipeline development has been stopped, so it is not a viable option. The other option is Telegraf.

Telegraf is a plugin-driven server agent for collecting and reporting metrics and data from various sources. It is part of the TICK (Telegraf, InfluxDB, Chronograf, Kapacitor) stack, which is an open-source platform for processing time-series data. Integration with InfluxDB is then simple. Telegraf has a wide range of built-in input plugins that allow it to collect data from various sources, including system metrics, logs, databases, and network devices. It includes MDT and gNMI plugins developed in cooperation with Cisco.

Chosen tool for the role of collector is Telegraf. Due to using it during my bachelor thesis an fulfilling all requirements it seems as best option.

2.11 Visualization tools

The significant task from a user experience standpoint to consider is data visualization. This part of the system was one of the longest in testing, and its development is the most dynamic. Since my bachelor thesis, I tested several options and tried to find the best solution. Finally, the most probable choices ended up being Grafana, Kibana, and Nagios.

Grafana is an open-source software platform that allows users to visualize and analyze data in real-time. It provides a way to create interactive, customizable dashboards that display data from various sources such as databases, web services, and other applications.

With Grafana, users can create panels that display data in various formats, including graphs, tables, and heatmaps. Users can also configure alerts to notify them when certain conditions are met, and they can create annotations to provide context to their data. Although alerting option is built in Grafana, after some testing, I decided that it was not usable for our needs.

Similar to Grafana is Kibana software. Kibana is part of the Elastic Stack, which includes Elasticsearch, Logstash, and Beats, and offers integrations with other tools in the stack. Regarding visualization, both tools are pretty similar and provide comparable tools. Grafana has a more user-friendly interface that is designed for non-technical users, whereas Kibana's interface is more geared toward data analysts and developers.

Overall, Grafana is better suited for general-purpose data visualization and analysis, while Kibana is more focused on log analysis and monitoring.

Although Nagios is a bit different type of visualization from the other two, I will explain why I did incorporate it in these options. Grafana and Kibana are both data visualization and analysis platforms for real-time data. On the other hand, Nagios is best used for displaying alerts. As I already mentioned, the handling of alerts in Grafana was insufficient. Because of this, I decided to split visualization into two parts. One will display graphs and long-term statistics, and the other will handle alerts. Alternatives for Nagios are also plenty. In this regard, Nagios has been chosen as an already deployed and functioning system in CESNET that currently displays alerts from servers and old network devices.

2.12 Data processing

After completing testing in my bachelor thesis, a question concerning data normalization arose. Even though YANG models are well-defined, they sometimes differ from different devices. Also, if we are about to combine MDT with SNMP protocol, some data normalization would be beneficial. For this part, several options were considered. The possibilities are:

- Use built-in processors in Telegraf.
- Use Apache Kafka system.
- Write Python script to normalize data.

According to documentation Apache Kafka could be used for this purpose, but I didn't have the opportunity to test it. It's one of the things to consider testing with the system, but it probably wouldn't be part of the system from the start. Apache Kafka is a comprehensive software and requires a lot of testing.

Another option is to use the internal processors of the Telegraf collector. This solution was tried on a testing system deployed during my bachelor thesis. It could be used, but it also needs more testing, as it is unclear what the collector's performance will be in this scenario. During regular operation, Telegraf only does a little processing with incoming data, but with this plugin turned on, the hardware requirements need to be clarified. With this data processing Telegraf can become a bottleneck of the system.

The last option I tried was implementing a Python script to normalize data. The details of the script will be shown in a later chapter, but after evaluation, I've decided not to progress with this solution at this moment. This solution doesn't have any performance concerns but development concerns. As I would be the single developer of this script, it would create obstacles in the future for maintenance and development.

2.13 Monitoring systems

As a part of my research, I've tried to find currently available systems that utilize MDT for monitoring. But as I've described in section 2.3, this term can still be quite vague. Also, some vendors of these monitoring systems claim they are using "telemetry" to monitor, but this can even mean SNMP, as described previously. The problem with these claims is that I've very little chance to verify which form of telemetry they are using, as often the systems are closed and paid.

What I've managed to verify is the monitoring system developed by Cisco. During the Cisco Live conference in February 2023, I had a chance to attend a lab focused on this system. The solution consisted of some similar parts as will be part of my monitoring system. Figure 2.8 shows that visualization and database have used the same tools: Grafana and InfluxDB. The difference is in the collector and automation tools. Cisco uses its own Crosswork Platform Infrastructure and Crosswork Data Gateway. This solution provides GUI to automate data collection through Crosswork Infrastructure. It's definitely a viable and functional option. Deployment through GUI was relatively easy. This solution also provides API for more machine-based deployment. But its huge drawback is the price. Cisco doesn't post any pricing publicly, but based on discussions with the engineer leading the lab, this solution started to make viable sense from tens of thousands of monitored devices in the network. Considering that it is only part of the solution, where it's still needed to maintain the database and create views in Grafana, which isn't a small task, it has been rejected CESNET networking team.

Another solution that claims to use telemetry is Spotlight - Network Telemetry System [25]. Unfortunately, I didn't find any documentation for this tool, so the only information is from their website. But based on this quote: "An in-house designed analysis engine contacts devices on a round-robin basis using a wide variety of protocols including, but not limited to, SNMP, HTTP API, SSH." it's doubtful that this system uses Model-Driven Telemetry (MDT) as is the goal of my system.

The system which seems to be working with required telemetry is SigNoz [2]. According to the documentation, this tool uses a gRPC call to collect data. But it doesn't fulfill the second part of the defined Model-Driven Telemetry term. It doesn't work with Yet another next generation (YANG) models, and the tool is not designed to communicate with network devices.

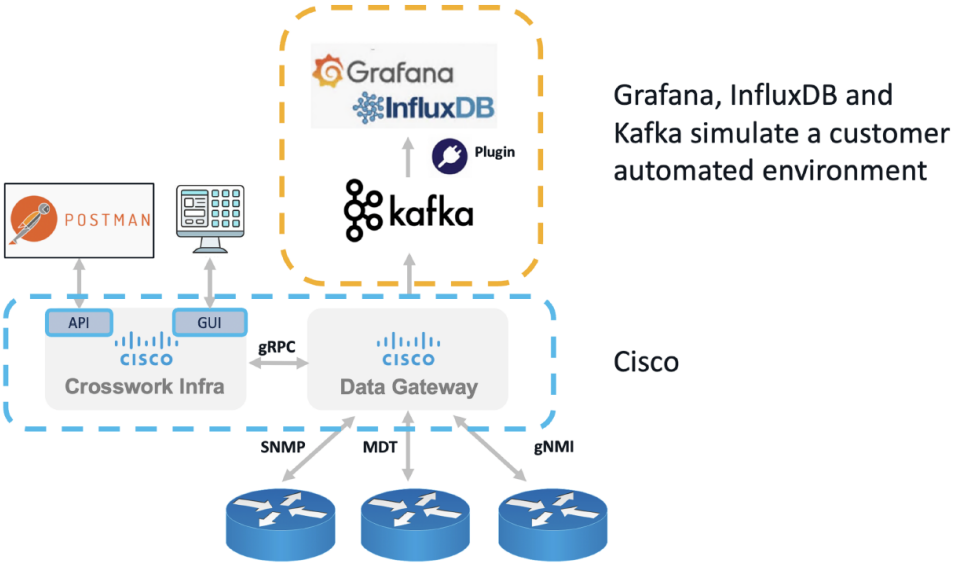


Figure 2.8: Architecture of Cisco lab

Chapter 3

Design

3.1 System requirements

The first task to do in system design is to specify requirements that the system should be capable of meeting. For simplicity, I've created an item list that can be later checked in the testing part.

- Collector selection
  - ☐ MDT Dial-in capable
  - ☐ MDT Dial-out capable
  - ☐ SNMP capable
  - ☐ Database interoperability
- Data retention
  - ☐ Archive at least half-year-old data
  - ☐ Database must have backups
- Security
  - ☐ Collector can accept data only from registered devices
  - ☐ All publicly exposed parts must have appropriate firewall rules
- Logging
  - ☐ All parts must send logs to defined points
- Resiliency to failure
  - ☐ System must retain some monitoring capability even with some parts in shutdown

The collector selection part was partially discussed in 2.10 section. The collector must be able to collect data using Model-Driven Telemetry dial-in

and dial-out methods. This requirement is essential as the whole system is based on this technology. The SNMP requirement is due to backward compatibility. Some of the old network devices are not MDT capable and will remain in the network.

The data retention requirement is for a database. Firstly it defines the need for a database, as theoretically, monitoring can be done without memory, but that system would be useless. Secondly, according to some regulations, CESNET must store data for a defined time. Another requirement is to backup the database to a remote location in case of failure.

The system's security is a significant part as a portion of the system will be exposed on public IP addresses. The system will be storing sensitive data about clients and need to be secured accordingly. The setup of a firewall on all nodes is vital. Also, all system management will be done from a secure network and only using public key authentication SSH protocol.

The logging requirement comes from internal CESNET regulations. Part of the infrastructure is the Syslog server on which all systems are required to send logs.

Resiliency to failure is essential in order to minimize monitoring availability. When part of the system fails, the overall functionality should be minimal. In case of collector failure, it's probable to lose some data, but the whole system shouldn't be affected. In conclusion, some system parts should be duplicates, doing the same tasks.

3.2 Current monitoring systems in CESNET

I did research about currently deployed monitoring systems during my bachelor thesis. I included two systems that perform monitoring of network devices. These systems were HP Network Node Manager and G3.

HP Network Node Manager (NNM) is a network management tool developed by Hewlett-Packard that provides real-time monitoring, fault detection, and device configuration management capabilities for complex, multi-vendor networks. NNM uses SNMP to discover and monitor network devices such as routers, switches, servers, and printers. NNM includes features such as event correlation, root-cause analysis, and threshold-based alerting to help network administrators quickly identify and respond to network issues.

The currently deployed version is quite obsolete. It was decided that due to the high cost, the newer version isn't going to be purchased.

Another described system was G3. The G3 is an internally developed system also based on SNMP. The purpose of this system isn't to provide network alerts but to provide long-term statistics. The development of this system wasn't wholly abandoned, but the leading developer moved to other projects. As a result, it was decided to leave this development altogether and replace this system.

The following monitoring system currently deployed in CESNET to monitor is Nagios. I've already mentioned this system as it's planned to utilize it for visualization of alerts from the Model-Driven Telemetry monitoring system.

As part of the tender for network modernization, Cisco Evolved Programmable Network Manager was bought. EPNM is a network management tool that provides end-to-end network visibility and control for multi-vendor, multi-domain networks. It is designed to simplify network management tasks and improve operational efficiency. The most important part of this system in a modernized network will be the management and provisioning of the DWDM network. This system also provides monitoring options. But after testing, it was decided it doesn't meet the criteria for replacement of both G3 and HP NNM. EPNM will be used as a provisioning tool but not as a monitoring tool. Also, this system lacks the ability to collect data using Model-Driven Telemetry.

3.3 System design

The Model-Driven Telemetry monitoring system will comprise of several parts. The simplest overview of the system can be seen in figure 3.1. The system has three main parts: collectors, database, and visualization. Another additional part is management.

During design, it was necessary to think ahead about possible upgrades or significant system changes. Therefore the system basically has two branches. The production and development branches are deployed on different machines. The visualization of these branches is in figure 3.2. The production servers are dark blue, and the development servers are orange. The figure shows that database, visualization, and management have their development part, but the collectors don't. The reason for this is based on the collector's design.

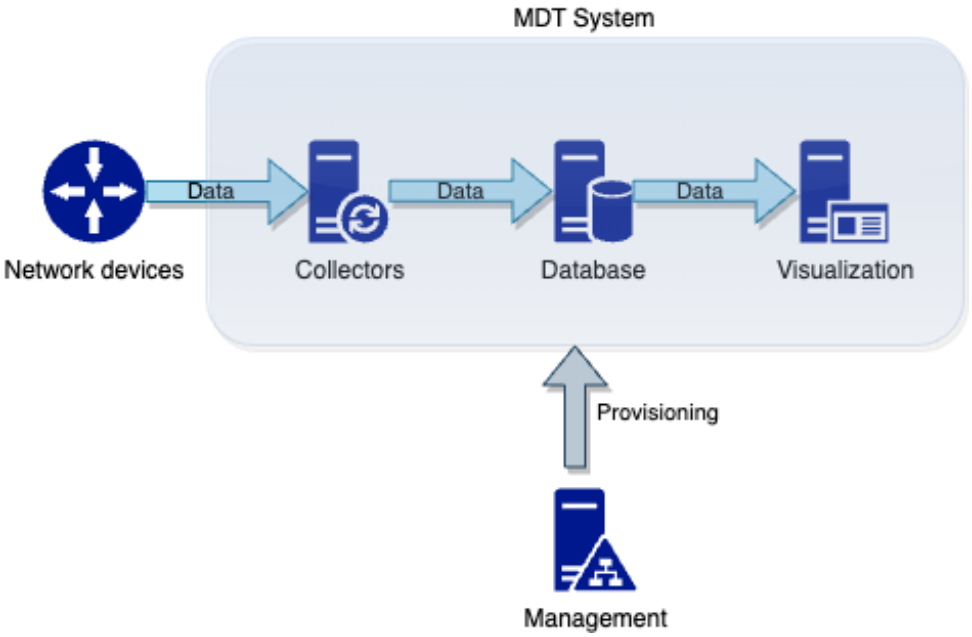


Figure 3.1: Basic overview of Model-Driven Telemetry monitoring

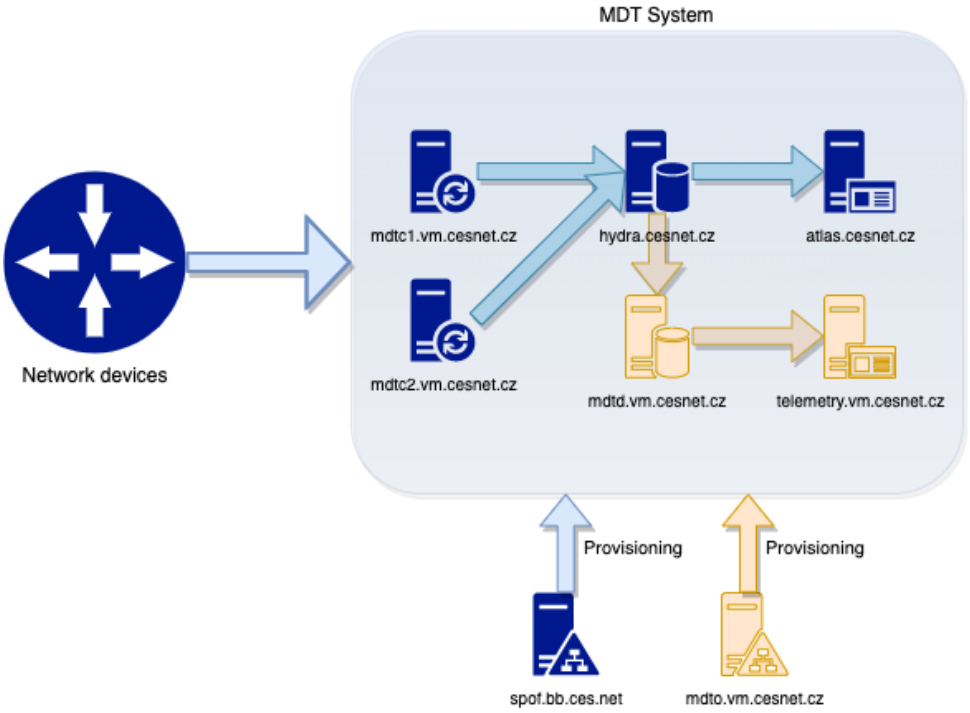


Figure 3.2: Production and development parts of system

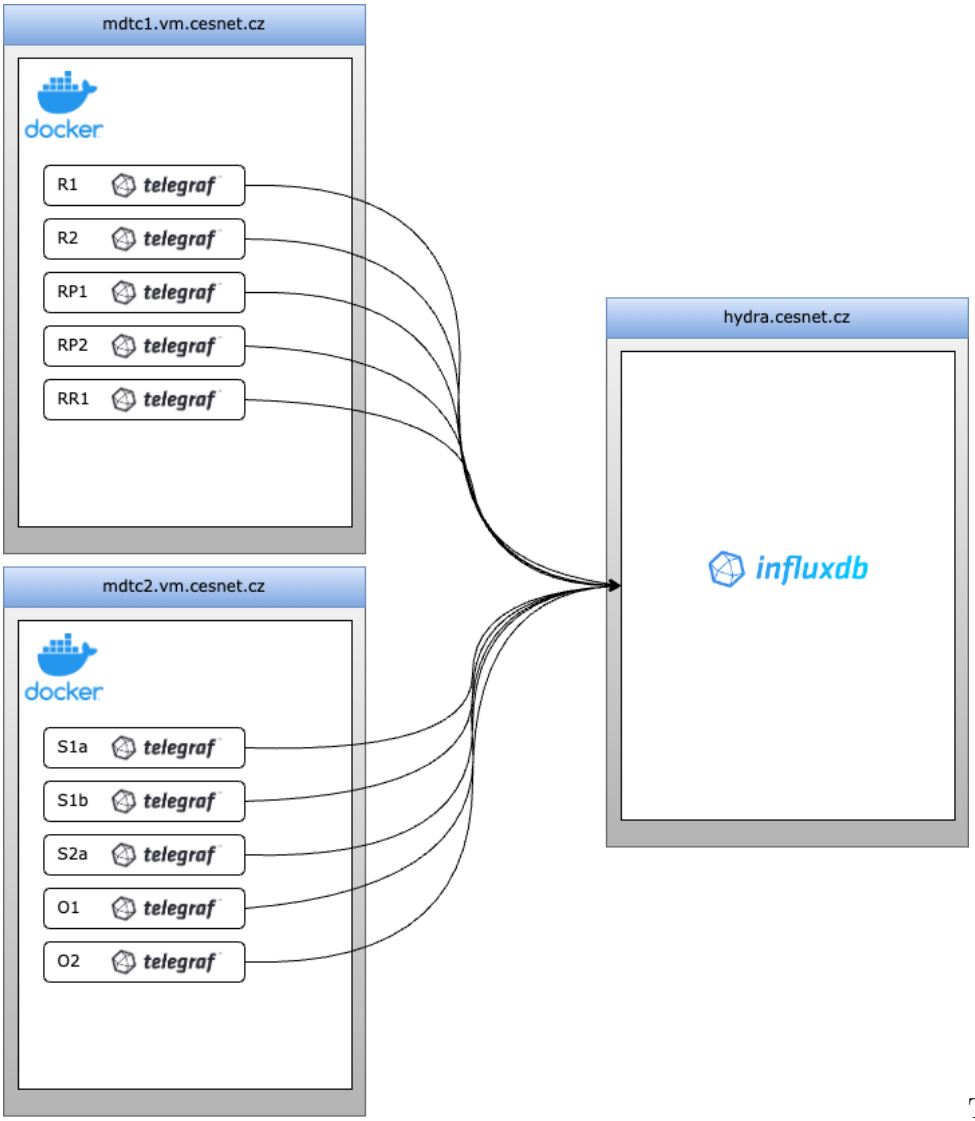
3.3.1 Collector design

The open-source Telegraf has been chosen as the collectors software based on analysis. The collectors will need to handle additional data processing. Every network device will have its own Telegraf instance to satisfy this condition. These instances will be deployed as Docker containers for simplicity of design and management. The design can be seen in figure 3.3.

Due to security reasons and the fact that all of the CESNET3 network has management interfaces on the private network, which is not routed globally, two collectors have been deployed. The mdtc1.vm.cesnet.cz is routed publically and can be accessed globally via its IPv4 address. The mdtc2.vm.cesnet.cz is routed only locally and accessible only in the management network. The MDTC1 is supposed to collect data from older devices that don't have access to the private network. MDTC2 collects data from new devices in the CESNET3 via the management network.

To satisfy reliability conditions, both MDTC1 and MDTC2 are virtual machines located in the Czech Education and Scientific NETwork virtualization platform. This platform is geographically distinct, and virtual machines are automatically migrated to the following location if one location fails.

Since Telegraf instances are deployed in Docker, there is no need to deploy a testing collector. Updating Docker doesn't break its containers, and all other parts of the collectors are deployed using Ansible. Also, testing newer versions of Telegraf can be done by deploying another Docker container.



Th

Figure 3.3: Visualization of collectors

3.3.2 Database design

For the database, the InfluxDB software has been chosen. The database will also be deployed in a virtualization platform, the same as collectors. In order to provide an upgrade of the database without problems, a development branch will be deployed along with the production one.

The production database hydra.cesnet.cz will store all data from production Telegraf instances deployed on MDTC1 and MDTC2 into CESNET3 bucket. The development database mdto.vm.cesnet.cz will collect data from testing Telegraf instances into the special testing bucket.

The reliability conditions should be satisfied again by automatically migrating the virtualization platform. To provide safety for data, an everyday backup of the database will be made, and also, a weekly backup of the

database from HYDRA to MDTO will be made. This will safeguard data in case of unexpected system failure.

The data retention will be handled yearly. The database has the option to retain data in buckets for a limited period of time. For this purpose, 365 days have been chosen. Older data will be available in the form of database backup, probably in aggregated form.

3.3.3 Visualization design

Grafana will handle statistics about devices and their interfaces. The virtual machine atlas.cesnet.cz will be deployed as production. Development visualization will be on server telemetry.vm.cesnet.cz. The main reason for the development part is to have a place for creating new dashboards and upgrade testing of Grafana.

In the first version of the system, there will be two dashboards. One displays statistics about device interfaces, and the second shows the device status itself.

The alert from devices will be handled by the Nagios system. In CESNET, there is already deployed Nagios system for monitoring of servers. Only additional probes will be added.

3.3.4 Management design

The management part will have several components. Netbox as the source of data. Two virtual servers with Ansible for configuration deployment.

Netbox itself already has two instances done.cesnet.cz and netbox.done-test.cesnet.cz. The done.cesnet.cz will serve as the data source for production. The main pieces of information for Ansible will be:

- Device name
- Device location
- Device source IPv4 address
- IPv4 address of the collector
- TCP port number of the collector

Virtual servers for Ansible are also in development and production. The server spof.bb.ces.net will serve as a production branch. On this server, there is also deployed NETCONF management of network devices. The development branch is on server mdto.vm.cesnet.cz. This machine is unable to configure network devices, as the NETCONF Ansible configuration deployment is managed by another team that has its own testing machine.

The part of management could be considered the syslog servers and data storage. Every server in the CESNET infrastructure must send its internal logs to a syslog server. The MDT monitoring system will send syslog data to vinovago.cesnet.cz and zarovka.cesnet.cz. The database backup destination for long-term use will be on CESNET storage department hardware.

Chapter 4

Implementation

4.1 Configuration of network devices

The first parts of the pipeline of the monitoring system are the configurations of network devices. As was already designed, this part will be handled by NETCONF protocol. I have utilized the already existing NETCONF Ansible repository [15] , which does have the ability to configure network devices. Part of the implementation was to extend the repository with configuration specifically for Model-Driven Telemetry.

Configuration of devices with Cisco IOS-XR software has three parts: Destination-group, Sensor-group, and Subscription. Used YANG model for configuration is *Cisco-IOS-XR-um-telemetry-model-driven-cfg*.

Reference configuration of Destination-group can be seen in listing 4.1. The template file is relatively long as NETCONF configuration uses XML modeling language. In the example, I've marked the important parts that are parametrized. Values: *telemetry.server_name*, *telemetry.server_ip*, *telemetry.server_port* and *telemetry.protocol* are collected for every device from Netbox. This configuration sets to which collector will the network device send MDT data. Encoding of data is selected *self-describing-gpb* [16].

[15] This repository is located on an internal CESNET Gitlab server.

[16] Different options for data collection were described in my bachelor thesis [16, pp. 19].

```
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <telemetry xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-
      telemetry-model-driven-cfg">
   <model-driven>
    <destination-groups>
     <destination-group>
      <destination-group-id>
       {{ telemetry.server_name }}/
      </destination-group-id>
      <destinations>
       <destination>
        <destination-name>
         {{ telemetry.server_ip }}/
        </destination-name>
        <port-number>
         {{ telemetry.server_port }}/
        </port-number>
        <encoding>
         <self-describing-gpb/>
        </encoding>
        <protocol>
         <{{ telemetry.protocol }}/>
        </protocol>
       </destination>
      </destinations>
     </destination-group>
    </destination-groups>
   </model-driven>
  </telemetry>
```

Listing 4.1: Model-Driven Telemetry configuration teplate for IOS-XR - Destination-groups

The next part of the configuration is Sensor-group. Sensor-group is used internally on network devices to select which data should be collected and sent to the collector. Configuration template can be seen on 4.2. This section defines which YANG models should be used. On IOS XR is limited to a maximum of 16 sensor-paths in a single *sensor-group-id*. Due to this restriction, the configuration of this part is not dynamic but statically set. The automation of this functionality is planned to be implemented in further versions of the monitoring system, and YANG models will be selected based on Netbox.

Used YANG models on IOS XR devices can be seen in table 4.1. Most of these models are filtered for particular fields. For example, in *Cisco-IOS-XR-pfi-im-cmd-oper*, only operational-status, admin-status, and MTU are used.

Following table 4.2 shows possible modules for expansion in the future. Most of them are focused on routing. The reason for not including them right away

is that, for example, *Cisco-IOS-XR-ip-rib-ipv4-oper* and *Cisco-IOS-XR-ip-rib-ipv6-oper* contain whole routing table, that can be pretty large on peering routers. Including these models will require more testing and will take longer time. Another interesting data can be *Cisco-IOS-XR-platform-inventory-oper*, which could serve for automatic inventarization.

```
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <telemetry xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-
      telemetry-model-driven-cfg">
   <model-driven>
    <sensor-groups>
     <sensor-group>
      <sensor-group-id>XR-ifstatus</sensor-group-id>
      <sensor-paths>
       <sensor-path>
        <sensor-path-id>Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/
            interface-xr/interface</sensor-path-id>
       </sensor-path>
      </sensor-paths>
     </sensor-group>
    </sensor-groups>
   </model-driven>
  </telemetry>
```

Listing 4.2: Model-Driven Telemetry configuration teplate for IOS-XR - Sensor-groups

Model	Description
Cisco-IOS-XR-pfi-im-cmd-oper	Interface information (status, admin-status)
Cisco-IOS-XR-infra-statsd-oper	Interface statistics
Cisco-IOS-XR-wdsysmon-fd-oper	CPU utilization
Cisco-IOS-XR-nto-misc-oper	RAM utilization
Cisco-IOS-XR-ipv4-io-oper	Interface IPv4 information
Cisco-IOS-XR-ipv6-ma-oper	Interface IPv6 information
Cisco-IOS-XR-telemetry-model-driven-oper	Model-Driven Telemetry status
Cisco-IOS-XR-controller-optics-oper	Interface optical data

Table 4.1: Used YANG models IOS XR

The last part of the configuration is Subscription. The subscription part merges Destination-groups and Sensor-groups together. This configuration can be seen in listing 4.3. Parts of this configuration are templatized, but most of them are static. In the listing is shown only part of the subscriptions. The important part is the *sample-interval*. This parameter sets in milliseconds difference between data collections. If we set this to zero, then the subscription

Model	Description
Cisco-IOS-XR-wd-oper	Watchdog information
Cisco-IOS-XR-segment-routing-srv6-oper	Segment Routing with IPv6 dataplane
Cisco-IOS-XR-policy-repository-oper	Routing policy operational data
Cisco-IOS-XR-platform-inventory-oper	Inventory operational data
Cisco-IOS-XR-mpls-vpn-oper	L3VPN operational data
Cisco-IOS-XR-l2vpn-oper	L2vpn operational data
Cisco-IOS-XR-evpn-oper	EVPN operational data
Cisco-IOS-XR-ip-rib-ipv4-oper	IPv4 RIB operational data
Cisco-IOS-XR-ip-rib-ipv4-oper	IPv6 RIB operational data

Table 4.2: Possible expansion of YANG models IOS XR

will work on change [17]. For this purpose, two subscriptions were configured.

An interesting part of a subscription is *source-interface*. The initial intention was to use a management interface that is in a special VRF [18]. But during testing, I came across a probable bug on Cisco ASR 9903. If I configured the router to send data in this management VRF, the router didn't send anything. The same configuration on Cisco NCS 540, which is also IOS XR, is behaving as expected and sends data to the collector in the management VRF. This seems to be a bug and has been reported to Cisco engineers. As a workaround to this problem, I decided to send data from all IOS XR devices in global VRF, which is working. The result is that the collector for all IOS XR devices is MDTC1 instead of MDTC2.

[17] Diferences between these two options were described in my bachelor thesis [16, pp. 12-13].

[18] Virtual Routing and Forwarding is a technology used in computer networking to create separate virtual routing tables within a single physical router or switch. Each VRF instance is isolated from other VRF instances and has its own unique routing table, allowing multiple virtual networks to coexist on the same physical infrastructure.

```
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
  <telemetry xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-
      telemetry-model-driven-cfg">
    <subscriptions>
     <subscription>
      <subscription-id>101</subscription-id>
      <sensor-groups>
       <sensor-group>
        <sensor-group-id>XR-ifstatus</sensor-group-id>
        <sample-interval>30000</sample-interval>
       </sensor-group>
      </sensor-groups>
      <destinations>
       <destination>
        <destination-id>{{ telemetry.server_name }}</destination-id>
       </destination>
      </destinations>
      <source-interface>MgmtEth0/RP0/CPU0/0</source-interface>
     </subscription>
    </subscriptions>
   </model-driven>
  </telemetry>
```

Listing 4.3: Model-Driven Telemetry configuration teplate for IOS-XR - Subscriptions

The Cisco IOS-XE configuration is somewhat more straightforward 4.4. All needed parts are in a single *<subscription>* block. A slight drawback is worse readability, but this language is designed for devices and not to be human-readable. The next drawback is that IOS-XE doesn't allow configuring multiple *sensor-group* in comparison with IOS-XR. It can be bypassed by configuring multiple *<subscription>*.

Parts of the file are templatized. The *<receivers>* block is taken from Netbox. Most of the IOS-XE devices are collected at the MDTC2 collector that is located in the management network. The *<source-vrf>* has to be configured to *Mgmt-vrf* to ensure connection.

Used YANG models can be seen in table 4.3:

Model	Description
Cisco-IOS-XE-interfaces	Interface information (status, admin-status)
Cisco-IOS-XE-process-cpu-oper	CPU utilization
Cisco-IOS-XE-memory-oper	RAM utilization
Cisco-IOS-XE-mdt-oper	Model-Driven Telemetry status
Cisco-IOS-XE-transceiver-oper	Interface optical data

Table 4.3: Used YANG models IOS XE

Every model is collected using two subscriptions [19]. First, collects data in predefined intervals and second on change. Based on the use of IOS-XE devices in the CESNET3 network as aggregation switches, there is no need to collect any data on routing.

```
<config xmlns:xc="urn:ietf:params:xml:ns:netconf:base:1.0">
 <subscription-config xmlns="urn:ietf:params:xml:ns:yang:ietf-event-
     notifications">
    <subscription operation="replace">
      <subscription-id>101</subscription-id>
      <stream xmlns:yp="urn:ietf:params:xml:ns:yang:ietf-yang-push">
          yp:yang-push</stream>
      <encoding xmlns:cyp="urn:cisco:params:xml:ns:yang:cisco-xe-ietf-
          yang-push-ext">cyp:encode-kvgpb</encoding>
      <xpath-filter xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push
          ">/interfaces-ios-xe-oper:interfaces/interfaces-ios-xe-oper:
          interface</xpath-filter>
      <receivers>
        <receiver>
          <address>{{ telemetry.server }}</address>
          <port>{{ telemetry.port }}</port>
          <protocol xmlns:cyp="urn:cisco:params:xml:ns:yang:cisco-xe-
              ietf-yang-push-ext">cyp:grpc-tcp</protocol>
        </receiver>
      </receivers>
      <source-vrf>Mgmt-vrf</source-vrf>
      <period xmlns="urn:ietf:params:xml:ns:yang:ietf-yang-push
          ">6000</period>
    </subscription>
 </subscription-config>
</config>
```

Listing 4.4: Model-Driven Telemetry configuration teplate for IOS-XE

4.2 Collector setup

The collector design is based on Docker containers. The first task is to install and configure Docker on collector nodes. Installation is relatively straightforward, and instructions are available on many websites based on the desired host machine. Since Docker has no unique configuration in my system, I won't go into any more details.

The next task of devices setup was to secure access to them. The obvious thing is to deny access to nodes through SSH password and allow only public key authentication. Another part is to set up a basic firewall. Since both nodes have host OS Ubuntu 22.04.2 LTS, UFW (Uncomplicated Firewall) package was already installed. Although some administrators would have reservations

[19] Both in format X01 and X02, where X is replaced with numbers.

about using UFW and argue for using another firewall like *nftables*, I've decided that for collector use, UFW is sufficient. The important advantage is its easy management through Ansible.

The security of nodes was easier to do on MDTC2 since this node has a private address (IPv4: 10.7.1.11) and is already located behind the firewall. The collector MDTC1 has a public address (IPv4: 78.128.211.217 and IPv6: 2001:718:1:1f:50:56ff:feee:217) and is potentially more vulnerable. Despite this, security on both nodes is handled similarly. The rules of the firewall are divided into global and Ansible parts. The global part allows traffic only on TCP port 22 from specific address ranges. The second part is handled by the management node of the system through Ansible.

To ensure consistency of installed software packages and more straightforward updates, *etckeeper* was installed on all nodes. It enables a simple rollback to the previous software version in case of a problem during an update.

Based on the design, data collection is handled by Telegraf containers. Every network device has its own instance. This enables the pass of tags from network documentation done by Netbox to the monitoring system. The configuration of each instance is parameterized. The Telegraf configuration file is a template that is being filled by the Ansible playbook.

In listing 4.5 are shown variables setup of tags for each device. The purpose of these tags is to filter devices in the monitoring system the same way as in Netbox. For example, *site* tag enables filtering devices on the same location.

Next part of configuration 4.6 is the connection to the database InfluxDB. Access to the database is done using a token which is also passed from the Ansible playbook.

The last part of the configuration is focused on data collection. In listing 4.7 can be seen Model-Driven Telemetry collecting Telegraf. As transport protocol *grpc* is used. This parameter is static since all MDT-capable devices in the CESNET3 network can use it. The only variable is *service_address* which defines the TCP port the collector should run on. The remaining part is aliases which are entirely voluntary. I've decided to use them to rename the original YANG models to a more understandable form.

The configuration of SNMP is also included 4.8. The monitoring system will also collect data from old network devices, which is part of long-term development. The system is ready for this monitoring, but Model-Driven Telemetry monitors all CESNET3 devices.

```
[global_tags]
  deviceid = "{{item.key}}"
  site = "{{item.value.site.name}}"
  siteid = "{{item.value.site.id}}"
  device_role = "{{item.value.device_role.name}}"
  device_type = "{{item.value.device_type.model}}"
  device_manufacturer = "{{item.value.device_type.manufacturer.name
      }}"
  status = "{{item.value.status.value}}"
```

Listing 4.5: Telegraf configuration - Tags

```
[[outputs.influxdb_v2]]
  urls = ["https://mdtd.vm.cesnet.cz:8086"]
  token = "{{ database_token }}"
  organization = "CESNET"
  bucket = "CESNET3"
```

Listing 4.6: Telegraf configuration - InfluxDB

```
# # Cisco model-driven telemetry (MDT) input plugin for IOS XR, IOS
    XE and NX-OS platforms
 [[inputs.cisco_telemetry_mdt]]
  transport = "grpc"
# ## Address and port to host telemetry listener
  service_address = ":{{item.value.custom_fields.telemetry_port}}"

   [inputs.cisco_telemetry_mdt.aliases]
    if-counters = "Cisco-IOS-XR-infra-statsd-oper:infra-statistics/
        interfaces/interface/latest/generic-counters"
    ipv4-stats = "Cisco-IOS-XR-ipv4-io-oper:ipv4-network/nodes/node/
        interface-data/vrfs/vrf/details/detail"
    ipv6-stats = "Cisco-IOS-XR-ipv6-ma-oper:ipv6-network/nodes/node/
        interface-data/vrfs/vrf/details/detail"
    if-status = "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-
        xr/interface"
    optics-info = "Cisco-IOS-XR-controller-optics-oper:optics-oper/
        optics-ports/optics-port/optics-info"
    cisco-cpu = "Cisco-IOS-XR-wdsysmon-fd-oper:system-monitoring/cpu-
        utilization"
    cisco-memory = "Cisco-IOS-XR-nto-misc-shmem-oper:memory-summary/
        nodes/node/detail"
```

Listing 4.7: Telegraf configuration - Model-Driven Telemetry

```
 # Retrieves SNMP values from remote agents
 [[inputs.snmp]]
   agents = ["udp://195.113.144.70:161"]
# ## Timeout for each request.
    timeout = "300s"
# ## SNMP version; can be 1, 2, or 3.
    version = 3
# ## SNMPv3 authentication and encryption options.
    sec_name = ":{{ snmp_user }}"
    auth_password = ":{{ snmp_passwd }}"

  [[inputs.snmp.table]]
   name = "snmp"
   oid = "IF-MIB::ifXTable"

   [[inputs.snmp.table.field]]
     name = "ifName"
     oid = "IF-MIB::ifName"
     is_tag = true
```

Listing 4.8: Telegraf configuration - SNMP

4.3 Ansible management setup

In order to make system management automated as much as possible, Ansible software has been used. Ansible deploys configuration based on prewritten playbooks.

As this playbook is distributed via SSH protocol, a management node had to be chosen. For this purpose, I decided to use the existing virtual server spof.bb.ces.net, which already has Ansible installed and has access to both network devices and Netbox API. Ansible playbooks that configure devices automatically are deployed on this management server, as described in the previous chapter. This server is hidden behind a firewall, and members of the networking department in CESNET manage security.

Ansible playbook consists of tasks that are making described actions. The playbook to manage the networking system consists of several of these tasks. The first part handles fetching data from Netbox system 4.9. This task makes a query to Netbox API with a proper access token that has been deleted from the listing for security reasons. According to filters, in this example, based on tag *cesnet3* fetches data and saves them into dictionary variable using *set_fact* directive. The task consists of other keywords, such as *delegate_to*, that overwrite the destination of execution of the playbook. In this case, to localhost since collectors don't have a connection to Netbox API. Keyword *when* is used in most tasks. It ensures that the task will be done only on the proper collector. Otherwise, task would be done on both collectors and

applied only on desired. This would result with Telegraf containers on all collector nodes for every device.

The next part of the pipeline done by the playbook is creating Telegraf templates 4.10. The template file has been mostly described in the previous chapter. This task only fills missing values in the curly brackets of the template and saves configs to the desired location. A noteworthy part is *backup: true*, which keeps older configuration files with timestamps if they have been changed. This helps with possible error tracebacks.

The critical task is creating Telegraf containers on the desired collector. It exposes the port of the container to the host machine. Netbox defines this port, and Telegraf receives data from network devices on it. Containers are set always to restart.

The last part of the playbook is setting of firewall on colletors 4.12. As was described, colletors have minimum exposed ports for better security. To be able to collect data, it needs to listen at least on some ports. Based on set IP addresses and ports in Netbox, ports are open to specific devices. Thanks to that, no other device can send data to a particular collector who isn't the supposed receiver.

Currently, the script must be initiated manually, but the plan is to enhance Netbox with a simple button that would trigger this playbook. Nevertheless, in the meantime, to automate it, the Ansible playbook is started every night with Cron job. This ensures the deployment of a new device to monitoring even if the administrator forgets to run the playbook after installing the new device.

```
tasks:
# query a list of devices
- name: Obtain list of devices from NetBox
  delegate_to: localhost
  set_fact:
    netbox: "{{ netbox + [item] }}"
  loop: "{{ query('netbox.netbox.nb_lookup',
                  'devices',
                  api_endpoint='https://done.cesnet.cz',
                  api_filter='tag=cesnet3',
                  token='') }}"
  when: " item.value.custom_fields.telemetry_server == hostvars[
      inventory_hostname]['ansible_env'].SSH_CONNECTION.split(' ')[2]
       "
  loop_control:
    label: "{{ item.value.name }}"
```

Listing 4.9: Ansible playbook - Netbox data

```
tasks:
- name: Generate configuration files for telegraf instances
  template:
    src: telegraf-template.j2
    dest: "/root/configs/telegraf.{{ item.value.name }}.conf"
    backup: true
    mode: 0640
  with_items:
  - "{{ netbox }}"
  when: " item.value.custom_fields.telemetry_server == hostvars[
      inventory_hostname]['ansible_env'].SSH_CONNECTION.split(' ')[2]
       "
  loop_control:
    label: "{{ item.value.name }}"
```

Listing 4.10: Ansible playbook - Telegraf template

```
  tasks:
    # Create four containers on each managed host using default image
        and command.
    - name: Create default containers
      community.docker.docker_container:
        name: "{{ item.value.name }}"
        image: "{{ default_container_image }}"
        published_ports: "{{ item.value.custom_fields.telemetry_port
            }}:{{ item.value.custom_fields.telemetry_port }}"
        restart_policy: "always"
        volumes: "/root/configs/telegraf.{{ item.value.name }}.conf:/
            etc/telegraf/telegraf.conf:ro"
      with_items:
      - "{{ netbox }}"
      when: " item.value.custom_fields.telemetry_server == hostvars[
          inventory_hostname]['ansible_env'].SSH_CONNECTION.split(' ')
          [2] "
      loop_control:
        label: "{{ item.value.name }}"
```

Listing 4.11: Ansible playbook - Docker containers

```
tasks:
  - name: Allow ports for devices
    community.general.ufw:
      state: enabled
      rule: allow
      src: "{{ item.value.custom_fields.telemetry_source_ip }}"
      port: "{{ item.value.custom_fields.telemetry_port }}"
      proto: tcp
      comment: "CESNET3 {{ item.value.name }}"
      log: true
    with_items:
    - "{{ netbox }}"
    when: " item.value.custom_fields.telemetry_server == hostvars[
        inventory_hostname]['ansible_env'].SSH_CONNECTION.split(' ')
        [2] "
    loop_control:
      label: "{{ item.value.name }}"
```

Listing 4.12: Ansible playbook - UFW rules

4.4 Database setup

The setup of the database server was similar to other parts of the system. From a security standpoint firewall has been set up that allows communication only from the collectors. The addition to this was generating of a TLS certificate for the server. Since the communication between collectors and the database can be encrypted, the certificate has been generated and signed by the CESNET Certification Authority.

The InfluxDB is divided based on *organizations* and *buckets*. The organization can represent multitenant database use, where a bucket distinguishes data. *CESNET* organization was created with the bucket *CESNET3*. On the bucket, data retention of 365 days has been set. Any older data will be removed from the database. This setup is important in order to keep the database query reasonably long.

Another part of the database setup was to create simple scripts that are periodically run by Cron. The purpose of these scripts is to back up the database by command *influx backup*. The destinations of backups are two. One is on server mdtd.vm.cesnet.cz, which runs testing instance of the InfluxDB database. The other destination is in CESNET Storage Department, which serves as a long-term archive. The Cron job is run weekly.

The last part of the database setup was syslog logging to CESNET servers. Additional software *syslog-ng* was installed. The configuration isn't complicated 4.13. The exact setup is used on the collector nodes as well as management and visualization nodes.

```
# Configure the remote destination
destination d_net { tcp("vinovago.cesnet.cz"
                      port(514)
                      tls(
                             key_file("/etc/ssl/tcs/sectigo-key.pem")
                             cert_file("/etc/ssl/tcs/sectigo-cert.pem
                                 ")
                             ca_dir("/etc/ssl/trusted_ca")
                             trusted_dn("CN=vinovago.cesnet.cz, *")
                      )
                      flush_lines(100)
                      flush_timeout(5000)
                    );
                };
```

Listing 4.13: Syslog configuration

4.5 Visualization setup

The configuration of visualization consisted of several parts. First was a basic server setup similar to other servers of the system. The *etckeeper*, *syslog-ng*, and firewall tools.

Another important part was generating certificates. As this is the entire system's front end, which will be accessed from the web browser, the appropriate certificates need to be created to ensure secure HTTPS traffic. For this purpose, I have used the CESNET TCS certificate, which is validated by GEANT Vereniging 4.1.

The primary system to install for visualization was Grafana. Installation guides for various host systems are available, so I won't describe it as no deviations were made.

The only thing to set up was the authentication of users. As this system is available to the world, simple authentication must be made. Since the free version of Grafana is used, there is no option to handle users by Grafana itself. One option would be to create user accounts locally, but this solution is not scalable. Instead, Grafana was setup to display dashboards without any user authentication, an Apache2 software was put in front of Grafana. The Apache2 server runs on the same server on TCP ports 80 (HTTP) and 443 (HTTPS). The Apache2 proxy setup consists of two parts.

The first is for protocol HTTP 4.14. The configuration is simple as this part only redirects all requests to HTTPS.

The second part is for HTTPS protocol 4.15. The *OIDC* options are configured with hidden parameters. This ensures that anyone with a CESNET identity is able to see Grafana dashboards. Next is the configuration of *Proxy*. As Grafana natively runs on TCP port 3000, Apache redirects accepted users to this port. The last part is the setup of encryption for HTTPS protocol.

The keywords *SSL* are used as the location for server private keys.

The Grafana setup is mostly default. An important part was the setup of dashboards. For the first version of the system, two dashboards were made. These dashboards are still relatively simple as more focus was made on the whole system's design. Additionally, this will require more work in the following versions based on user feedback.

The first dashboard displays simple interface data 4.2. The details about interface *description*,*IPv4 address*, *IPv6 address*, *optical wavelength*, and *optical power* can be seen. Selection of interfaces can be made by defined variables *Site*, *Device*, and *Interface*. These variables are queries into the same database and allow users to navigate to the correct interface.

The second dashboard monitors state data about the devices themselves 4.3. On this dashboard, we can see statistics about used RAM memory and CPU usage in the last five seconds.

The last part of the visualization is the Nagios setup. This system is managed by other workers in CESNET, and this system only provides access to the InfluxDB database. I won't describe the setup as it wasn't done by me.

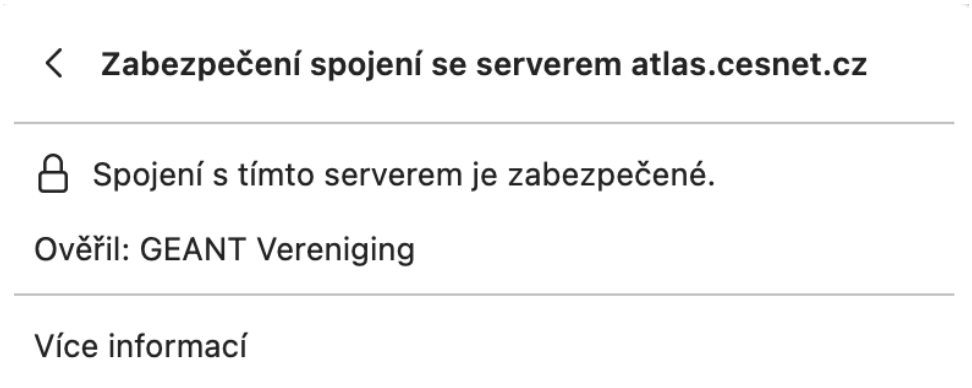


Figure 4.1: Visualization certificate

```
<VirtualHost *:80>
       ServerName atlas.cesnet.cz
       Redirect permanent / https://atlas.cesnet.cz
</VirtualHost>
```

Listing 4.14: Apache2 - HTTP setup

```
<VirtualHost _default_:443>
        ServerName atlas.cesnet.cz
        DocumentRoot /var/www/html

        OIDCProviderMetadataURL https://login.cesnet.cz/oidc/.well-
            known/openid-configuration
        OIDCClientID <OIDC_client_id>
        OIDCClientSecret <OIDC_secret>
        OIDCScope "openid profile email eduperson_entitlement"
        OIDCRedirectURI https://atlas.cesnet.cz/oauth2callback

        OIDCCryptoPassphrase <OIDC_crypto_passphrase>

        OIDCResponseType code
        OIDCPassClaimsAs environment
        OIDCSessionInactivityTimeout 3600
        OIDCSessionMaxDuration 86400
<Proxy *>
        AuthType openid-connect
        Require all denied
        Require valid-user
        Require claim "preferred_username~.+"
        Options FollowSymLinks
        AllowOverride None
</Proxy>
        ProxyPreserveHost On
        ProxyPass "/.well-known" !
        ProxyPass / http://localhost:3000/
        ProxyPassReverse / http://localhost:3000/

        SSLEngine on
        SSLProtocol All -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
        SSLHonorCipherOrder On
        SSLCertificateFile </path/to/servercert>.pem
        SSLCertificateKeyFile </path/to/serverkey>.key
</VirtualHost>
```

Listing 4.15: Apache2 - HTTPS setup

4.6 Data rewriting

Although YANG data models are often represented as future data models for monitoring and configuring network devices, they still have some disadvantages. One disadvantage is the immense differences between models, which should collect the same data.

For example, this system collects data about the interface status from the network. This collection is done on several types of Cisco devices. Even though

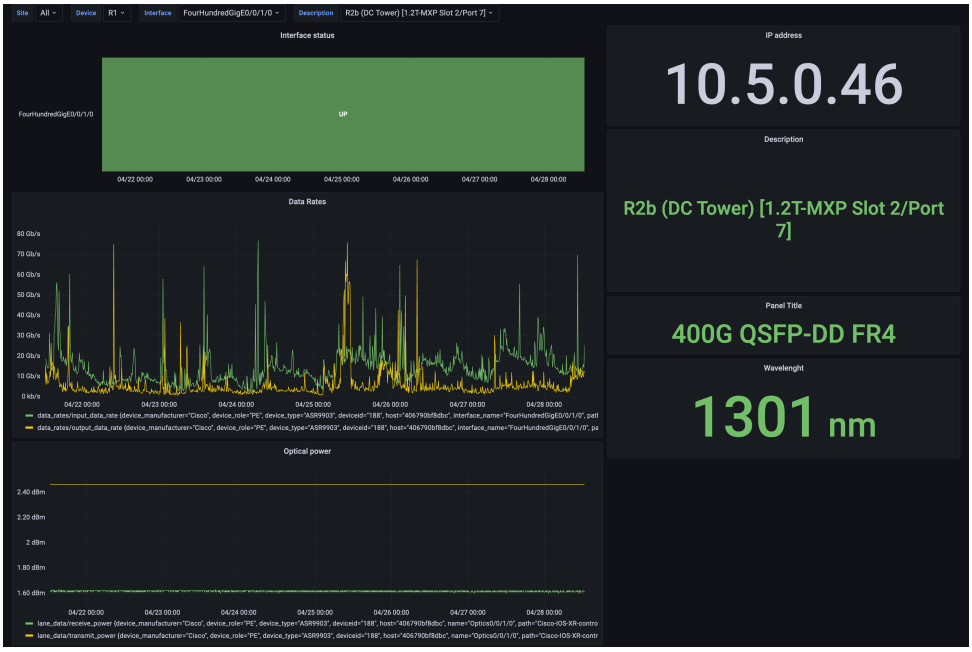


Figure 4.2: Interface dashboard in Grafana

Figure 4.3: Device dashboard in Grafana

the vendor is the same, the operating systems are different. In one case, it is IOS XR, and in another, IOS XE. This wouldn't be that much of a problem if the YANG models were similar. But in the case of IOS XR, I'm using "Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface" model, and in the case of IOS XE, "Cisco-IOS-XE-interfaces-oper:interfaces/interface". Again even different paths are understandable, but the problem starts with data tags and values that are sent from the device. One example is that IOS XR uses as tag *interface_name*, but the IOS XE model uses only *name* for

interface name. Other mismatches are plenty. Some of them are not much of a problem as they can be easily hidden by using the correct database query. In any case, these complicate queries, and it is better to normalize data to the same data structure.

4.6.1 Python scripts

During analysis, several options were considered. Firstly I will describe the option that has been researched the longest.

The need for data normalization was obvious during testing in my bachelor thesis. For this purpose, I've started to develop a Python script that would be able to process incoming data. The script was divided into two main parts. The *init_convertor.py* and *data_convertor.py*.

The first part is *init_convertor.py*. The development of this script began before the Netbox was used as the primary source of truth for the network. Its purpose was then dual. One was to create a template file for each YANG data model that served as a controller for data processing. The second was to create a list of devices that should be monitored. The design was that a single Telegraf was supposed to run on TCP port 57000 with the following configuration 4.16. The main difference was that the *exec* as output was utilized instead of using the *InfluxDB* plugin.

```
# OUTPUT PLUGINS #
# # Send metrics to command as input over stdin
[[outputs.exec]]
   command = ["/etc/telegraf/data_convertor/init_convertor.py"]
   data_format = "influx"
# SERVICE INPUT PLUGINS #
 [[inputs.cisco_telemetry_mdt]]
  transport = "tcp"
  service_address = ":57000"
```

Listing 4.16: Telegraf configuration - Python script

The noteworthy parts of this script are the following functions 4.17. The function *fill_model_file* creates a template file for each YANG model. The result of this function is the template file 4.18. Generally, the function takes every tag and field from the YANG model measurement and generates a template. It consists of several keywords that are later used by *data_convertor.py*. It sets for each value *action*, *rename*, and *translation_table*. The actions were:

- NONE - remove this value
- KEEP - keep value in an unchanged state
- ADD - adds new value
- MODIFY - modify the value based on subsequent actions

Options *rename* and *translation_table* are used only with the keyword *MODIFY*. One enables to rename tag or field as is seen with tag *source*. The other one enables to remap of some values. An example is mapping of *state* from string representation to an integer.

The second function *generate_source_list* was supposed to be a list of monitored devices, but this functionality has been replaced with the Netbox system.

The next part of Python scripts to rewrite data was the script to convert incoming data and store them to InfluxDB. The source files of this script are attached to this thesis. I won't show any parts as these scripts are lengthy, and all essential functionalities were already described. These scripts work based on template files generated by *init_convertor.py*.

The advantages of this solution were its universality and that it was designed specifically for the use case in CESNET. Unfortunately, disadvantages were a bigger problem for production implementation. As I was the single developer of this scripts and there wasn't a possibility of adding more developers, the decision to use something simpler was made. Development of a program such as this requires a larger team of developers, and development by a single programmer would make it a great single point of failure in the concept of the whole system.

```
def fill_model_file (self, measurement_name):
    logging.info("Writing config file for " + measurement_name + ".")
    with open(MODEL_PATH + measurement_name, "w") as file:
        file.write("measurement_input: " + measurement_name + "\n")
        file.write("measurement_output: " + "\n")
        file.write("\n")
        file.write("translation_tables: " + "\n")
        file.write(" " + "basic_translation_table: &
            basic_translation_table" + "\n")
        file.write(" " + "tmp: tmp_value" + "\n")
        file.write("\n")
        file.write("tags: " + "\n")
        for key in self.data_input.get('tags').items():
            file.write(" " + key[0] + ":\n")
            file.write(" " + "action: " + "KEEP" + "\n")
            file.write(" " + "rename: " + "~" + "\n")
            file.write(" " + "translation_table: " + "{}" + "\n")
        file.write("fields: " + "\n")
        for key in self.data_input.get('fields').items():
            file.write(" " + key[0] + ":\n")
            file.write(" " + "action: " + "KEEP" + "\n")
            file.write(" " + "rename: " + "~" + "\n")
            file.write(" " + "translation_table: " + "{}" + "\n")

def generate_source_list(self):
    source = self.data_input.get('tags').get('source')
    file_check(SOURCE_PATH)
    with open(SOURCE_PATH, "r+") as file:
        for line in file:
            if source in line:
                logging.info(source + " already exist in source_list
                    file. Skipping.")
                break
        else: # not found, we are at the eof
            file.write(source) # append missing data
            file.write("\n")
            logging.info("Writing " + source + " in source_list file
                .")
```

Listing 4.17: Python script - Init configuration

```
measurement_input: Cisco-IOS-XR-pfi-im-cmd-oper-interfaces-interface-
    briefs-interface-brief.yaml
measurement_output: interface-brief

translation_tables:
  basic_translation_table: &basic_translation_table
    tmp: tmp_value
tags:
  subscription:
    action: KEEP
    rename: ~
    translation_table: {}
  source:
    action: MODIFY
    rename: "router"
    translation_table: {}
fields:
  bandwidth:
    action: KEEP
    rename: ~
    translation_table: {}
  state:
    action: MODIFY
    rename: ~
    translation_table:
      "im-state-down": 0
      "im-state-up": 1
```

Listing 4.18: YANG template file

4.6.2 Telegraf

Another option was to use Telegraf's internal processing. This option was decided to be a good enough option. Telegraf processor plugins are well-documented and relatively simple. Though they don't offer as many options as custom scripts, they are more viable in long-term sustainability.

To return to the initial problem. Telegraf processor can resolve this situation by using the following configuration 4.19. We can normalize the *name* tag between several models using rename functionality. The Telegraf provides much more processing, and more of them are used in the monitoring system.

```
# PROCESSOR PLUGINS #
[[processors.rename]]
  namepass=["if-status-XE"]
  [[processors.rename.replace]]
    tag = "name"
    dest = "interface_name"
```

Listing 4.19: Telegraf - Processor plugin

Chapter 5

Testing

5.1 Fulfilling system requirements

The first part of testing the whole system was a resolution of system requirements from section 3.1.

- Collector selection
  - ☑ MDTDial-in capable
  - ☑ MDTDial-out capable
  - ☑ SNMP capable
  - ☑ Database interoperability
- Data retention
  - ☑ Archive at least half-year-old data
  - ☑ Database must have backups
- Security
  - ☑ Collector can accept data only from registered devices
  - ☑ All publicly exposed parts must have appropriate firewall rules
  - ☑ Visualization must be accessible via HTTPS
- Logging
  - ☑ All parts must send logs to defined points
- Resiliency to failure
  - ☑ System must retain some monitoring capability even with some parts in shutdown

5.1.1 Collector selection

The first section of the requirements was the setup of the collector part. As the Telegraf software was chosen for the role of collector, all conditions were fulfilled.

The collector can initiate the Model-Driven Telemetry Dial-in setup. Although this setup is not used in the current configuration of the monitoring system, it's possible, and for purposes of this thesis, have been tested. The input plugin *gNMI (gRPC Network Management Interface)* needs to be used on the Telegraf part. Following the documentation on the Cisco guide and enabling gNMI on one device, I was able to test it successfully. The enabled gNMI interface on the device then displays the following 5.1. The configuration on the device is simply about executing commands 5.1.

```
gnxi
gnxi server
gnxi port 57500
```

Listing 5.1: Cisco IOS-XE gNMI configuration

Next was the MDT Dial-out setup. This configuration is used and described in the implementation sections. The check of functioning Model-Driven Telemetry subscription on IOS-XR can be used command 5.2, which output is seen in figure 5.2. On this show command, it can be easily checked configuration from Ansible. The destination port is set to 57017, which corresponds with the port defined in Netbox 5.3.

```
sh telemetry model-driven destination MDTC1
```

Listing 5.2: Cisco IOS-XR check MDTsubscrition

Similar testing was done on the IOS-XE. The command 5.3 can be executed based on subscription number. The result is then following 5.4.

```
show telemetry ietf subscription 101 detail
```

Listing 5.3: Cisco IOS-XE check MDTsubscrition

The previous parts serve great to check if Model-Driven Telemetry is correctly configured on devices, but unfortunately not always show if any data are being sent. For this purpose, we can use the database itself. If the device sends data and collecting Telegraf receives data correctly, it will appear in InfluxDB. The command 5.4 will show some of the network devices that send data to the collector. In the moment of testing, the list consists of all devices set in Netbox 5.5.

```
S1b#sh gnxi state detail
Settings
========
  Server: Enabled
  Server port: 57500
  Secure server: Disabled
  Secure server port: 9339
  Secure client authentication: Disabled
  Secure trustpoint:
  Secure client trustpoint:
  Secure password authentication: Disabled

GNMI
====
  Admin state: Enabled
  Oper status: Up
  State: Provisioned

  gRPC Server
  -----------
    Admin state: Enabled
    Oper status: Up

  Configuration service
  ---------------------
    Admin state: Enabled
    Oper status: Up

  Telemetry service
  -----------------
    Admin state: Enabled
    Oper status: Up
```

Figure 5.1: IOS XE - show gnxi state detail

```
import "influxdata/influxdb/schema"

schema.tagValues(bucket: "CESNET3", tag: "source")
```

Listing 5.4: Cisco IOS-XE check MDTsubscrition

The *SNMP capable* point has been shown in Telegraf configuration 4.8. Unfortunately, any further testing wasn't done during the writing of this thesis as SNMP is no longer the preferred way of monitoring, and all new devices are monitored by Model-Driven Telemetry. Although many other

```
RP/0/RP0/CPU0:R1#sh telemetry model-driven destination MDTC1
Tue May  2 18:24:54.589 CEST
  Destination Group Id:  MDTC1
  --------------------
    Destination IP:       78.128.211.217
    Destination Port:     57017
    Subscription:         101
    State:                Active
    Encoding:             self-describing-gpb
    Transport:            grpc
    TLS :                 False
    Total bytes sent:     24932925959
    Total packets sent:   284822
    Last Sent time:       2023-05-02 18:24:51.1626529760 +0200
  Collection statistics:
    Maximum tokens                  : 4000
    Event tokens                    : 750
    Cadence tokens                  : 688
    Token processed at              : 2023-05-02 18:23:21.240504 +0200
    Cadence token advertised at     : 2023-05-02 18:23:21.241504 +0200
    Event token advertised at       : 2023-05-02 18:23:21.240504 +0200
    GNMI initial synchronization time:
    Pending queue size              : 0
    Pending queue memory size (bytes): 0
    Processed events                : 4
    Collection tokens               : 688

    Collection Groups:
    ------------------
      Id: 102
      Sample Interval:      30000 ms
      Heartbeat Interval:   NA
      Heartbeat always:     False
      Encoding:             self-describing-gpb
      Num of collection:    15395
      Incremental updates:  0
      Collection time:      Min:  1674 ms Max:  4298 ms
      Total time:           Min:  1678 ms Max:  4303 ms Avg:  2292 ms
      Total Deferred:       0
      Total Send Errors:    0
      Total Send Drops:     0
      Total Other Errors:   0
      No data Instances:    0
      Last Collection Start:2023-05-02 18:24:45.1620645760 +0200
      Last Collection End:  2023-05-02 18:24:47.1622999760 +0200
      Sensor Path:          Cisco-IOS-XR-pfi-im-cmd-oper:interfaces/interface-xr/interface
```

Figure 5.2: IOS XR - show telemetry model-driven destination MDTC1

users use Telegraf for monitoring using SNMP and guides are available. In that order, I consider this requirement as fulfilled.

The *Database interoperability* point was about the capability of writing data from the collector to the database. The functionality of this point has already been shown in previous paragraphs.

5.1.2 Data retention

Since time series databases can grow into enormous proportions, a data retention policy had to be made. The Influxdb itself provides data deletion after some specified time. For the monitoring system, I've chosen a data retention interval of 365 days 5.6. This setup should be sufficient to keep the

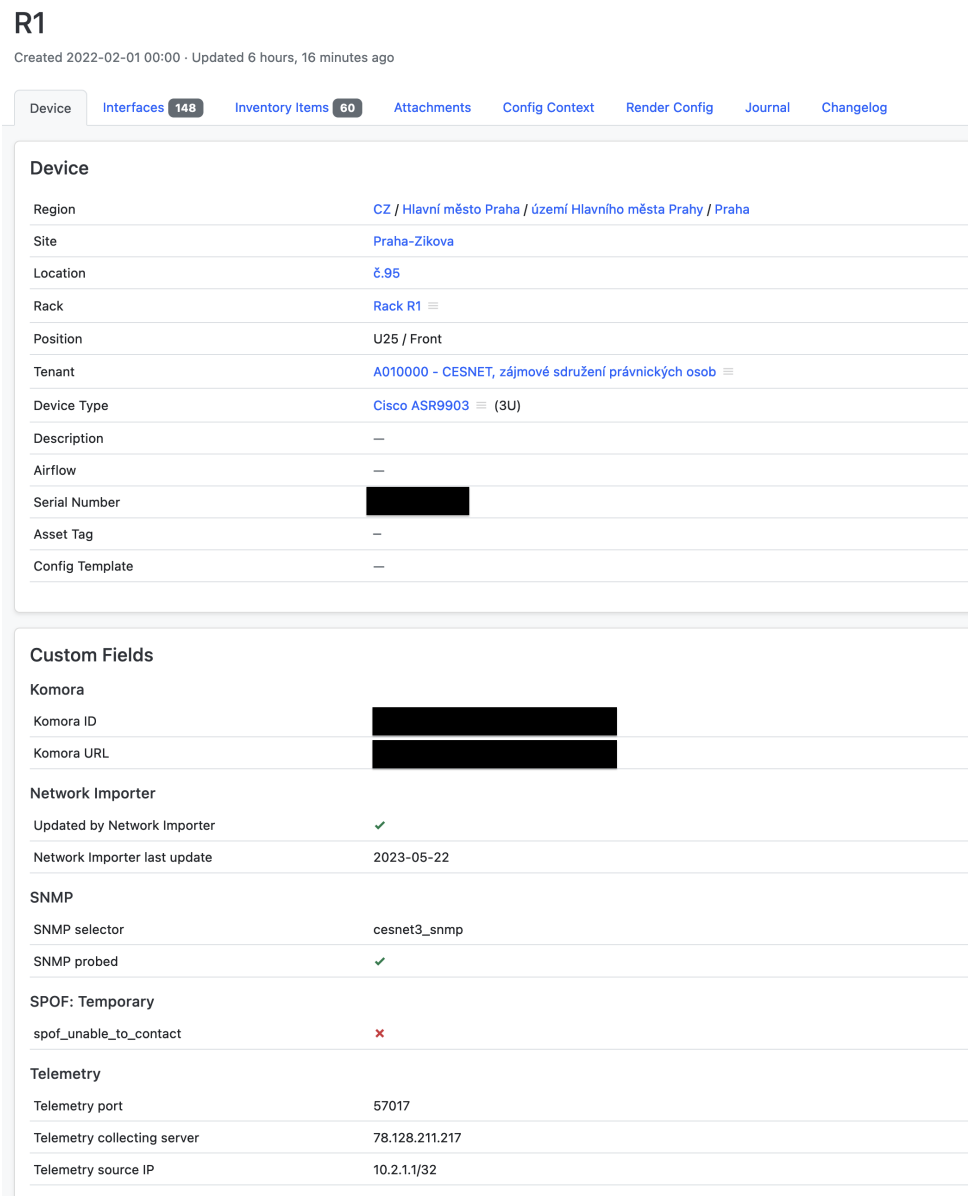


Figure 5.3: Netbox - router R1

database running smoothly and retain enough data.

Most of the time, data older than one year are useless. But a small percentage of use cases still exist when this data could be needed. In that order, a shell script was written to back up the database 5.5. This script is run by cron in specified intervals. It backups database locally and also sends data to CESNET Data storage for long-term archive. Thanks to this script point, *Archive at least half-year-old data* and *Database must have backups* can be checked. The archived data can then be simply recovered in case of unexpected problems on the virtualization platform by command *influx restore*.

```
S1a#sh telemetry ietf subscription 101 detail
Telemetry subscription detail:

  Subscription ID: 101
  Type: Configured
  State: Valid
  Stream: yang-push
  Filter:
    Filter type: xpath
    XPath: /interfaces-ios-xe-oper:interfaces/interfaces-ios-xe-oper:interface
  Update policy:
    Update Trigger: periodic
    Period: 3000
  Encoding: encode-kvgpb
  Source VRF: Mgmt-vrf
  Source Address:
  Notes:

  Legacy Receivers:
    Address                                 Port     Protocol        Protocol Profile
    ------------------------------------------------------------------------------------
    10.7.1.11                               57046    grpc-tcp
```

Figure 5.4: IOS XE - show telemetry ietf subscription 101 detail

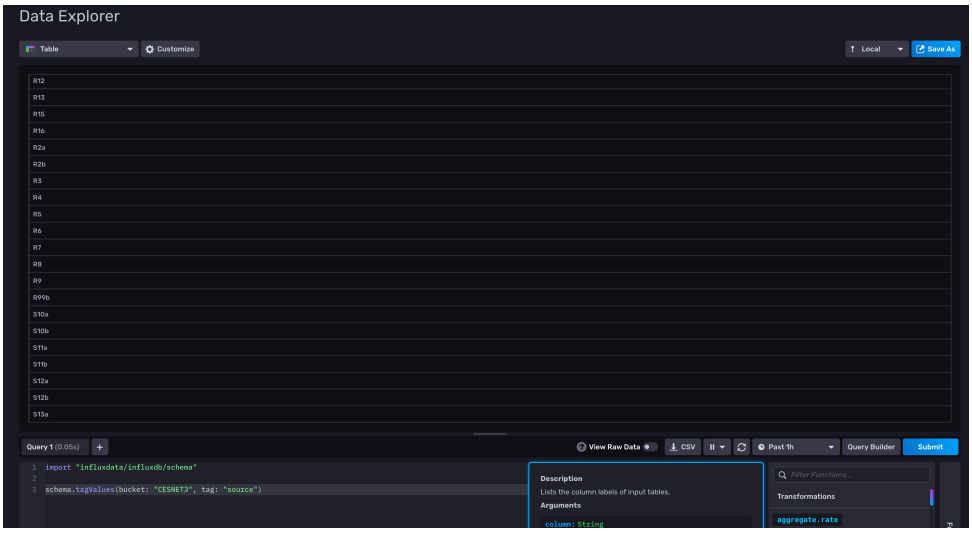


Figure 5.5: InfluxDB check sources

```
#!/bin/bash
DATE=$( date '+%F_%H:%M:%S' )
influx backup /var/db_backup/$DATE -t $DB_TOKEN
scp -r /var/db_backup/$DATE loub@ssh.du5.cesnet.cz:~/
```

Listing 5.5: Shell script to backup database

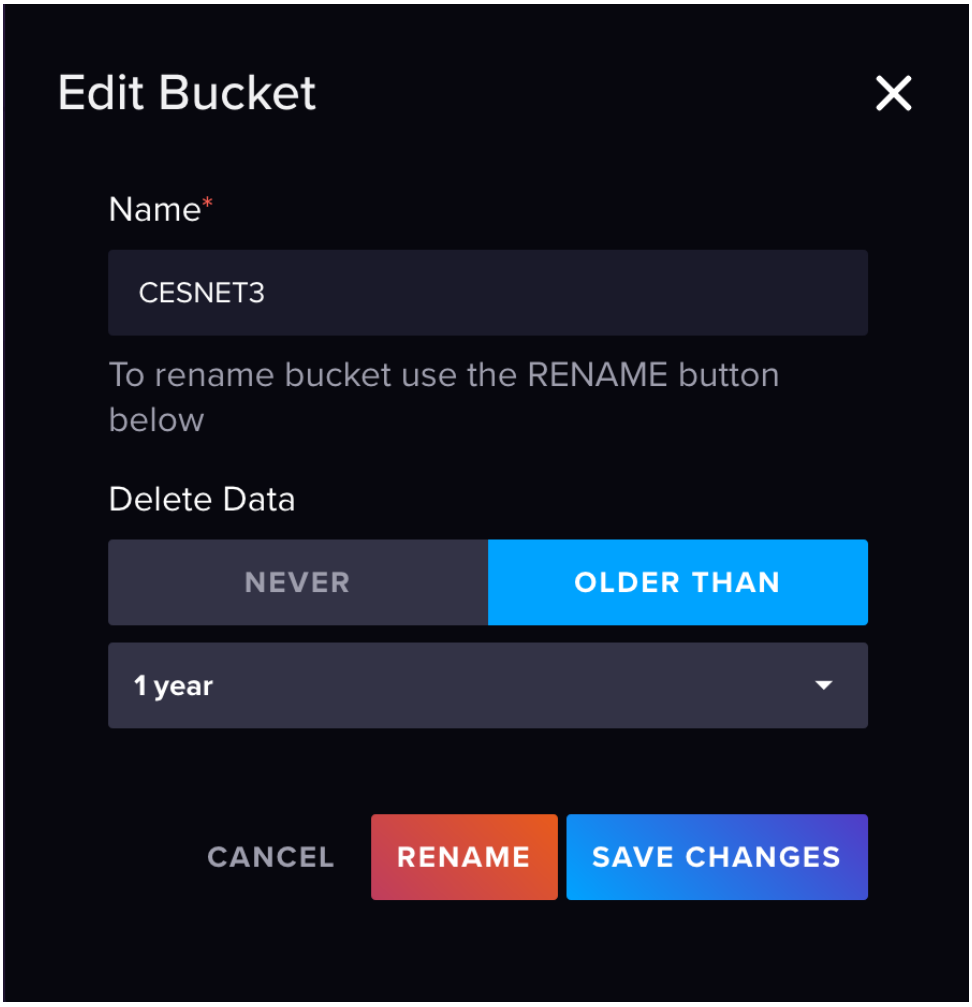


Figure 5.6: InfluxDB data retention

5.1.3 Security

The security section consisted of several tasks. The first one was that the collector would only accept data defined in Netbox. This was achieved by setting up of firewall. Figure 5.7 shows an example from one of the collectors. A specific TCP port accepts traffic only from one host IP address.

The next point was about setting firewall rules for all nodes. All show commands from nodes can be seen in the attachments. But mostly, this point was about securing the SSH port. On database to secure its endpoint TCP port 8086. The visualization was the only part that can have somewhat loose rules, but still, only CESNET IP addresses are allowed to access it.

The last part was about Grafana being exposed on TCP port 443 with the correct certificate. This has been proven in the Implementation section 4.5.

```
root@mdtc1:[ ~ ] $ ufw status
Status: active

To                         Action      From
--                         ------      ----
22                         ALLOW       195.113.222.128/25
22                         ALLOW       10.7.1.15
57017/tcp                  ALLOW       10.2.1.1                   (log) # CESNET3 R1
57018/tcp                  ALLOW       10.2.2.1                   (log) # CESNET3 R2a
57019/tcp                  ALLOW       10.2.2.2                   (log) # CESNET3 R2b
57020/tcp                  ALLOW       10.2.3.1                   (log) # CESNET3 R3
57021/tcp                  ALLOW       10.2.4.1                   (log) # CESNET3 R4
57022/tcp                  ALLOW       10.2.5.1                   (log) # CESNET3 R5
57023/tcp                  ALLOW       10.2.6.1                   (log) # CESNET3 R6
57024/tcp                  ALLOW       10.2.7.1                   (log) # CESNET3 R7
57025/tcp                  ALLOW       10.2.8.1                   (log) # CESNET3 R8
57026/tcp                  ALLOW       10.2.9.1                   (log) # CESNET3 R9
57027/tcp                  ALLOW       10.2.10.1                  (log) # CESNET3 R10
57028/tcp                  ALLOW       10.2.11.1                  (log) # CESNET3 R11a
57029/tcp                  ALLOW       10.2.11.2                  (log) # CESNET3 R11b
57030/tcp                  ALLOW       10.2.12.1                  (log) # CESNET3 R12
57031/tcp                  ALLOW       10.2.13.1                  (log) # CESNET3 R13
57032/tcp                  ALLOW       10.2.15.1                  (log) # CESNET3 R15
57033/tcp                  ALLOW       10.2.16.1                  (log) # CESNET3 R16
57034/tcp                  ALLOW       10.2.17.1                  (log) # CESNET3 R17
57035/tcp                  ALLOW       10.2.24.1                  (log) # CESNET3 R24
57036/tcp                  ALLOW       10.2.28.1                  (log) # CESNET3 R28
57037/tcp                  ALLOW       10.2.0.1                   (log) # CESNET3 RP1
57038/tcp                  ALLOW       10.2.0.2                   (log) # CESNET3 RP2
57039/tcp                  ALLOW       10.2.0.3                   (log) # CESNET3 RP3
57040/tcp                  ALLOW       10.2.0.4                   (log) # CESNET3 RP4
57041/tcp                  ALLOW       10.2.0.5                   (log) # CESNET3 RP5
57042/tcp                  ALLOW       10.3.2.9                   (log) # CESNET3 RR1
57043/tcp                  ALLOW       10.3.1.6                   (log) # CESNET3 RR2
57044/tcp                  ALLOW       10.3.2.13                  (log) # CESNET3 RR3
57045/tcp                  ALLOW       10.3.1.7                   (log) # CESNET3 RR4
57079/tcp                  ALLOW       10.2.27.1                  (log) # CESNET3 R27
57080/tcp                  ALLOW       10.2.29.1                  (log) # CESNET3 R29
```

Figure 5.7: Firewall on MDTC1

5.1.4 Logging

The testing if logging is functional was done through SSH access to the logging server. All the logs from the system are aggregated into a single folder. The folder can be seen in figure 5.8.

```
loub@vinovago:~$ ls -la logs/telemetry
lrwxrwxrwx 1 root root 30 Jan 28  2020 logs/telemetry -> /var/log/cls-servers/telemetry
loub@vinovago:~$
```

Figure 5.8: Log folder on CESNET syslog server

5.1.5 Resiliency to failure

This point can be misinterpreted. To ensure complete resiliency, every part would need to be duplicated. This resiliency is basically provided by system location. Since all parts of the system are located in the CESNET virtualization platform, which has two geographically detached nodes, this

allows resiliency to hardware failures. In case of failure, virtual machines are migrated to other locations.

The current design with the development branch can be easily switched to production so that even software problems wouldn't result in system collapse.

5.2 Statistics about collected data

There are many ways to check that data are being collected from devices and written into the database. One of the ways was shown in 5.2 command. We can list statistics about *Total bytes sent* and *Total packets sent* using correct show commands for devices running on IOS-XR OS. The command is helpful for checking if the device sends data to the collector, but listing the outputs of this command in this thesis would take up too much space. Also, on the Cisco IOS-XE platform is no internal statistics of sent data, so no similar show command can be executed.

Another way to check that collector receives data from devices is using command *tcpdump* on the collector. As shown in figures 5.9 for collector MDTC1 and in 5.10 for MDTC2. The *tcpdump* command was collecting packets for one minute. As we can see, both collectors are receiving data from devices. The way the command was executed doesn't prove itself that all devices are sending data to collectors. This could be done by limiting the filter of *tcpdump* from collecting packets on the whole subnet to single devices.

```
root@mdtc1:[ ~ ] $ tcpdump -w dump.out net 10.2.0.0/16 & sleep 60; kill $!
[1] 342483
tcpdump: listening on ens160, link-type EN10MB (Ethernet), snapshot length 262144 bytes
4544 packets captured
4558 packets received by filter
0 packets dropped by kernel
```

Figure 5.9: *tcpdump* on MDTC1

```
root@mdtc2:[ ~ ] $ tcpdump -w dump.out net 10.3.0.0/16 & sleep 60; kill $!
[1] 8951
tcpdump: listening on ens192, link-type EN10MB (Ethernet), snapshot length 262144 bytes
3477 packets captured
3617 packets received by filter
0 packets dropped by kernel
```

Figure 5.10: *tcpdump* on MDTC2

Another way would be to analyze *dump.out* files in Wireshark. These files are added as attachments to this thesis. On both can be seen that during that one minute, all IP addresses that were supposed to send data did.

This command was also executed on database 5.11, and we can see that the collectors are sending data to the database. This *tcpdump* output file is included in the attachments.

The previous checks are helpful for debugging. We can suffice with a simple check. If the device is sending data to the collector, it then transfers data to

the database, where they are saved. The device will appear in the Grafana visualization as a possible source. This list can be compared with the list of devices in Netbox.

```
root@hydra:[ ~ ] $ tcpdump -w dump.out host 185.8.160.228 or host 78.128.211.217 & sleep 60; kill $!
[1] 361821
tcpdump: listening on ens160, link-type EN10MB (Ethernet), snapshot length 262144 bytes
4550 packets captured
4550 packets received by filter
0 packets dropped by kernel
```

Figure 5.11: *tcpdump* on HYDRA

5.3 Visualization testing

The visualization testing consists of two systems. First is the Nagios software that has been used primarily for alerting of network outages. The main reasons for using Nagios were two. Firstly the operators on the CESNET service desk already work with it and are used to it. Secondly, the alert handling in Grafana is not that well implemented and would need much more work.

The Nagios system is handled by other admins, so I didn't post any configurations in this thesis. Nagios initiates the communications between this monitoring system and Nagios. It sends queries to InfluxDB and displays received data. The alerts for one of the routers are in figure 5.12.

The Grafana tool is used for visualizing statistics about network devices and their interfaces. The basic dashboards have already been described in section 4.5. These dashboards have been modeled based on the G3 system developed by CESNET. Testing regarding visualization is quite limited. Since Grafana only does queries into InfluxDB, then all devices already in the database are also visible in Grafana.

Currently, Grafana dashboards are used and being tested by CESNET NOC team. Unfortunately, I don't have any feedback yet. I expect that requests for changes in dashboards will come up throughout time.

Host	Service	Status	Last Check	Duration
R1-Praha1	EVPN ethernet-segment	OK	20:56:36	37d 3h 10m 29s
	HW Health	OK	20:57:51	0d 2h 24m 55s
	IFSTATUS CESNET3	OK	20:58:12	12d 7h 8m 21s
	NTP time	OK	21:00:13	26d 10h 6m 30s
	PING	OK	20:56:35	37d 3h 10m 35s
	l2vpn	OK	20:56:36	12d 6h 35m 11s

Figure 5.12: Router R1 in Nagios

Chapter 6

Conclusion

This work is a direct continuation of my bachelor thesis, "Network traffic monitoring using Model-Driven Telemetry" [16]. I have focused in it on creating a monitoring stack for testing of new monitoring technology Model-Driven Telemetry. Based on the results from testing in the thesis, further research and possible deployment of a monitoring system were considered.

This thesis's focus was on deploying a monitoring system that would collect monitoring data from devices using both Model-Driven Telemetry and SNMP technologies. In that function, the deployed monitoring system is more than capable, thanks to the use of Telegraf software for the collector part of the system. The plugins of Telegraf provide many more monitoring options for the system in the future for possible extensions.

The system fulfills all the requirements which were set before its deployment. It collects data using Model-Driven Telemetry from devices in the CESNET3 network. The plan is to incorporate all routers and switches into the system in the future as well as all devices from the DWDM layer of the network. It's based entirely on open-source software. The collectors are deployed as Telegraf containers in Docker. The time-series database InfluxDB is used for storing data. Visualization of alerts and statistics is handled by Nagios and Grafana tools. Simple management by the Ansible playbook is used. The system is capable of storing data for a long time with consideration of the performance of the database. The critical parts of the system have backups in case of failure. The development branch of the system provides the continuous development of the system on its testing servers without affecting production servers.

The system is currently used by CESNET NOC [20]. As of writing this thesis, I didn't receive any request for a change of Grafana dashboards. Grafana is one tool that will need further development as the created dashboards need more work done. In the current state, they are still relatively simple.

During the last month of development, new agenda from management emerged. The character of the system design is competent and can be extended for more uses than simple monitoring. In the future, the system should calculate and display SLA [21] for CESNET members and clients. This

[20] Network operation center (NOC)

[21] Service level agreement (SLA)

feature is approved, but thanks to a late decision, it will be included in future system releases.

Appendix A

Bibliography

[1] URL: https://www.cesnet.cz/sluzby/pripojeni/topologie/ (visited on 05/02/2023).

[2] Vaishnavi Abirami. *Monitor GRPC calls with OpenTelemetry - explained with a Golang example*. Feb. 2023. URL: https://signoz.io/blog/opentelemetry-grpc-golang/ (visited on 02/23/2023).

[3] M. Bjorklund. *YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. RFC 6020. http://www.rfc-editor.org/rfc/rfc6020.txt. RFC Editor, Oct. 2010. URL: http://www.rfc-editor.org/rfc/rfc6020.txt (visited on 02/21/2023).

[4] Eduardo Canale, Claudio Risso, and Franco Robledo. *Optimal design of an IP/MPLS over DWDM Network*. Apr. 2014. URL: https://www.scielo.br/j/pope/a/Hdt4jnDDQ5npgjnVC9f8pdM/ (visited on 05/02/2023).

[5] Benoit Claise, Joe Clarke, and Jan Lindblad. *Network programmability with Yang: The structure of network automation with Yang, Netconf, restconf, and gnmi*. Addison-Wesley, 2019.

[6] Ribbon Communications. *What Is IP MPLS?* URL: https://ribboncommunications.com/company/get-help/glossary/ip-mpls (visited on 05/02/2023).

[7] *Docker*. May 2023. URL: https://cs.wikipedia.org/wiki/Docker (visited on 02/23/2023).

[8] Rob Enns. *NETCONF Configuration Protocol*. RFC 4741. RFC Editor, Dec. 2006. URL: https://www.rfc-editor.org/info/rfc4741 (visited on 02/21/2023).

[9] Chinmay Gaikwad. *Making the most out of OpenTelemetry*. Feb. 2023. URL: https://techblog.cisco.com/blog/most-out-of-opentelemetry (visited on 02/23/2023).

[10] Craig Hill. *Evolving Network Automation Techniques for Real-Time Applications [PowerPoint slides]*. URL: https://tenica-gs.com/wp-content/uploads/2019/05/1-11-Evolving-Network-Automation-Techniques-for-Real-Time-Applications-Craig-Hill-Cisco.pptx (visited on 02/23/2023).

[11] *How Ansible works. Red Hat Ansible*. URL: https://www.ansible.com/overview/how-ansible-works (visited on 02/23/2023).

[12] *HTTPS*. May 2023. URL: https://en.wikipedia.org/wiki/HTTPS (visited on 02/23/2023).

[13] Hitesh Jethva. *Ansible vs Puppet – What is the Difference? (Pros and Cons). Cloud Infrastructure Services*. URL: https://cloudinfrastructureservices.co.uk/ansible-vs-puppet/ (visited on 02/23/2023).

[14] Hitesh Jethva. *Ansible vs Terraform – What is the Diference? (Pros and Cons)*. URL: https://cloudinfrastructureservices.co.uk/ansible-vs-terraform/ (visited on 02/23/2023).

[15] Hitesh Jethva. *Terraform vs Puppet – What's the Difference? (Pros and Cons)*. URL: https://cloudinfrastructureservices.co.uk/terraform-vs-puppet-whats-the-difference/ (visited on 02/23/2023).

[16] Ladislav Loub. *Monitorování provozu síte pomocí Model-Driven Telemetry*. Czech. Bachelor thesis. ČVUT, Jan. 2021.

[17] *Netconf*. Dec. 2021. URL: https://en.wikipedia.org/wiki/NETCONF (visited on 02/23/2023).

[18] *NetFlow*. May 2023. URL: https://en.wikipedia.org/wiki/NetFlow (visited on 02/23/2023).

[19] *Optical Fiber Communication: The science behind it*. URL: https://yourwebsitedemo.in/hfcl-groups/blog/optical-fiber-communication.html (visited on 05/02/2023).

[20] Peter. *Real-time flow telemetry for routers*. Apr. 2022. URL: https://blog.sflow.com/2022/04/real-time-flow-telemetry-for-routers.html (visited on 02/23/2023).

[21] *Puppet (software)*. Apr. 2023. URL: https://en.wikipedia.org/wiki/Puppet_(software) (visited on 02/23/2023).

[22] *Simple Network Management Protocol*. May 2023. URL: https://en.wikipedia.org/wiki/Simple_Network_Management_Protocol (visited on 02/23/2023).

[23] Haoyu Song, Zhenqiang Li, Pedro Martinez-Julia, Laurent Ciavaglia, and Aijun Wang. *Network Telemetry Framework*. RFC 4741. RFC Editor, Dec. 2006. URL: https://datatracker.ietf.org/doc/draft-song-opsawg-ntf/03/ (visited on 02/21/2023).

[24] Haoyu Song, Fengwei Qin, Pedro Martinez-Julia, Laurent Ciavaglia, and Aijun Wang. *Network Telemetry Framework*. RFC 9232. RFC Editor, May 2022. URL: https://www.rfc-editor.org/info/rfc9232 (visited on 02/21/2023).

[25] *Spotlight - network telemetry system*. URL: https://techex.co.uk/monitoring/spotlight-network-telemetry-system (visited on 02/23/2023).

[26] *The Premiere Network Source of Truth. Netbox*. URL: https://docs.netbox.dev/en/stable/ (visited on 02/21/2023).

[27] *What is flow telemetry?* URL: https://www.netscout.com/what-is/flow-telemetry (visited on 02/23/2023).

[28] *What is int (in-band network telemetry)*. July 2022. URL: 'https://cloudswit.ch/blogs/what-is-int%5C%EF%5C%BC%5C%88in-band-network-telemetry%5C%EF%5C%BC%5C%89/' (visited on 02/23/2023).

[29] *What is terraform: Terraform: HashiCorp developer*. URL: https://developer.hashicorp.com/terraform/intro (visited on 02/23/2023).

[30] *XML*. May 2023. URL: https://en.wikipedia.org/wiki/XML (visited on 02/23/2023).

Appendix B

Glossary

API Application Programming Interface. 12, 13, 19, 35

ASR Aggregation Services Routers. 4, 30

CESNET Czech Education and Scientific NETwork. vi, 1, 2, 4–6, 9–11, 18, 22, 24–27, 32, 33, 35, 38–40, 44, 53, 55, 56, 58, 59

CLI Command-line interface. 3, 10, 11

CPU Central processing unit. 40

CRS Carrier Routing System. 2, 5

DWDM Dense Wavelength-division Multiplexing. vi, 4–6, 23

EPNM Evolved Programmable Network Manager. 23

gNMI gRPC Network Management Interface. 13, 17, 50

gRPC Google Remote Procedure Calls. 13, 19

GUI Graphical user interface. 3, 10, 19

HTTP Hypertext Transfer Protocol. 12, 13, 19, 39

HTTPS Hypertext Transfer Protocol Secure. 12, 39

IETF Internet Engineering Task Force. 11, 13

IOS Internetworking Operating System. 4, 5, 30

IP Internet Protocol. vi, 4–6, 22, 55

IPv4 Internet Protocol version 4. 40

IPv6 Internet Protocol version 6. 40

ISO/OSI International Organization for Standardization/Open Systems Interconnection model. 5

JSON JavaScript Object Notation. 12, 13

LXC Linux Containers. 16

MDT Model-Driven Telemetry. iv, vi, 1–3, 5, 7, 9, 11, 13, 16–19, 21–23, 26–29, 31–34, 49–51, 59

MPLS Multi-Protocol Label Switching. vi, 4–6

MTU Maximum Transmission Unit. 28

NCS Network Convergence System. 4, 5, 30

NETCONF Network Configuration Protocol. vi, 7, 11–13, 26, 27

NOC Network operations center. 58, 59

OIDC OpenID Connect. 39

OOB Out-of-Band. 4

RAM Random-access memory. 2, 40

RESTCONF Representational State Transfer Configuration Protocol. 12, 13

RFC Request for Comments. iv, 1, 5, 11, 13

RPC Remote procedure call. 11, 13

SLA Service level agreement. 59

SNMP Simple Network Management Protocol. 1–3, 5, 7, 18, 19, 21, 22, 33, 49, 51, 52, 59

SSH Secure Shell. 3, 7, 12, 19, 22, 32, 55, 56

SSL Secure Sockets Layer. 40

TCP Transmission Control Protocol. 7, 10, 13, 33, 39, 55

TCS Trusted Certificate Service. 39

TLS Transport Layer Security. 38

UFW Uncomplicated Firewall. 32, 33

VRF Virtual Routing and Forwarding. 30

WiFi Wireless Fidelity. 5

XML Extensible markup language. 7, 11–13, 27

YANG Yet another next generation. 4, 7, 11, 13, 18, 19, 27, 28, 31, 33, 41, 43

Appendix C

Attachments

- `src` ..... directory with source files
  - `captured_traffic` ..... captured packets
  - `config` ..... configuration files
  - `thesis` ..... thesis's source files in LaTeX
- `text` ..... thesis text
  - `thesis.pdf` ..... thesis text in PDF format

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Loub Ladislav** Personal ID number: **466847**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Telecommunications Engineering**

Study program: **Electronics and Communications**

Specialisation: **Communications Networks and Internet**

II. Master’s thesis details

Master’s thesis title in English:

Monitoring System Based on Telemetry in Network CESNET3

Master’s thesis title in Czech:

Monitorovací systém založený na telemetrii v síti CESNET3

Guidelines:

Following your bachelor thesis, describe and evaluate the new developments in telemetry and monitoring data collection.
Explore existing monitoring tools and evaluate the possibilities of extending these tools with telemetry.
Build a production monitoring system for data collection using Mode-Driven Telemetry and SNMP. Test the monitoring system.

Bibliography / sources:

[1] Loub L, Monitorování provozu sítě pomocí Model-Driven Telemetry, Bakalářská práce, ČVUT v Praze, 2021
[2] Song, H., Qin, F., Martinez-Julia, P., Ciavaglia, L., and A. Wang, 'Network Telemetry Framework', RFC 9232, DOI 10.17487/RFC9232, May 2022, <https://www.rfc-editor.org/info/rfc9232>.

Name and workplace of master’s thesis supervisor:

Ing. Jan Kubr, Ph.D. Department of Computer Graphics and Interaction FEE

Name and workplace of second master’s thesis supervisor or consultant:

Date of master’s thesis assignment: **13.01.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Jan Kubr, Ph.D. Supervisor’s signature	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

. | Date of assignment receipt | Student’s signature