TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Magnus Jakob Epkin 205858IAAB

Automating the Network Device Configuration Process for the Health and Welfare Information Systems Centre

Bachelor's thesis

Supervisor: Mohammad Tariq Meeran PhD

Tallinn 2025

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Magnus Jakob Epkin 205858IAAB

Võrguseadmete konfigureerimise protsessi automatiseerimine Tervise ja Heaolu Infosüsteemide Keskuses

Bakalaureusetöö

Juhendaja: Mohammad Tariq Meeran PhD

Tallinn 2025

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Magnus Jakob Epkin

06.01.2025

Abstract

This thesis aims to create an automated process to configure network switches for branch offices serviced by the Health and Welfare Information Systems Center. This is achieved by creating a process that utilizes the Zero Touch Provisioning capabilities that exist on the switches.

The author provides a comparative overview of manual and automated network configuration methods along with their benefits and drawbacks. Then the author provides an overview of the methodology used for selecting the tools to be used in an experimental setup to test the Zero Touch Provisioning process.

The thesis is concluded by the authors analysis of results from the experimental setup, bringing out future possibilities to expand on the work that was done.

This thesis is written in English and is 44 pages long, including 6 chapters and 5 figures.

Annotatsioon

Võrguseadmete konfigureerimise protsessi automatiseerimine Tervise ja Heaolu Infosüsteemide Keskuses

Antud lõputöö eesmärk on luua automaatne protsess, mille käigus on võimalik automaatselt seadistada võrgulüliteid kasutuseks Tervise ja Heaolu Infosüsteemide Keskuse poolt teenindatavates asutustes.

Töö käigus võrdleb autor manuaalset ja automaatset võrguseadistust, tuues välja mõlema meetodi eeliseid ja puudujääke. Seejärel toob autor esile metoodika mille põhjal valida töö eesmärgi täitmiseks kasutatavad töövahendid.

Peale töövahendite valimist seadistab autor prototüübi, mille eesmärk on automaatne võrgulülitite seadistamine eelnevalt teadaolevate parameetrite järgi. Prototüübi töö efektiivsuse mõõtmiseks mõõdab autor ajakulu vahet manuaalselt ja automaatselt seadistamise puhul.

Töö lõpus analüüsib autor prototüübi valideerimise käigus saadud tulemusi ning toob esile võimalusi juba tehtud töö edasisteks arendamiseks ja parandamiseks.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 44 leheküljel, 6 peatükki, 5 joonist.

List of abbreviations and terms

API	Application Programming Interface
DHCP	Dynamic Host Configuration Protocol
IP	Internet Protocol
IPSec	Internet Protocol Security
IPv4	Internet Protocol Version 4
IT	Information technology
JunOS	JunOS
Network Switch	A network device that connects devices on a computer network by utilizing packet switching to receive and forward data
OS	Operating System
OTP	One Touch Provisioning
SDN	Software Defined Networking
SFP	Small Form-Factor Pluggable
SNMP	Simple Network Management Protocol
TEHIK	Health and Welfare Information Systems Centre (Tervise ja Heaolu Infosüsteemide Keskus)
VLAN	Virtual Local Area Network
VM	Virtual Machine
YAML	YAML Ain't Markup Language
ZTP	Zero Touch Provisioning

Table of contents

List of figures

1 Introduction

In many organizations, manual configuration of network equipment remains the standard practice. While this approach generally works well at smaller scales, it can become problematic in larger environments, where speed and consistency are important. But at any scale, manual configuration is more prone to errors and inconsistency, often due to differing approaches among network administrators.
Automation has become increasingly instrumental in managing IT infrastructure, enabling streamlining various tasks and reducing human error across deployments. One of the processes used in infrastructure automation is ZTP (Zero-Touch Provisioning). ZTP allows for new devices to be configured with minimal manual intervention.
This thesis aims to automate much of the processes used for configuring network equipment by using the ZTP process to configure network equipment that are to be deployed in branch offices serviced by TEHIK (Health and Welfare Information Systems Centre).

1.1 Problem Statement

The current practice of manually configuring network equipment at TEHIK can be prone to errors in configuration tasks that take extra time to troubleshoot if not caught before the devices are sent out for deployment. This manual approach can also cause problems with administrators using different approaches when configuring devices, leading to more inconsistencies as time goes on.

1.2 Research goals and scope

The goal of the thesis is to design and implement a solution to configure network equipment with minimal manual intervention, thus reducing chances for human error, increasing consistency across device configurations, and saving time for network administrators, who can then focus more on other tasks.

The scope of the thesis covers automatic configuration of Juniper Networks EX2300 series network switches, while branch office environments also include configuring firewalls, those fall out of scope due to their increased complexity.

1.3 Research questions

The following research questions have been formulated to guide this research.

- What are the challenges of manual configuration and how can automation help in addressing the challenges?
- What are the limitations of automating configuration tasks?
- What type of skills and knowledge are required by administrators in a ZTP environment?

2 Background

This chapter presents an overview of the differences between manual and automated configuration, their benefits and drawbacks. Additionally, it examines the role of automation in network configuration and reviews comparative studies on its effectiveness.

2.1 Overview of network configuration methods

This chapter gives an overview of the advantages and drawbacks of manual and automated configuration.

While still widely used in smaller environments, the classic way of configuring network devices using a physical console port on the device can become troublesome in larger environments, where separately configuring devices will end up taking too much time to be sustainable. [1], [2]

Manual configuration is still widely used for testing and troubleshooting devices, however for production deployments it can be time-consuming, more prone to errors in configuration, and more open to different approaches from administrators, which in turn can lead to problems when an organization is transitioning to a more automated approach to network management. [2]

Automated configuration aims to provide a solution to problems found with manual configuration by providing a way to effectively standardize configurations across deployments, reduce errors, and save time needed by administrators to do their work. However, this comes with the need to train administrators on new tools and assumes at least basic knowledge of programming languages. [3]

2.2 Automation in network management

This chapter covers the most common capabilities used to enable automation in network management with a closer look at SDN (Software Defined Networking), Configuration Management Tools, and ZTP along with OTP (One Touch Provisioning)

2.2.1 Software Defined Networking

SDN allows for network management from a centralized application. With SDN, administrators can consistently manage the network devices. While network architecture consists of three planes: Data, Management, and Control planes. In traditional network design all the planes are on the device itself. SDN works by separating the networks' control and data planes to create software-programmable infrastructure. This allows for network administrators to use APIs (Application Programming Interfaces) to programmatically provision and manage remotely connected devices. Figure 1 shows how SDN architecture differs from a traditional architecture. [4], [5]

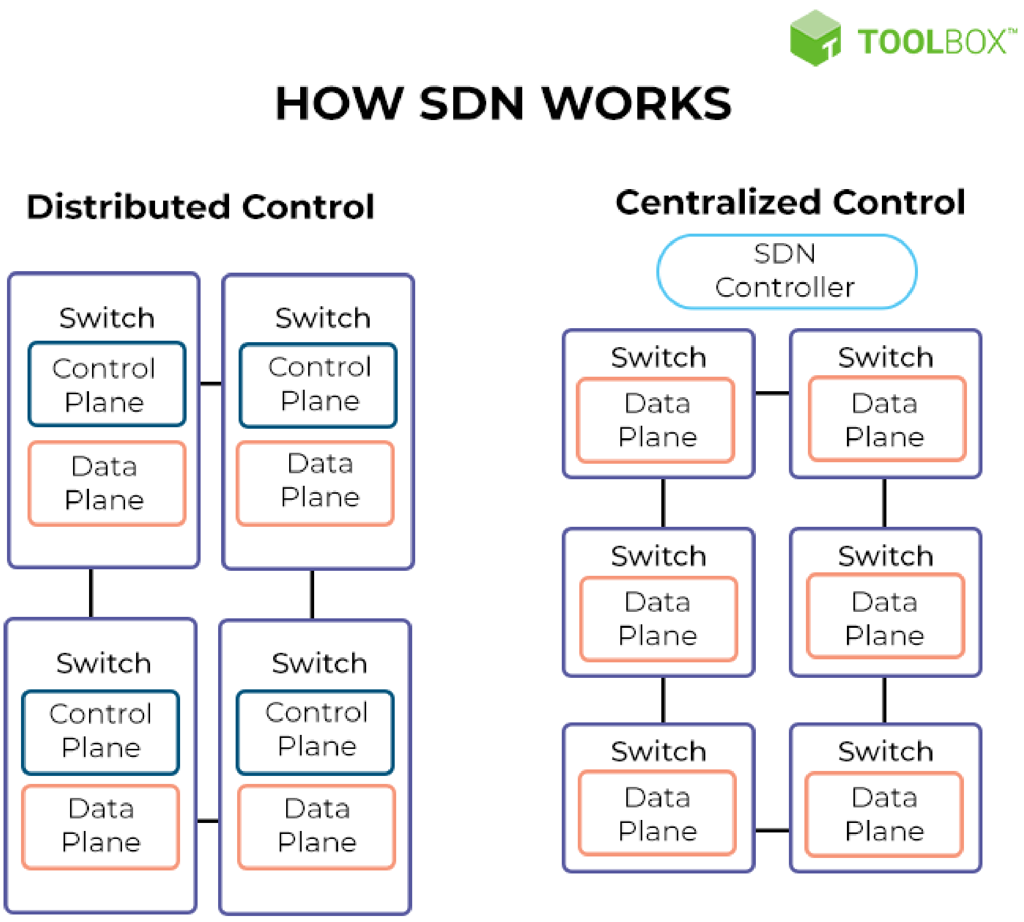

Figure 1. SDN architecture [6]

2.2.2 Configuration management tools

This section covers some of the most popular tools used for configuration management in networks with a focus on open-source tools.

Ansible is an open-source suite of command-line tools written in Python and developed by Red Hat Inc. Some of the main factors supporting the adoption of Ansible are its agentless architecture running in a “push” model, meaning that no extra software is required on managed machines for Ansible to do its work. Ansible performs its operations using Playbooks written in YAML that contain a series of *Plays* to complete, each play being one to multiple tasks to complete on any number of hosts that are defined in a separate file. Each task is a call to an Ansible module that is made for a specific task.

Playbooks can further be split into reusable groups called *roles* that can be reused wherever needed. Tasks are all executed in order on all hosts. [7], [8]

Another tool used for configuration management is Puppet, while Puppet is originally an agent-based tool, meaning that an agent is required to be installed on a managed host. However, an agentless addition to Puppet, called Puppet Bolt exists, to improve compatibility with network devices. Just like Ansible, Puppet needs two files to be configured for operation, a target file, which contains hosts that are changed, and a file that contains configuration changes to be applied. Puppet change files can be written in YAML or a custom Puppet language, which are then executed one after the other on a managed host. [8]

The final automation tool this thesis will investigate is Chef. Contrary to Ansible and Puppet Bolt, Chef uses an agent-based architecture, where a Chef agent must be installed on a managed host. Chef uses *Cookbooks* and *Recipes* to apply changes to targeted hosts. Chef is written in Ruby and Erlang [9].

2.2.3 Zero and One Touch Provisioning

ZTP is a process that is used to provision devices with minimal manual intervention from administrators. ZTP is often used in larger enterprise environments where it is not reasonable to manually configure a new device whenever one is needed.

ZTP works by using a DHCP (Dynamic Host Configuration Protocol) request to acquire information on where to go for its configuration and software. The response from a DHCP server to a device being provisioned contains the location of the boot server, the locations of the configuration file and if necessary, an OS (Operating System) image to update to. Extra parameters can also be sent, such as DNS (Domain Name System) server address, the transfer protocol to be used, an address for a NTP (Network Time Protocol) server, and a Syslog server address. [10]

OTP utilizes a similar workflow to ZTP but differs in that OTP requires a point of contact from an administrator outside a device simply being connected to a network. "OTP is often used in situations where ZTP would need additional configuration, such as for VLAN (Virtual Local Area Network) or static IPv4 addresses configuration … It is important to note that not all ZTP implementations are truly zero touch, and some devices may require minimal touch or one touch provisioning." [11].

2.3 Comparative studies

Studies show that implementing a more automated approach can provide a reduction in time to deploy hardware while reducing human-error in configuration and providing more consistency across devices in different environments. [1], [2], [3], [12]

When it comes to configuration management, it is generally agreed, that Ansible, due to the use of Python, has a simpler learning curve when compared to other tools with a similar purpose. Furthermore, an agentless design means that devices do not need to install any extra software to be configured further. [7], [8], [9]

ZTP and OTP enable higher efficiency when there is a need to provision devices in bulk, while reducing need for administrator input and helping in error reduction due to the possibility of creating a validated base from which future configurations are made from. [10], [11]

3 Method

This chapter covers the methodology used to design, implement, and test the tools used for the automatic provisioning of the Juniper Networks EX2300 network switches. An experimental approach was chosen as it aligns with the goal of the thesis to design and implement a solution to configure network devices with minimal manual intervention.

3.1 Review process

This section covers the review process and criteria used for selecting the tools used for the experimental setup. For selecting the tools to be used, the following requirements were observed:

- Compatibility with Juniper networks devices. As the goal of the project revolves around Juniper EX2300 series of network switches, it was vital that any tools used were also supported by the switches.

- Future expandability. Should the scale of the deployment increase, it is vital to have the ability to do so with minimal changes to the experimental setup that was created. For this, the DHCP server capability of the Juniper devices was selected as it enables to quickly add capacity for DHCP clients should the need arise.

- Ease of troubleshooting. In case of any network problems, the paths used by the ZTP process should be easily traced to simplify any troubleshooting.

- Ease of management. The setup should be easily understood and managed. Due to this, Ansible, with its YAML-based playbooks and Jinja2 templates was selected, as it simplifies adding new capabilities to the device configuration when needed.

3.2 Experimental setup

A Linux-based VM (Virtual Machine) was deployed with a Nginx web server to host any necessary OS and configuration files.

The experimental setup consists of zeroized Juniper EX 2300 series network switches to act as the devices to be provisioned. The switches are running a version of JunOS (Juniper OS) that supports ZTP. The switches come with 12, 24, or 48 ports that operate at up to 1 gigabit speeds and between two and four SFP (Small Form-Factor Pluggable) ports that can operate at up to 10 gigabit speeds, all switches also include a management port [13] [14]. The switches support the ZTP process on any bound interface other than the console port.

A Juniper SRX300 firewall was selected as the DHCP server, as it has the capabilities needed, furthermore, this reduced the time needed for troubleshooting network problems, would any occur. The SRX300 was connected to the VM hosting the configuration files via an IPSec tunnel, this was chosen to reduce troubleshooting time in case of any network problems. Figure 2 shows the network topology used for testing. The firewall has 6 ports that operate at 1 gigabit speeds and 2 SFP ports that can operate at up to 10 gigabit speeds [15].

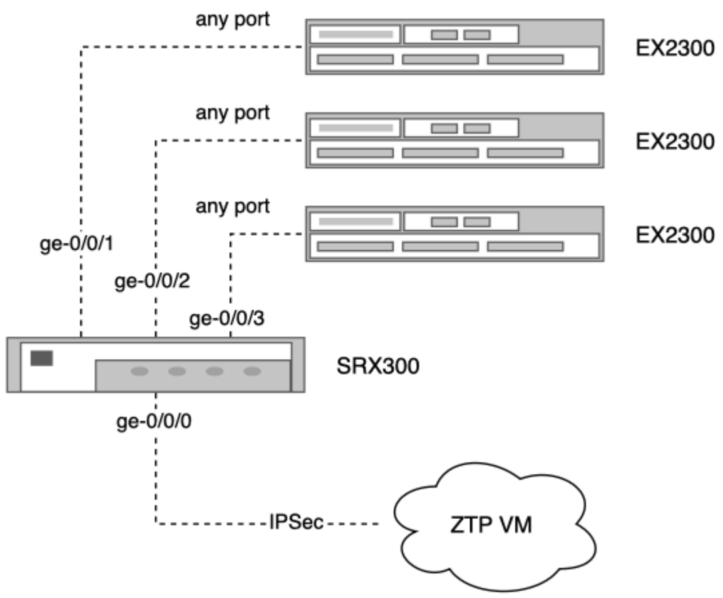

Figure 2. Topology of the experimental setup

The DHCP server configuration passes the following parameters to a connected host via set options: an IP address, a router address, the IP of the configuration server and location of the configuration file on it, the location of an OS file on the configuration server, and the address of an NTP server. Providing an NTP server address is vital due to JunOS packages being signed by Juniper, if a device being provisioned has a time that is set to before the signature date on the OS image, the upgrade will fail due to an invalid signature. Furthermore, each port configuration was specifically for an EX2300 switch, as the DHCP options could provide one OS image filename. If a device that does not support the provided image is connected, the ZTP process will be stuck until the problem is noticed and corrected.

Ansible was selected as the tool to create the device configurations. This was done as Juniper devices had active support for automation with Ansible and Ansible was already used to manage configurations at TEHIK, additionally the author had the most familiarity with Ansible at the time. As Ansible works based off YAML files containing the variables needed to build the configuration, the variables provided would be used by Jinja2 template files to assemble the device configuration, which would then be uploaded to the VM that were provided to the zeroized devices during the ZTP process.

3.3 Data collection and analysis

For a better overview of efficiency improvements, time taken to configure 1 and multiple switches following the same setup criteria were measured. When measuring time taken for multiple switches, setup was performed under the assumption, that the switches would be deployed at the same site, meaning that less changes would have to be applied to the variable files used for building the switch configurations.

4 Experimental design

This chapter covers the tools and procedures used to test the ZTP process for provisioning network switches.

4.1 Equipment and tools

Juniper EX2300 series network switches were selected to be used in the experimental setup as they are most common on a branch office environment.

A Juniper Networks SRX300 firewall was selected to function as a DHCP server to provide an IP address and ZTP parameters to use for the network switches. The SRX connects to the virtual machine used to host the configuration files via an IPSec tunnel. On the SRX, 1 port was reserved as an uplink, while 3 ports were configured to provide the DHCP information to any connected devices. Figure 3 shows the DHCP lease configuration used on the SRX300.

```
access {
    address-assignment {
        pool ZTP-DHCP-SW-1 {
            family inet {
                network 10.20.200.0/29;
                range ZTP-DHCP-SW-1-RANGE {
                    low 10.20.200.3;
                    high 10.20.200.6;
                }
                dhcp-attributes {
                    maximum-lease-time 3600;
                    router {
                        10.20.200.2;
                    }
                    boot-file https://*ZTP VM
IP*/config/sw1/sw.config;
                    boot-server *ZTP VM IP*;
                    option 42 ip-address *NTP server IP*;
                    option 43 hex-string
001F6F732F6A756E6F732D61726D2D33322D32312E3452332D53372E362E74677A0114
636F6E6669672F7377312F73772E636F6E66696703056874747073;
                }
            }
        }
    }
}
```

Figure 3. SRX300 DHCP address assignment pool configuration

An Ansible playbook and Jinja2 template were created to build configuration files and upload them to the VM that the DHCP configuration pointed towards. The playbook included basic checking to help ensure an invalid configuration file was not uploaded to the ZTP VM. This checking made sure that any VLANs used by interfaces and were declared and in cases where any separate interfaces were added to ranges, the ranges were checked to also exist.

A Jinja2 template was built using a variety of “for loops” for any part of the configuration that might have multiple instances such as interface ranges and VLANs. For sections of the configuration that might not always be needed were added in when statements for them existed in the variable files (see appendix 3).

A second Jinja2 template was made that contained any common configuration information such as NTP server addresses, local user accounts, and authentication servers. A second template was made to ease adding new device types to provisioning, such as firewalls, as the name implies, the common portion generally is common across devices (see appendix 4).

Variable files were created following a common naming scheme, where the name of the file matched the to-be IP of the device to be provisioned. The variables included the future management IP address of the device, the switch hostname, the SNMP (Simple Network Management Protocol) location, definitions of port ranges and their modes, the ability to add separate interfaces to a range, configurations ability for unique interfaces such as uplinks, a layer three address for in-band management, and VLAN definitions (see appendix 5).

4.2 Experimental procedure

The switches used for testing were selected randomly from cold storage and checked to be zeroized and ensured they were running a version of JunOS that supports ZTP. For the ZTP process, parameters were selected that would mimic the needs of a branch office environment, this included access ports for end-users, Wi-Fi access points, and printers. In the ZTP workflow, the requirements for device configurations were made to mimic a common branch office environment. Figure 4 shows the workflow to be used with the ZTP process.

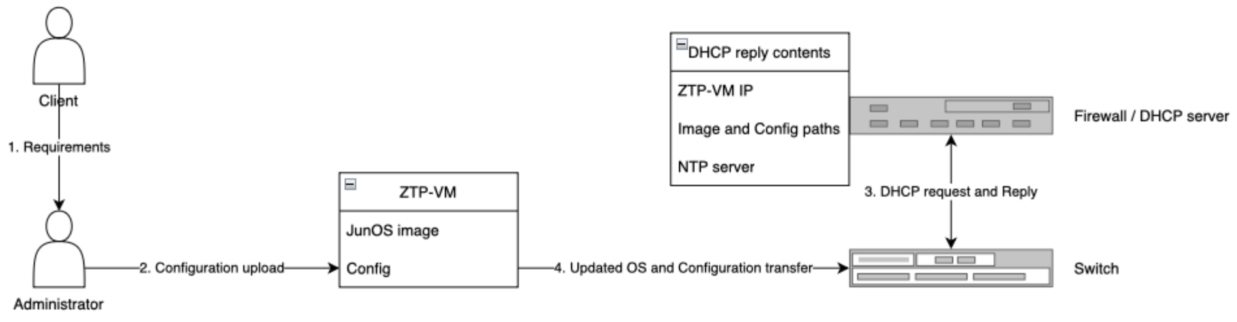

Figure 4. ZTP workflow

For the switch configuration experiments, switches taken from storage were powered on and checked via console access to be zeroized and running JunOS release 12.2 or later as only releases after 12.2 support ZTP [10].

The ZTP process is automatically started on a switch that is running on a factory-default configuration once it detects a bound interface and receives at least the minimal data needed for ZTP from a DHCP server [10].

Precise time measurements were deemed unnecessary, and time measurements were done by observing the time at the start and end of any measured actions, such as creating templates or manually configuring a switch.

Once the ZTP process on the device was finished, the configuration was checked to be in accordance with the template that was provisioned earlier.

4.3 Evaluation metrics

The metrics used for evaluation of the ZTP solution consisted of time needed to set up the DHCP server, time taken from filling the provisioning template to system readiness, and comparisons to manual configuration following similar criteria. Furthermore, evaluation considered the time where an administrator was actively engaged with the configuration process.

5 Results and discussion

This chapter covers the results of the experimental setup and provides an analysis, bringing out practical implications, the author also aims to answer the research questions that were formulated in the beginning of the thesis.

5.1 Results

For setting up the DHCP service on the SRX300 firewall, being a one-time operation, took roughly 1 hour, mainly due to the OS file path DHCP option needing to be written in hexadecimal form, which was unfamiliar to the author at the time.

The time taken for manual configuration of one switch was measured to be consistently around 10 minutes, however, the ZTP process took around five to six minutes to prepare and upload the configuration file depending on the number of parameters provided.

In cases where multiple switches were required, changing only a couple of necessary variables meant that each extra switch that had to be provisioned added roughly 2 minutes to the time where an administrator had to actively provide inputs to the process. For gathering data about multiple switches, three switches were used to simulate a larger branch office deployment.

After a brief introduction to formatting and the workflow of the ZTP process, an experienced network administrator needed more time to make ready a switch compared to the author, who created the components used, mainly due to the unfamiliar nature of the process, however, this can be overcome as familiarity and confidence in the process grows.

5.2 Analysis

From the time measurements taken by the author, it was seen that compared to manual configuration, a templated approach achieved a 40% - 50% improvement in time taken for an administrator to ensure a switch would be ready. The improvement was more increasingly more evident when multiple switches were provisioned. When readying 3 switches manually was measured to take around 25 minutes, the ZTP process took around

10 minutes of attention from administrators. These results were recorded without the need to perform an OS update. Figure 5 shows the comparison of time taken to ready new switches.

When an OS update was also needed, the manual approach took longer to perform since the switch would also have to be manually updated, adding around five minutes to the time taken. Due to how the ZTP process is set up, an OS image is always downloaded from the VM hosting the configuration files and checked against the currently running version. While this approach means that in cases where an OS update was not necessary, the process took longer to finish compared to a manual approach, however, it was deemed to be acceptable as it would be unreasonable to keep all stored devices constantly updated.

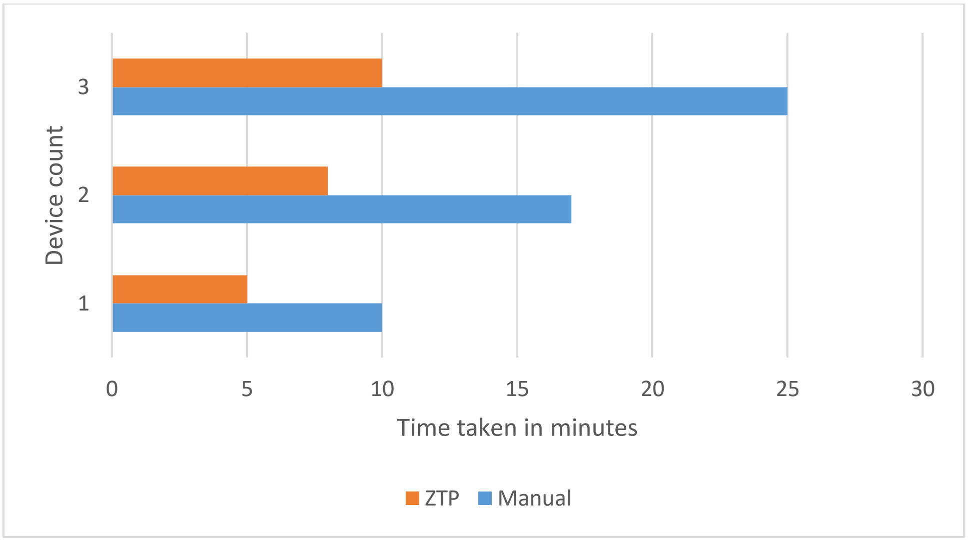

Figure 5. Comparison of time taken to configure new switches

Another benefit of the ZTP-based approach was the lack of need for administrators to manually be in contact with the switches. In cases where no network administrators were present to plug new switches into the firewall that was hosting the DHCP server, other employees with access to storage rooms could easily be guided to perform the necessary tasks.

5.3 Answers to research questions

In the opening chapter of the thesis, three research questions were formulated. In this section the author aims to answer them based on the results obtained from the experimental setup used to test the ZTP process.

5.3.1 What are the challenges of manual configuration and how can automation help in addressing the challenges?

Manual configuration was found to be time consuming and more open to personal input from administrators than an automated approach. Due to the small sample size used for testing, a change in errors made during manual configuration and in the automated process was difficult to establish. However, the author believes that using a templated approach can still lead to a reduction in errors as all templates used for configuration can be validated to work and are unlikely to change from one device to another. More validation steps could be added when checking the validity of variable files before building a configuration file from them.

5.3.2 What are the limitations of automating configuration tasks?

Limitations in the process could arise if a wide variety of devices need to be able to be configured with the ZTP process as the DHCP configuration on the SRX300 can provide only one OS image as part of the set DHCP options. If a device that uses a different JunOS image is connected to the wrong port, the ZTP process will be stuck as the device reboots but is unable to boot into the OS, requiring manual intervention to start the device from a recovery snapshot.

5.3.3 What type of skills and knowledge are required by administrators in a ZTP environment?

The skills needed by administrators were seen to align with the requirement for easy management as the naming used in making the variable files was intuitive enough to grasp by someone who had only recently been introduced to the tools used. For management, basic knowledge of Ansible, Python, Jinja2, and intermediate knowledge of the Juniper Command Line Interface was enough to manage and improve on the previously built system.

5.4 Practical implications

The result of the experiment shows a noticeable decrease in time taken to prepare devices that are destined to be deployed in a branch office environment. The results also show an increased capacity to respond to a sudden need for new devices when network administrators are unable to physically move and connect to-be provisioned devices by making the connection process simple for non-technical personnel.

Moreover, the existence of the setup enables to use backed up configuration files more effectively in cases of hardware failure on already deployed hardware by simply uploading backed up configuration files to the ZTP VM, speeding up any restoration process. This would however necessitate the expansion of ports with different DHCP configurations for new device models to ensure that replacement systems are up to date when deployed.

Further consistency improvements could be achieved by making groups that define every parameter for each client and then use the groups instead of having freedom to edit variable names in the files that are used to build device templates.

6 Summary / Conclusions

This thesis explored the implementation of ZTP at TEHIK. With research identifying challenges of manual configuration compared to an automated approach, with a focus on reducing time consumption and increasing consistency across device configurations.

During the experimental phase, an improvement of around 50% was measured in time taken to ready a switch, with more switches needing to be configured, the improvement was even more so noticeable.

It was also seen, that the ZTP setup, other than provisioning switches for new environments, could also be used to quickly restore configurations for replacement switches in cases of hardware failure. The deployment was also seen to be useful for situations where technical personnel was not available to start the provisioning process on new devices, non-technical personnel were easily guided to perform any necessary tasks.

The thesis also brings out some ways to improve the solution to provide better consistency in configuration and reduce error rates by reducing the amount variables that might need to be edited. Due to the small sample size, a rate for errors was not possible to properly establish.

Future work could expand on readying devices for branch environments, such as provisioning firewalls. Another possibility for future work could cover implementation of the solution in a branch environment, simplifying addition of new devices, should the need arise, or provisioning an entire branch office deployment at the branch. Furthermore, the author believes that the work done in this thesis could be changed and built upon to build a ZTP system in other organizations similar in size to TEHIK.

References

[1] M. Aruoja, "Tallinn University of Technology Digital Library," Tallinn University of Technology, 2020. [Online]. Available: https://digikogu.taltech.ee/et/Item/7e3d98e3-20ac-4f75-887b-d562eb9f7429. [Accessed 3 November 2024].

[2] F. S. Bruschetti, J. Guevara, M. C. Abeledo and D. A. Priano, "An Empirical Evaluation of Automated Configuration Tools for Software-Defined Networking: A Usability and Performance Perspective," Ingénierie des Systèmes d'Information, 31 October 2023. [Online]. Available: https://iieta.org/journals/isi/paper/10.18280/isi.280502. [Accessed 2 November 2024].

[3] A. Whaley, "Q&A with DevNet Creator Award Winner – Joel King," Cisco Blogs, 18 August 2020. [Online]. Available: https://blogs.cisco.com/developer/qa-with-joel-king. [Accessed 2 November 2024].

[4] Juniper Networks, "What is SDN?," [Online]. Available: https://www.juniper.net/us/en/research-topics/what-is-sdn.html. [Accessed 14 November 2024].

[5] F. Bannour, S. Souihi and A. Mellouk, "Distributed SDN Control: Survey, Taxonomy, and Challenges," IEEE, 12 December 2017. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8187644. [Accessed 15 November 2024].

[6] H. Ashtari, "What Is Software-Defined Networking (SDN)? Definition, Architecture, and Applications," 10 February 2022. [Online]. Available: https://www.spiceworks.com/tech/networking/articles/what-is-sdn/. [Accessed 27 November 2024].

[7] Red Hat, "Ansible in depth," 2017. [Online]. Available: https://cdn2.hubspot.net/hub/330046/file-480366556-pdf/pdf_content/Ansible_in_Depth.pdf?t=1390852822000. [Accessed 16 November 2024].

[8] S. Wågbrant and V. D. Radic, "AUTOMATED NETWORK CONFIGURATION," 6 June 2022. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1667034/FULLTEXT01.pdf. [Accessed 16 November 2024].

[9] J. Johansson, "A COMPARISON OF RESOURCE UTILIZATION AND DEPLOYMENT TIME FOR OPEN-SOURCE SOFTWARE DEPLOYMENT TOOLS," 2017. [Online]. Available: https://www.diva-portal.org/smash/get/diva2:1117279/FULLTEXT01.pdf. [Accessed 17 November 2024].

[10] Juniper Networks, "Zero Touch Provisioning," [Online]. Available: https://www.juniper.net/documentation/us/en/software/junos/junos-install-upgrade/topics/topic-map/zero-touch-provision.html. [Accessed 10 November 2024].

[11] Palo Alto Networks, "What Is Zero Touch Provisioning (ZTP)?," [Online]. Available: https://www.paloaltonetworks.com/cyberpedia/what-is-zero-touch-provisioning-ZTP. [Accessed 12 November 2024].

[12] F. Compare, "Network Configuration Manual vs Automated," Flare Compare, 28 October 2021. [Online]. Available: https://flarecompare.com/Networking/Network%20configuration%20%20manual%20vs%20automated/. [Accessed 3 November 2024].

[13] Juniper Networks, "EX2300 Ethernet Switch Datasheet," [Online]. Available: https://www.juniper.net/us/en/products/switches/ex-series/ex2300-ethernet-switch-datasheet.html. [Accessed 14 November 2024].

[14] Juniper Networks, "EX2300-C Compact Ethernet Switch Datasheet," [Online]. Available: https://www.juniper.net/us/en/products/switches/ex-series/ex2300-c-compact-ethernet-switch-datasheet.html. [Accessed 14 November 2024].

[15] Juniper Networks, "SRX300 Line of Firewalls for the Branch Datasheet," [Online]. Available: https://www.juniper.net/us/en/products/security/srx-series/srx300-line-firewalls-branch-datasheet.html. [Accessed 14 November 2024].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.

Appendix 2 – Ansible playbook

```
- name: testconf generation
  hosts: ZTP_VM
  gather_facts: yes
  vars_files:
    - "all_vars/common.yml"
    - "host_vars/{{ device_ip }}.yml" #"host_vars/tst-sw.yml"
  vars:
    # Specifies which file to use, as all variable files are named
after their IP
    device_ip: 10.20.30.2
    # Specify target directory
    target_dir: sw1
    conf_target: ZTP_VM

  tasks:

    # Valitation
    - name: Validate all VLANs in ranges are declared
      assert:
        that:
          - "'{{ item.value.vlan }}' in switch.vlans"
        fail_msg: "VLAN '{{ item.value.vlan }}' is referenced in range
'{{ item.key }}' but not declared under vlans."
      loop: "{{ switch.ranges | dict2items }}"
      loop_control:
        label: "{{ item.key }}"

    - name: Validate that each range_member key exists in ranges
      assert:
        that:
          - item.key in switch.ranges
        fail_msg: "Range member '{{ item.key }}' is not defined in
ranges."
      loop: "{{ switch.range_members | dict2items }}"
      loop_control:
        label: "{{ item.key }}"

    - name: Validate VLANs for each separate interface entry
      assert:
        that:
          # Check if the port VLAN is 'all' or contains VLANs that are
declared
          - item.value.vlan == 'all' or
```

```
            (item.value.vlan | regex_replace('[\\[\\]]', '') | split('
') | difference(switch.vlans) | length == 0)
        fail_msg: "VLAN(s) '{{ item.value.vlan }}' in
separate_interfaces '{{ item.key }}' are not declared in the vlans
list."
      loop: "{{ switch.separate_interfaces | dict2items }}"
      loop_control:
        label: "{{ item.key }}"

    # Rendering
    - name: Render ex template
      template:
        src: roles/ex/templates/main.conf.j2
        dest: /var/www/templates/rendered_main.conf.part

    - name: Render common template
      template:
        src: roles/common/templates/common.conf.j2
        dest: /var/www/templates/rendered_common.conf.part

    - name: Assemble configuration
      assemble:
        src: /var/www/templates/
        dest: /var/www/templates/assembled.config
        remote_src: true
        regexp: '^rendered_'

    - name: Upload the rendered file to the target server
      become: yes
      copy:
        src: /var/www/templates/assembled.config
        dest: /var/www/config/{{ target_dir }}/sw.config
        mode: '0777'
        remote_src: true
      # On success, notify that conf was sent
      notify:
        - SentHandler

  handlers:
    - name: SentHandler
      debug:
        msg: "Configuration sent."
```

Appendix 3 – main.j2 template file

```
system {
    host-name {{ switch.hostname }};
}
interfaces {
{% for range_name, range_data in switch.ranges.items() %}
    interface-range {{ range_name }} {
        member-range {{ range_data.start }} to {{ range_data.end }};
{% if switch.range_members is defined and range_name in
switch.range_members %}
{% for member_interface in switch.range_members[range_name] %}
        member {{ member_interface }};
{% endfor %}{% endif %}
        unit 0 {
            family ethernet-switching {
                interface-mode {{ range_data.mode }};
                vlan {
                    members {{ range_data.vlan }};
                }
            }
        }
    }
{% endfor %}
{% for iface_name, iface_data in switch.separate_interfaces.items() %}
    {{ iface_data.port }} {
        unit 0 {
            family ethernet-switching {
                interface-mode {{ iface_data.mode }};
                vlan {
                    members {{ iface_data.vlan }};
                }
            }
        }
    }
{% endfor %}
    irb {
{% for unit, unit_data in switch.irb.items() %}
        {{ unit }} {
            family inet {
                address {{ unit_data.address }};
            }
        }
{% endfor %}
    }
}
```

```
{% set ip_parts = device.mgmt.ip.split('.') %}
routing-options {
    static {
        route 0.0.0.0/0 next-hop 10.20.{{ ip_parts[2] }}.1;
    }
}
protocols {
    dot1x {
        authenticator {
            authentication-profile-name aruba-clearpass;
            interface {
{% for range_name, range_data in switch.ranges.items() %}
{% if range_data.dot1x == 'y' %}
                {{ range_name }} {
                    supplicant multiple;
                    transmit-period 2;
                    mac-radius;
                    server-fail permit;
                }
{% endif %}{% endfor %}
{% if switch.dot1x_disable is defined and switch.dot1x_disable %}
{% for interface in switch.dot1x_disable %}
            {{ interface }} {
                    disable;
                }
{% endfor %}
{% endif %}
            }
        }
    }
    lldp {
        management-address {{ device.mgmt.ip }};
        interface all;
    }
    rstp {
        interface all;
    }
}
{% if 'VOIP' in switch %}
switch-options {
{% set voip_interface = switch.VOIP.interface %}
{% if voip_interface %}
    voip {
        interface {{ switch.VOIP.interface }} {
            vlan VOIP;
            forwarding-class assured-forwarding;
        }
    }
{% endif %}
}
{% endif %}
vlans {
{% for vlan_name, vlan_data in switch.vlans.items() %}
    {{ vlan_name }} {
    {% if 'description' in vlan_data %}
    description "{{ vlan_data.description }}";
    {% endif %}
    vlan-id {{ vlan_data.id }};
    {% if 'irb' in vlan_data %}
    l3-interface irb.{{ vlan_data.irb }};
    {% endif %}
}
{% endfor %}
}
```

Appendix 4 – common.j2 template file

```
system {
    root-authentication {
        {{ common.root.key }}; ## SECRET-DATA
    }
    login {
        class restoration {
            permissions [ access admin firewall flow-tap interface
network routing secret security snmp storage system trace view view-
configuration ];
        }
        class ssh-timeout {
            idle-timeout 120;
            permissions all;
        }
        user backup {
            uid 2020;
            class super-user;
            authentication {
                {{ common.users.hadaline.key }}; ## SECRET-DATA
            }
        }
        user restore_user {
            uid 2002;
            class restoration;
            authentication {
                {{ common.users.rancid.key }}
            }
        }
        user remote {
            uid 2010;
            class super-user;
        }
    }
    services {
        ssh {
            root-login deny;
            protocol-version v2;
            max-sessions-per-connection 32;
            sftp-server;
        }
        netconf {
            ssh;
        }
    }
    auto-snapshot;
    time-zone Europe/Tallinn;
    authentication-order [ radius password ];
    ports {
        console log-out-on-disconnect;
    }
    radius-server {
        {{ common.radius.server }} {
            secret "{{ common.radius.secret }}"; ## SECRET-DATA
        }
    }
    radius-options {
        password-protocol mschap-v2;
    }
    accounting {
        events [ login change-log interactive-commands ];
        destination {
            radius {
                server {
                    {{ common.accounting.server }} {
                        secret "{{ common.accounting.secret }}"; ##
SECRET-DATA
                        source-address {{ device.mgmt.ip }};
                    }
                }
            }
        }
    }
    syslog {
        user * {
            any emergency;
        }
        host {{ common.syslog}} {
            authorization any;
            security any;
            change-log any;
            source-address {{ device.mgmt.ip }};
            explicit-priority;
        }
    }
    ntp {
{% for ntp in common.ntp %}
        server {{ ntp }};
{% endfor %}
        source-address {{ device.mgmt.ip }};
    }
}
```

```
chassis {
    alarm {
        management-ethernet {
            link-down ignore;
        }
    }
}
snmp {
    location "{{ switch.snmp.location }}";
    v3 {
        usm {
            local-engine {
                user TEHIKV3 {
                    authentication-sha {
                        authentication-password {{ common.snmp.auth
}}; ## SECRET-DATA
                    }
                    privacy-aes128 {
                        privacy-password {{ common.snmp.privacy }}; ##
SECRET-DATA
                    }
                }
            }
        }
        vacm {
            security-to-group {
                security-model usm {
                    security-name SNMP-GROUP-NAME {
                        group SNMPV3GROUP;
                    }
                }
            }
            access {
                group SNMPV3GROUP {
                    default-context-prefix {
                        security-model usm {
                            security-level privacy {
                                read-view SNMPVIEW;
                            }
                        }
                    }
                }
            }
        }
    }
    view SNMPVIEW {
        oid .1 include;
    }
}
access {
    radius-server {
        {{ common.radius.server }} {
            secret "{{ common.radius.secret }}"; ## SECRET-DATA
            source-address {{ device.mgmt.ip }};
        }
    }
    profile aruba-clearpass {
        authentication-order radius;
        radius {
            authentication-server {{ common.accounting.server }};
            accounting-server {{ common.accounting.server }};;
            options {
                nas-identifier {{ device.mgmt.ip }};
            }
        }
    }
}
```

Appendix 5 – Switch variable file

```
device:
  mgmt:
    ip: 10.20.30.2

switch:
  hostname: "ZTP-TST-EX2300"

  snmp:
    location: "TEHIK"

  ranges:
    CLIENT-A-ACCESS:
      start: ge-0/0/0
      end: ge-0/0/4
      vlan: Client-A
      mode: access
      dot1x: y
    CLIENT-B-ACCESS:
      start: ge-0/0/5
      end: ge-0/0/6
      vlan: Client-B
      mode: access
      dot1x: y
    WIFI:
      start: ge-0/0/7
      end: ge-0/0/9
      vlan: WIFI
      mode: access
      dot1x: n

  range_members:
    CLIENT-A-ACCESS:
      - ge-0/0/11

  separate_interfaces:
    Uplink:
      port: xe-0/1/0
      vlan: all
      mode: trunk
    SW-2:
      port: ge-0/1/1
      vlan: '[Client-A MGMT]'
      mode: trunk
    PRINTER:
```

```
    port: ge-0/0/10
    vlan: PRINTER
    mode: access

irb:
  unit 50:
    address: "{{ device.mgmt.ip }}/24"

vlans:
  Client-A:
    id: 100
  Client-B:
    id: 101
  MGMT:
    description: MGMT
    id: 50
    irb: 50
  WIFI:
    id: 300
  PRINTER:
    id: 301
  VOIP:
    id: 302
```

Appendix 6 – Switch console output

```
Auto Image Upgrade: DHCP INET Options for client interface irb.0
BootFile:
https://10.121.1.11/config/sw1/sw.config ConfigFile:
config/sw1/sw.config
 ImageFile: os/junos-arm-32-21.4R3-S7.6.tgz Gateway: 10.20.200.2 DHCP
Server:

10.20.200.2 File Server: 10.121.1.11 Options state: All options set

Auto Image Upgrade: DHCP INET Client Bound interfaces : irb.0

Auto Image Upgrade: DHCP INET Client Unbound interfaces : vme.0

Auto Image Upgrade: DHCP INET6 Client Bound interfaces :

Auto Image Upgrade: DHCP INET6 Client Unbound interfaces : irb.0 vme.0

Auto Image Upgrade: Active on INET client interface : irb.0

Auto Image Upgrade: Interface::   "irb"

Auto Image Upgrade: Server::      "10.121.1.11"

Auto Image Upgrade: Image File::  "junos-arm-32-21.4R3-S7.6.tgz"

Auto Image Upgrade: Config File:: "sw.config"

Auto Image Upgrade: Gateway::     "10.20.200.2"

Auto Image Upgrade: Protocol::    "https"

Auto Image Upgrade: Start fetching sw.config file from server
10.121.1.11 through irb using https

Auto Image Upgrade: File sw.config fetched from server 10.121.1.11
through irb

Auto Image Upgrade: Start fetching junos-arm-32-21.4R3-S7.6.tgz file
from server 10.121.1.11 through irb using https

Auto Image Upgrade: File junos-arm-32-21.4R3-S7.6.tgz fetched from
server 10.121.1.11 through irb

Auto Image Upgrade: Aborting image installation of junos-arm-32-
21.4R3-S7.6.tgz
```

```
 received from 10.121.1.11 through irb: Installed and fetched image
version same

Auto Image Upgrade: Applying sw.config file configuration fetched from
server 10.121.1.11 through irb
```