

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Leonid Peskov 212957IAAB

Streamlining Network Operations: Implementing Automated Deployment and Monitoring in a Multi-Vendor Environment

Bachelor's thesis

Supervisor: Mohammad Tariq
Meeran (PhD)

Co-Supervisor: Siim Vene (MSc)

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
INFOTEHNOLOOGIA TEADUSKOND

Leonid Peskov 212957IAAB

**Võrguoperatsioonide ühtlustamine:
Automaatne juurutamine ja järelevalve mitme
tootja keskkonnas**

Bakalaureusetöö

Juhendaja: Mohammad Tariq
Meeran (PhD)

Kaasjuhendaja Siim Vene (MSc)

Tallinn 2024

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: Leonid Peskov

Abstract

In the rapidly evolving landscape of network operations, manual processes for deployment and monitoring have become increasingly untenable. Company X, a hypothetical entity operating within this dynamic environment, faces challenges attributed to the inefficiencies and error-prone nature of manual network management - particularly within its multi-vendor network infrastructure. This bachelor thesis tries to meet these burning requirements for automation in network operations towards more efficiency, minimised operational risks, and assurance in rapid adaptability of changing network demands.

This research will delve into the integration of Continuous Integration/Continuous Deployment (CI/CD) practices in network operations using automation tools that include Ansible for configuration management, monitoring using Prometheus and Grafana, and orchestration of the CI/CD pipeline with GitHub Actions and Jenkins. This approach should be meant for the simplification of network management processes by bringing down manual intervention and, above all, allow a quick response to network conditions across a multi-vendor landscape.

This thesis is based on a thorough literature review supported by empirical research in the form of development and testing of a prototype system. The success of the system was tested in a simulation multi-vendor network environment for deployment efficiency, accuracy of monitoring, and general responsiveness towards different devices on a network.

The output of the work was a sophisticated automated system of deployment and observation of change implementation in the network with a possibility to substantially mitigate the problems that companies such as Company X have. This thesis will outline how the process of development was followed, challenges to be faced along the way, and, at the end, establish the successful implementation of an automated NetOps solution. This thesis is written in English and is 83 pages long, with 30 figures and 3 tables.

Annotatsioon

Kiiresti arenevas võrguoperatsioonide maastikus on manuaalsed kasutuselevõtu- ja seireprotsessid muutunud üha enam vastuvõetamatuks. Ettevõtte X, hüpoteetiline ettevõtte, mis tegutseb selles dünaamilises keskkonnas, seisab silmitsi väljakutsetega, mis on tingitud võrgu käsitsi haldamise ebatõhususest ja vigade esinemisest - eriti tema mitme tootja võrguinfrastruktuuris. Käesoleva lõputööga püütakse vastata nendele põletavatele nõudmistele automatiseerida võrguoperatsioone, et suurendada tõhusust, minimeerida tegevusriske ja tagada kiire kohanemisvõime muutuvate võrgu nõudmistega.

See bakalaureusetöö käsitleb pideva integratsiooni ja pideva kasutuselevõtu (CI/CD) tavade integreerimist võrguoperatsioonidesse, kasutades selleks automatiseerimise vahendeid, mille hulka kuuluvad Ansible konfiguratsiooni haldamiseks, järelvalve Prometheus ja Grafana abil ning CI/CD-putke orkestreerimine GitHub Actions ja Jenkinsi abil. See lähenemisviis peaks olema mõeldud võrgu haldamise protsesside lihtsustamiseks, vähendades käsitsi sekkumist, ja eelkõige võimaldama kiiret reageerimist võrgu oludele mitme tootja maastikul.

Käesolev lõputöö põhineb põhjalikul kirjanduse ülevaatusel, mida toetavad empiirilised uuringud prototüüp süsteemi väljatöötamise ja testimise näol. Süsteemi edukust testiti simulatsiooniga mitme tootja võrgukeskkonnas, et kontrollida kasutuselevõtu tõhusust, jälgimise täpsust ja üldist reageerimisvõimet võrgu erinevate seadmete suhtes.

Töö väljundiks oli keerukas automatiseeritud süsteem, mis võimaldab muudatuste rakendamise ja jälgimise keerukust võrgus oluliselt leevendada probleeme, mis on sellistel ettevõtetel nagu ettevõtte X. Käesolevas lõputöös kirjeldatakse, kuidas järgiti arendusprotsessi, väljakutseid, millega tuli selle käigus silmitsi seista, ja lõpuks tehakse kindlaks automatiseeritud NetOps-lahenduse edukas rakendamine. Käesolev lõputöö on kirjutatud inglise keeles ja on 83 lehekülge pikk ning sisaldab 30 joonist ja 3 tabelit.

List of abbreviations and terms

AI	Artificial intelligence
CI/CD	Continuous Integration and Continuous Deployment
CLI	Command Line Interface
MIB	Management Information Base
ML	Machine learning
NAT	Network Address Translation
NetDevOps	A blend of Network, Development, and IT Operations
NetOps	Network Operations
NetSecOps	Collaboration between network operations (NetOps) and security operations (SecOps) teams
OID	Object identifier
SDN	Software-Defined Networking
SNMP	Simple Network Management Protocol
SNMP	Simple Network Management Protocol
TCO	Total Cost of Ownership
YAML	Yet Another Markup Language

Table of contents

1 Introduction.....	11
1.1 Problem Statement.....	11
1.2 Goal of the Thesis.....	12
1.3 Objectives of the Research.....	12
1.4 Research Questions.....	12
2 Literature review.....	13
2.1 Overview of Network Operations (NetOps).....	13
2.2 Challenges in Manual Network Deployment and Monitoring.....	14
2.3 DevOps Integration in Network Management.....	15
2.4 Review of Network Automation Tools and Techniques.....	17
2.4.1 Ansible for Network Configuration and Automation.....	17
2.4.2 GitHub Actions for Continuous Integration and Deployment.....	18
2.4.3 Prometheus and Grafana for Network Monitoring.....	19
2.5 Continuous Integration and Continuous Deployment (CI/CD) in NetOps.....	19
2.6 Best Practices in Automated Network Operations.....	21
2.6.1 Network Automation and Virtualization.....	22
2.6.2 NetSecOps Integration.....	22
2.6.3 Embracing Cloud and Consolidation.....	22
2.6.4 Continuous Improvement and Innovation.....	22
3 Methodology.....	24
3.1 Research Method.....	24
3.2 Data Collection.....	24
3.3 Testing Environment.....	25
4 Experimental Design.....	26
4.1 Requirements.....	26
4.2 Tool Selection and Rationale.....	28
4.3 Network Environment and Topology Overview.....	30
4.4 Automation Strategy.....	33
4.4.1 Ansible Setup and Configuration.....	33
4.4.2 Monitoring with Prometheus and Grafana.....	33
4.4.3 CI/CD with GitHub Actions and Jenkins.....	34
5 Implementation.....	36
5.1 Project Structure and Setup.....	36
5.2 Ansible Implementation.....	40
5.2.1 Initialization (Init Role).....	40
5.2.2 Prometheus, Grafana, and SNMP Exporter (Monitoring Roles).....	41
5.2.3 Network Device Configuration (Cisco, Juniper, Mikrotik Roles).....	44
5.2.4 Network Infrastructure Configuration (Master Playbook: infra.yaml).....	45

5.3 Continuous Integration/Continuous Deployment (CI/CD).....	47
5.3.1 Continuous Integration (CI).....	47
5.3.2 Continuous Deployment (CD).....	50
5.4 Monitoring Setup.....	52
5.4.1 Prometheus Configuration.....	52
5.4.2 SNMP Configuration Generation.....	53
5.4.3 Grafana Dashboards and Configuration.....	53
6 Test Scenario: Implementing Network Changes through the CI/CD Pipeline.....	55
6.1 Scenario Overview.....	55
6.2 Step 1: Initiating the Change.....	55
6.3 Step 2: Continuous Integration Process.....	56
6.4 Step 3: Continuous Deployment.....	59
6.5 Step 4: Post-Deployment Verification.....	61
7 Results and discussion.....	63
7.1 Addressing Research Questions.....	63
7.2 Achievement of Objectives and Observations:.....	64
7.3 Conclusions and Future Directions.....	65
References.....	66
Appendix 1 - Configuration Details for the Prometheus Role.....	69
Appendix 2 - Detailed Ansible Tasks for Grafana Setup.....	70
Appendix 3 - Detailed Ansible Tasks for SNMP Exporter Setup.....	72
Appendix 4 - Detailed Ansible Tasks for Cisco Role.....	74
Appendix 5 - Detailed Ansible Tasks for Juniper Role.....	75
Appendix 6 - Detailed Ansible tasks for Mikrotik Role.....	76
Appendix 7 - Detailed GitHub CI Workflows.....	78
Appendix 8 - Detailed Configuration of Dynamic Prometheus.....	80
Appendix 9 - Detailed Grafana Data Source Configuration.....	82
Appendix 10 – Non-exclusive licence for reproduction and publication of a graduation thesis..	83

List of figures

Figure 1. Example of CI/CD pipeline in NetOps (Source:[10]).....	21
Figure 2. Company X's Multi-Vendor Network Topology for Automation Testing.....	30
Figure 3. Ansible configuration file.....	37
Figure 4. Ansible Inventory file.....	38
Figure 5. Ansible Group Variables file.....	39
Figure 6. Ansible init role.....	40
Figure 7. Master playbook infra.yaml.....	46
Figure 8. Continuous Integration Workflow Diagram for Network Automation.....	49
Figure 9. Continuous Deployment Process Diagram.....	51
Figure 10. Active Monitoring Targets in Prometheus.....	53
Figure 11. Grafana Mikrotik Dashboard.....	54
Figure 12. Git commit summary.....	56
Figure 13. GitHub Actions CI Workflow for Syntax and Lint checks.....	57
Figure 14. GitHub Actions CI Workflow for Multi-Vendor Configuration Validation.....	57
Figure 15. Results of CI workflows.....	58
Figure 16. Webhook Response for Development Branch Commit.....	58
Figure 17. Jenkins Job Triggered by Master Branch Commit.....	59
Figure 18. Jenkins job in progress.....	60
Figure 19. Jenkins Console Output - Ansible Playbook Execution.....	61
Figure 20. VLAN Implementation Status Post-Deployment.....	62
Figure 21. Prometheus Role Ansible Tasks.....	69
Figure 22. Grafana Role Ansible Tasks.....	71
Figure 23. SNMP-Exporter Role Ansible Tasks.....	73
Figure 24. Cisco Role Ansible Task.....	74
Figure 25. Juniper Role Ansible Task.....	75
Figure 26. Mikrotik Role Ansible Task.....	77
Figure 27. Workflow for Ansible Syntax and structure checks.....	78
Figure 28. Workflow for Batfish Network Configuration Analysis.....	79
Figure 29. Dynamic Prometheus Configuration.....	81
Figure 30. Dynamic Grafana Data Source Configuration.....	82

List of tables

Table 1. Rational of selected tools.....	28
Table 2. Company X Network Devices and Roles.....	31
Table 3. Key Metrics for Network Monitoring with Prometheus and Grafana.....	33

1 Introduction

Network operations (NetOps) are a methodical approach of maintaining and ensuring effectiveness of the modern IT infrastructures. Today, when network complexities and fragmentation of vendors keep increasing every day, traditional manual processes are becoming unsustainable for deploying and monitoring changes to the network. Besides overall slowing down of response times toward network demands, such manual procedures pose errors and inefficiency risks, arising from being manually processed over networks.

All this, in the end, will result in greater operational bottlenecks for companies operating in such environments, including our hypothetical "Company X." Such challenges will definitely give rise to greater downtimes, which in turn always mean delays in responding to network-related issues and also failing to match up with the demands of this ever-growing dynamism related to modern network infrastructures. This, in return, is reflected in the quality of the service provided to the customer with increased operational costs and the consumption of resources.

A shift toward NetOps automation is mission-critical to deliver against these challenges, offering the promise of greater efficiency, accuracy, and agility in network management. Automation enables the possibility of quick deployment and exact observation regarding changes in network services while it reduces manual invasion of task execution and reacts proactively upon the conditions that are characteristic of a network.

1.1 Problem Statement

Company X operates in a demanding multi-vendor network environment where an environment that dwelled upon manual deployment and monitoring till now has been represented as the organisation's response to the challenge, causing inefficiencies, operational risks, and loss of productivity. Such processes still remain without an

automated tool, making it even harder for Company X to rapidly adapt its service offerings and address emerging network demands.

1.2 Goal of the Thesis

This thesis mainly aims to design and implement an automatic deployment and monitoring system for all changes being done manually over the network at Company X. To decrease manual intervention, increase operational efficiency, and find that the system is geared up to respond quickly to changing network conditions. It is going to be developed in a multi-vendor environment with views of compatibility and integrability under security policies and requirements for Company X scalability.

1.3 Objectives of the Research

- To explore and evaluate existing technologies and methodologies for automating NetOps.
- To design an automated deployment and monitoring system that meets the specific needs of Company X.
- To implement and validate the effectiveness of the proposed system in a simulated multi-vendor network environment.

1.4 Research Questions

- What are the current challenges in manual network deployment and monitoring processes?
- How can automation transform NetOps to meet the demands of modern network infrastructures?
- What would constitute an effective automated deployment and monitoring system for a multi-vendor environment like Company X's?

2 Literature review

With the rapidly changing landscape of network operations, enterprises and organisations face the critical challenge of maintaining reliable, scalable and efficient IT infrastructures.

The requirement for agility and responsiveness in the management of networks has increasingly followed the most paradigmatic shift toward automation and integration. This section presents a review and analysis on the intersection of NetOps and DevOps principles, focusing on the role that Continuous Integration and Continuous Deployment (CI/CD) play in transforming network management.

It revisits the challenges of manual network deployment and monitoring, the role of DevOps in managing networks, and introduces new tools and methods for network automation. The review explores monitoring approaches and the best practices of automated network operations, shaping the development of a comprehensive, high-level network DevOps infrastructure. This analysis, based on modern literature, aims to offer insights on optimising network performance and reliability, crucial for supporting the dynamic needs of modern digital businesses.

2.1 Overview of Network Operations (NetOps)

The evolution of Network Operations (NetOps) was a response to the increasing complexity and dynamic requirements of today's network infrastructures. Conventionally, managing networks was straightforward, but with the arrival of new complex technologies like software-defined networking (SDN) and cloud-based services, the approach for network operation should have been changed. These modern infrastructures are characterised by their fluidity and the dynamic tempo at which changes need to be deployed, often stretching the capabilities of manual management methods to their limits [1].

The primary challenge has been the balancing act between maintaining high service levels and adapting to the expanding scale and complexity of network environments.

Most of the changes in the network were all brought in by hand, in so doing, bringing a lot of unsafe conditions, including waste of time taken through deployment, high susceptibility to errors, and non-maintenance of constantly changing compliance with security standards [1]. Mostly, they happen to be slow operations, time-consuming and executed with a lot of human inaccuracy error and are definitely not meant to be sustainable in the long run.

Moreover, there was pretty much a relatively new layer of complexity that sliced through the transition toward software-defined infrastructures. It, therefore, demanded more agile and flexible network operations that would transit with the pace at which the environment changes. This necessity for agility underscores the importance of streamlining operations, especially in multi-vendor environments, to manage complexity efficiently.

Much of the evolution within NetOps has followed radical solutions to some of the most perennial challenges within the dynamic operating environment. Historicized integration of advanced analytics automated and machine learning environments within NetOps and network management practices shall entail exceedingly highly advanced technological framework that shall drive operational efficiencies up, drive down the total cost of ownership (TCO), and improve maintained or enhanced service level despite the modern character towards which network infrastructures are increasingly coming to be characterised [1]. Such technologies will assist the organisation to achieve a more proactive and predictive approach towards managing the network, ensuring a resilient and secure network that caters to the dynamic business needs.

The move, which is very important, underscores a recognition of the role that modern enterprises play in the execution of efficient, scalable, and automated network operations. It embodies the will to push back against rather traditional challenges and the impetus to take opportunities thrown up by digital network transformations [1].

2.2 Challenges in Manual Network Deployment and Monitoring

The urgent need for radical improvement in outdated network deployment and monitoring practices is underscored by the inefficiencies and vulnerabilities of manual operations. Legacy networks, predominantly CLI-based and inflexible, are ill-equipped to meet the dynamic demands of modern digital environments. Shah and Dubaria (2019)

emphasise the necessity of embracing programmability and automation in network operations to enhance flexibility, reliability, and speed, thus overcoming the limitations of manual processes [2].

In 2019, Shah and Dubaira quoted the McKinsey research, saying that 95% of the changes within the network are still done manually, leading to 70% policy violations due to human mistakes [2]. This alone speaks of inefficiencies in operations and even more for inefficiencies in the financial aspect, as already \$60 billion was spent just on labour and tools for network operations [3]. Furthermore, the average time to resolve network issues can extend up to 5 hours, severely impacting network uptime and, by extension, business operations [2]. The transition to automation emerges as a pivotal strategy for streamlining deployment and monitoring processes across diverse vendor systems, addressing these inefficiencies head-on.

The main challenge in the manual deployment and monitoring through the network is human error. In one of the studies by Cisco, it is indicated that 74% of operators suggest that network changes have a great impact on business, while 97% of them attribute the outages to human errors [2][4]. These errors cause downtime and also are responsible for 22% of all unplanned outages [2][4]. It's high time urgent solutions, such as automation and programmability, that reduce manual intervention and promise time savings, reduced manual work, and minimised human errors are implemented.

The advent of NetDevOps signifies a pivotal shift towards addressing these challenges by infusing network engineering and operations with DevOps principles. This approach ensures the deployment of a version-controlled infrastructure through automated processes. Moreover, adopting Infrastructure as Code (IaaS) and Network as a Code (NaaS) principles facilitates viewing network configurations and devices akin to software. This perspective enables version control and automated provisioning, further distancing network operations from manual configurations and enhancing network adaptability and efficiency [2].

2.3 DevOps Integration in Network Management

The transformation of DevOps practices with network management will prove to be an evolutionary step toward a foundational shift to a highly agile, effective, and resilient network infrastructure. This is a very complicated transformation and cultural change

towards collaboration and, most importantly, rapid iteration, based on automation and continuous integration. For instance, the Juniper Network's report notes that 75% of Communications Service Providers, 71% of Enterprises, and 78% of Cloud Providers have adopted some form of network automation, underscoring the widespread recognition of DevOps' role in enhancing network operations' agility and efficiency [7].

A key component of this automation is the use of scripting tools, like Python, or the built-from-scratch automation tool that has scripting languages at its very core, like Ansible, which must be developed to effectively configure devices and adapt to the dynamic needs of networks. These tools have been instrumental in reducing repetitive tasks and scaling operational efficiency across diverse networking environments, as highlighted by the Juniper Networks's report's findings that reducing hard and repetitive work is the top driver for network automation across sectors [7].

The transition from manual configurations to automated scripts requires a deep understanding of network protocols, device APIs, and the specific requirements of the network architecture. Furthermore, integrating these automated scripts into a continuous deployment pipeline introduces additional layers of complexity, such as version control, testing, and rollback mechanisms, which are critical for maintaining network stability and security [5].

The implementation of DevOps in the management of networks is not a new technological shift but a radical cultural shift. This cultural shift necessitates a change in skill sets for network professionals, requiring a broader understanding of programming, software lifecycle processes, and agile methodologies. The move away from traditional network management strategies emphasises the need for a new approach to network lifecycle management [6]. Organisations that have embraced this shift, as the Juniper Network's report suggests, are not only meeting but exceeding their operational performance goals, further validating the strategic value of integrating DevOps into network management [7]. Integrating DevOps principles not only fosters collaboration but also significantly contributes to streamlining operations in complex multi-vendor environments by simplifying management tasks and enhancing operational efficiency.

On the other hand, integrating DevOps in network management would need shaking up to the debris of the existing toolset and processes [6]. Traditional methods are often inadequate for supporting the agile, iterative processes at the core of DevOps.

Organisations may need to adapt their current tools or adopt new ones that are more aligned with automation and collaboration principles [6].

This will also require the redefinition of other processes, such as changes moved from the change approval process to the incident response process, redefinition of the network management processes, in line with the increased speed of change and dependency on automatically-driven systems. While all this needs redefinition, it touches not only the technological gyrations but makes the changes in the governance, risk management, and compliance processes, adding layers to the complexity of DevOps integration saga.

The very idea of confluence between network operations and DevOps translates into something much more and well beyond the adoption of a package full of automation scripts. It is a question of cultural change, collaboration, and continuous improvement by raising the majority of the staff—from the network to be very agile and responsive. This includes changes in tools and processes to support a much more agile and responsive network infrastructure. In such complexities and challenges, the strengths in efficiency, flexibility, and security actually make the journey toward integration in DevOps a strategic initiative for any organisation aiming at differentiation in today's modern landscape of contemporary network operations.

2.4 Review of Network Automation Tools and Techniques

This section delves into the practical application, benefits, and impact of leveraging key tools such as Ansible, GitHub Actions, Prometheus, and Grafana for network automation.

2.4.1 Ansible for Network Configuration and Automation

Ansible is a powerful and flexible automation tool that, if not revolutionary, has sparked new life in the way network operations are conducted. Some of its strong points that it has provided at the disposal of network administrators lie in automatically configuring, deploying, and managing instances in various environments. It gives a simple human-readable syntax which allows running the most routine task even without agents and authoring custom scripts.

A notable use case is the automation of configuration changes across multiple devices. Through Ansible's playbooks, network operators can implement changes uniformly, ensuring consistency and reducing the potential for human error. The tool supports a range of modules for different network hardware, enabling seamless integration in multi-vendor environments [8][9]. This versatility is crucial for streamlining network operations, ensuring uniform configuration and automation practices across different vendor products.

The adoption of Ansible in network operations significantly minimises the time and effort associated with manual configurations. Organisations report improvements in deployment speed, operational efficiency, and a reduction in downtime. The automation of routine tasks frees up valuable resources, allowing teams to focus on more strategic initiatives [10].

2.4.2 GitHub Actions for Continuous Integration and Deployment

GitHub Actions significantly boost CI/CD pipelines through delivering automation at different stages of build, testing, and deployment with the help of the automated software development lifecycle. For the matter, in network automation, that can easily kick off Ansible playbooks in deploying network configurations following code changes in the GitHub repository.

Combining Github Actions with Ansible for network configuration updates, automatically sets up a workflow whereby changes in code get added into the network deployment activity, depending on a set of predetermined conditions. But after the integration of the same, any change within your network code automatically releases and propagates through your infrastructure to bring things up to the current state. Such automated workflows are essential for streamlining the deployment process in multi-vendor environments, ensuring that changes are efficiently rolled out across different systems.

This brings a proactive approach to network management, surely reducing the time of the deployment cycle, instilling more reliability when it comes to changes within the network, and supporting more agile ways of response to the needs of the network.

2.4.3 Prometheus and Grafana for Network Monitoring

Prometheus and Grafana can build a powerful combination for monitoring network performance and visualising metrics. Prometheus collects and stores metrics as time series data, which Grafana then visualises through interactive dashboards.

By monitoring a range of metrics, such as network throughput, latency, and error rates, Prometheus provides the data needed to assess network health. Grafana allows operators to visualise these metrics in real-time, offering insights into network performance and identifying potential issues before they escalate [14]. The integration of these tools into network operations is key to streamlining monitoring efforts, providing a unified view of performance across a multi-vendor landscape.

Integration of Prometheus and Grafana with network operations gives an opportunity to a system administrator toward full-scale infrastructure monitoring. In such a setup, it comes to rescue not only for maintaining good network performance but sustaining decision-making under a pool of data-driven insights. It identifies and corrects an issue swiftly, and with that, better network uptime brings a rise in the quality of services provided.

2.5 Continuous Integration and Continuous Deployment (CI/CD) in NetOps

Continuous Integration and Continuous Deployment (CI/CD) stand as pivotal frameworks within the domain of Network Operations (NetOps), facilitating a paradigm shift towards automation and efficiency in managing network configurations and deployments. This transformative approach, leveraging CI/CD pipelines, significantly enhances the agility and reliability of network infrastructure, aligning closely with the dynamic needs of modern digital enterprises. The adoption of CI/CD practices plays a pivotal role in streamlining network changes and deployments, ensuring consistent application across a complex multi-vendor environment with minimal human intervention.

Major emphasis of CI/CD in NetOps lies in automation of the process of integration of new or changed network configurations and ensuring their deployment across a multi-vendor environment with minimal human intervention. Such automation goes

through the whole cycle of network management, from its development, through testing, up to production, which enables adjustment to the call of the network and change of operational efficiency.

In such scenarios, Continuous Integration (CI) in NetOps would translate to an automatic test of the setup as soon as the developer commits the file. In case the same set of rules get instant alerts for every commit, such an issue is captured and rectified way earlier in the development lifecycle than the time the actual deployment would take place, promoting early notice that disturbances would not be at a large scale. This is noted further in the shift to automation of configuration testing, as again reinforced through documents such as "NetDevOps: A New Era Towards Networking & DevOps" and Red Hat on this guide to network automation in the current integrated network automation move [2][10].

Continuous Deployment takes the process forward by one step after Continuous Integration. The term refers to the process of incrementally deploying network changes to the production environment automatically, having released these changes shortly after they have passed all necessary tests. This directly ensures that the changed and tested configurations are forever running in the network. Several important tools required for such a process, like Ansible, which provides network automation operations in different environments, have been described in both documents. Therefore, the use of Ansible in the automation of deployment processes is very much in line with industry-changing Infrastructure as Code (IaC) whereby network configurations will now come under one system of management and deployment that is code-driven [2][10].

CI/CD in a NetOps environment is built on various tools and different applications, which are most likely inquisitive in character since they are set to automate the management life cycle of various beacons around an organisational framework. For example, the likes of Ansible are fond of strong frameworks when it comes to the automation of configuration, deployment, and testing of networks. Another example is the further integration of CI/CD pipelines, which allows for version control like Git and eventually a more manageable and collaborative workable way of controlling every change brought in. Additionally, visualisation will be done on the running monitoring as part of the CI/CD pipeline, which includes tools like Prometheus and Grafana. This, in

turn, will give real-time validation of the performance gain for the deployed network [10].

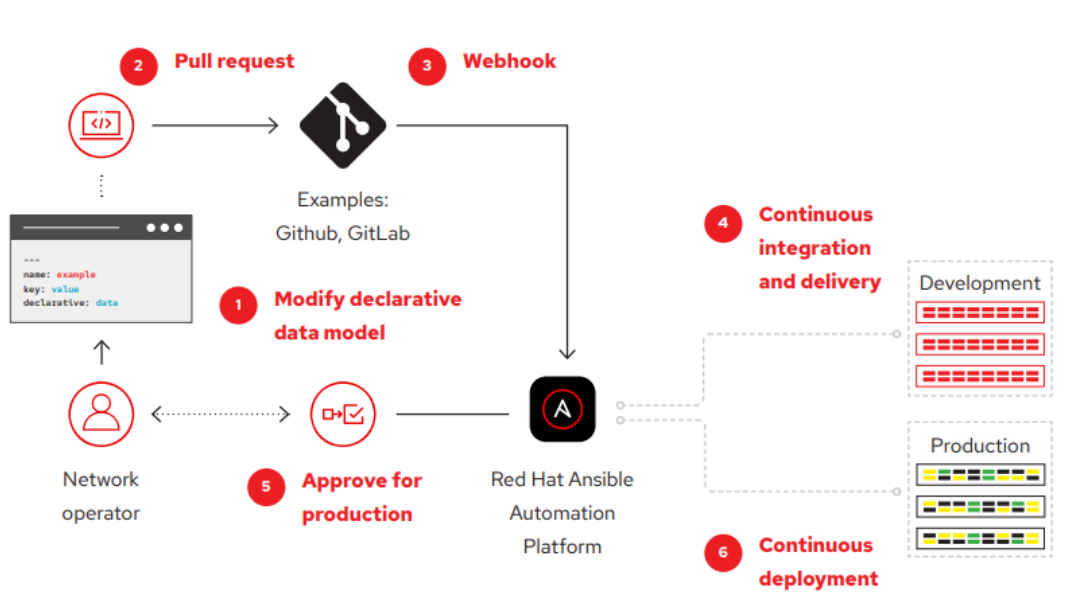


Figure 1. Example of CI/CD pipeline in NetOps (Source:[10])

Adoption of CI/CD in NetOps comes with numerous benefits: reduction in the presence of manually prone errors, betterment of compliance, zoom-in time for the response. It also proposes a set of challenges such as changing the culture demands on organisations, upskilling of members in terms of new automation practices and tools within network teams, and the problems burgeoning in managing CI/CD pipelines in such a multi-vendor realm. This means that the effective approach is, in turn, strategic in the sense of it being strong stakeholders buy-in, constant learning, and the right choice of available tool sets that will be in tune with setting organisational network architecture and operational goals.

2.6 Best Practices in Automated Network Operations

The continued evolution of network operations, raising the level of automation and integration with secure practices, has equally driven adoption of several best practices that support not only enhanced efficiency but also security and resilience. These modern best practices bring up technologies and methodologies to be used. Embracing these practices is essential for streamlining automated network operations, particularly in multi-vendor environments, where the coordination of different technologies and security protocols is paramount.

2.6.1 Network Automation and Virtualization

As network environments become increasingly complex, the role of automation and virtualization technologies becomes crucial. Such packet elaboration and the killing of the costs will be found easiest in Intent-based Networking (IBN), Network Functions Virtualization (NFV), Application Virtual Networks (AVNs), or Software-defined Networking (SDN), amongst the myriad of plenty found here. All will offer technologies that will allow them to automate actions that hold human errors in check, and double the responses. Flexibility, as required, also helps further bring in the needed scalability for improved resource management, and enhanced security [11].

2.6.2 NetSecOps Integration

The integration of network and security operations, known as NetSecOps, emphasises the importance of continuous monitoring, efficient threat intelligence, and automated incident response. Tools for intrusion detection and prevention, vulnerability scanning, and the use of AI and ML for threat detection are pivotal in this approach. Developing clear incident response plans and fostering a culture of security awareness throughout the organisation are recommended to bolster network resilience [12].

2.6.3 Embracing Cloud and Consolidation

This architectural shift to cloud-enabled applications demands an equally strategic shift for network teams to leverage the full benefits of cloud technology. One would argue it moves within the fold of DevOps and security teams, but very clearly, many efficiencies will continue being managed in complex and distributed environments in the process. That is precisely how such an integrated approach is going to streamline operations and best security practices in accordance with digital transformation initiatives set out by organisations [13].

2.6.4 Continuous Improvement and Innovation

Adopting new technologies such as machine learning for network management must be performed in order to automate processes such as root causes of analysis for improved network operations. They must be network team practices that are aligned in seeing to the achievement of the strategic needs of business organisations with a specific intent laid on how the network is used in meeting the business objectives. Continuous

innovation around continuous improvement must not only be directed in the adoption of new technologies but at having the right new technologies adopted in consideration of the strategic needs that the business has [13].

3 Methodology

This section outlines the methodology that was included over the research work for touching on how to develop, deploy, and evaluate an automated network operations (NetOps) system in a multi-vendor environment.

3.1 Research Method

The methodology encompasses both analytical and empirical research approaches to provide a robust foundation for the study.

Analytical Review: This will contain relevant literature explaining the current landscape for automation of operations in the network, CI/CD practices in NetOps, and strategies for monitoring. Author brings in an overview of some current case studies, scholarly articles, and industry reports that were exploring practices, challenges, and the current use of tools in network automation.

Empirical Research: Empirical methods will be applied through the development and deployment of a prototype system. This includes the data collection of the efficiency and performance over the responsiveness of the system to various network devices in a simulated multi-vendor environment. The empirical study will test the prototype against predefined functional and security requirements of a hypothetical organisation, referred to as "Company X".

3.2 Data Collection

Data will be gathered through multiple sources to ensure a comprehensive analysis:

Primary Data Collection: Direct interaction within the operational network environments through simulations reflecting real life and multi-vendor settings. This may be done through a controlled laboratory environment where the network changes are deployed and monitored by the system to be developed.

Interviews and Surveys: Conducting structured interviews with IT professionals and network administrators to gather insights into current challenges, practices, and perceptions towards automation in network operations.

Secondary Data Collection: Reviewing academic literature, industry reports, and documentation on tools like Ansible, Prometheus, Grafana, and GitHub Actions to gather secondary data on network automation trends, tools effectiveness, and implementation strategies.

3.3 Testing Environment

The research will apply a simulated network environment that will emulate a multi-vendor network setup in ascertaining that it is applicable in real setups. This testing environment will involve:

Multi-Vendor Network Emulation: A network consisting of different vendor hardware and software is used to test developed automation tools and processes for compatibility and interoperability.

CI/CD Pipeline Implementation: Implement CI/CD (Continuous Integration/Continuous Deployment) using GitHub Actions and Jenkins workflows for the automation in testing and deploying the changes for network configurations.

Monitor and Measure: Use Prometheus and Grafana for real-time monitoring and visualisation of performance network metrics, in order to ensure that changes deployed produce the desired effect without newly introducing the issues.

This framework methodically aims to develop and further rigorously evaluate an automated system designed specifically for streamlining network operations in the complex, multi-vendor landscape, so as to address the identified inefficiencies and operational challenges within the operational context of Company X.

4 Experimental Design

The experimental design, crucial for validating the system automation before its real-world application into Company X, ensures that any deployments or monitoring adjustments are tested in a controlled, non-disruptive manner. This section details the tools and environment utilised to achieve these objectives, highlighting the technical resources that support the testing and implementation phases.

4.1 Requirements

The push to operationalize the Network in an ideal multi-vendor environment puts in mind the operation under very sensitive tools and strategies. This further demands a solution which is not only effective and efficient in the process of installation but which considers functional flexibility and security of hardware and software-driven diversity environments. The following requirements have been identified as critical to the success of this project for Company X:

Open-Source and Self-Hosted Solutions: Where possible, preferred tools ideally should maintain some element of being open source in nature and self-hosted for high adaptability and low cost of use. It conforms to the highest philosophy in regards to controlling the operations environment of the network. Further taylorization is also possible to ensure no vendor lock-in. Open source software tools typically benefit from robust community support, which offers a wealth of knowledge and resources that drive innovation and problem-solving.

Compatibility with Diverse Network Device Operating Systems and Models: Given the multi-vendor nature of the network environment in Company X, it is essential that the chosen solutions offer broad compatibility across different device operating systems and models. This ensures that the automation and monitoring systems can be universally applied, reducing the need for device-specific configurations and simplifying the management process.

High Performance with Minimal Resource Consumption: The tools must be capable of high-performance operations without imposing significant resource demands on the network infrastructure of Company X. Efficiency in counting the major chances of

development will allow the execution system to serve extensive networks without compounding in performance. Such solutions, walking this tightrope, produce a network that is more responsive, catering as lightly as necessary to the requirements of a modern digital infrastructure.

Comprehensive Community Support and Documentation: Robust community support and comprehensive documentation are invaluable for the rapid deployment, troubleshooting, and evolution of the network operations system that we have in Company X. A vibrant community can offer insights, best practices, and innovative solutions to emerging challenges. Documentation, on the other hand, is essential for onboarding, training, and ensuring that the system can be effectively maintained and scaled over time.

4.2 Tool Selection and Rationale

Critically, the choice of the tools to be used in automation of the network functions was hinged on deducing whether they can meet some criteria seen as critical in their selection, such as working in multi-vendor environments, efficiency, or safety. After an in-depth evaluation, Ansible, Prometheus, Grafana, GitHub Actions, Jenkins and Ngrok were chosen for their specific strengths in deployment automation, real-time monitoring, and CI/CD processes, respectively.

The table below summarises the key features and considerations that led to the selection of these tools:

Table 1. Rational of selected tools

Tool	Key Feature	Benefit
Ansible	Agentless architecture	Simplifies deployment across devices
	Wide support for network devices	Ensures compatibility in multi-vendor setups
Prometheus	Time series data collection	Facilitates real-time monitoring
	Powerful querying capabilities	Enables detailed analysis
Grafana	Interactive dashboards	Enhances data visualisation
	Integration with Prometheus	Streamlines monitoring setup
GitHub Actions	CI/CD integration with GitHub	Automates deployment and testing workflows
	Direct integration in source control	Simplifies management of CI/CD pipelines
Jenkins	Automation server for orchestrating	Continuous Integration

	complex workflows	and Deployment by automating the execution of Ansible playbooks and other tasks upon code commits
Ngrok	Secure tunnelling to expose a local server behind NATs and firewalls to the public internet over secure tunnels.	Enables secure triggering of automation workflows in Jenkins from GitHub without exposing the Jenkins server to potential security risks
SNMP Exporter	Translates SNMP metrics into Prometheus format, leveraging SNMP for network device monitoring	Enhances network visibility and proactive management by integrating SNMP metrics with Prometheus for diverse device monitoring
Batfish	Simulates network behaviour for pre-deployment testing of configuration changes.	Reduces downtime and security risks by verifying changes against policies and detecting errors before application, enhancing network reliability and security

These bundle of tools work together in such a way that they combine to give a homogeneous, effective solution towards the handling, automating deployment and monitoring of very complex multi-vendor network environments, hence making the system to be robust and very flexible in the handling of such dynamic networking environments.

4.3 Network Environment and Topology Overview

This section presents an overview of the network topology where the automation strategy was tested. The topology, as illustrated in Figure 2, consists of a diverse set of networking equipment, indicative of a typical multi-vendor environment.

The complexity and diversity of the network topology under consideration are captured in Figure 2, which presents Company X’s multi-vendor network topology utilised for automation testing:

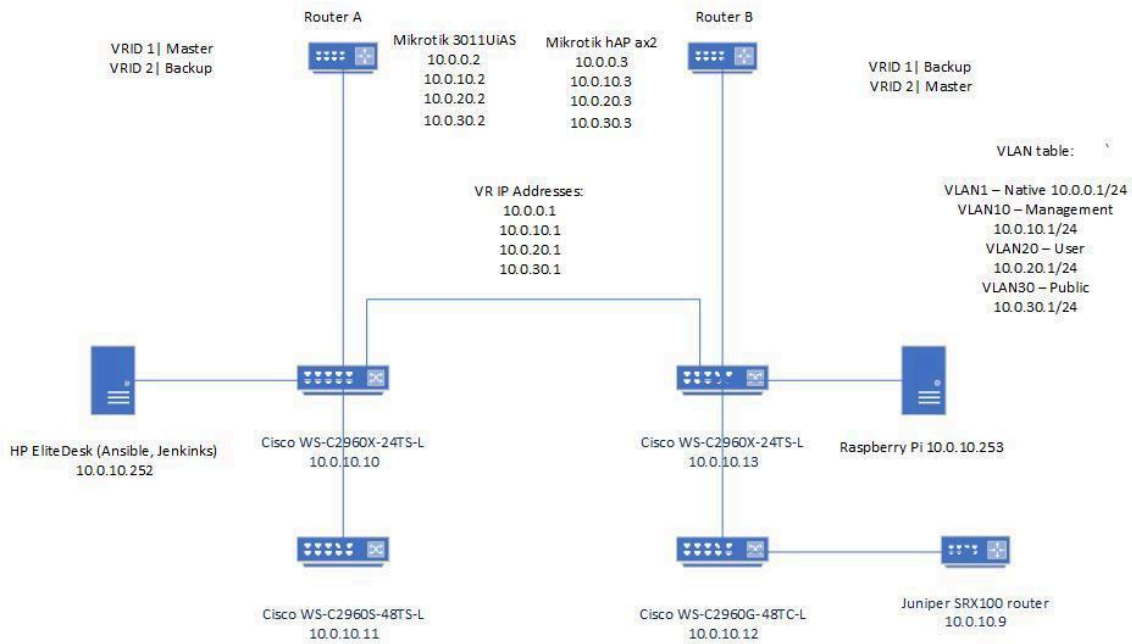


Figure 2. Company X’s Multi-Vendor Network Topology for Automation Testing

The network infrastructure includes devices from Cisco, MikroTik, and Juniper, which are widely used in industry settings and offer a representative sample for testing automation scripts and monitoring solutions. Detailed information about these devices, including their roles and specific models, is provided in Table 2, 'Company X Network Devices and Roles'. This table outlines the diverse hardware involved in the setup and plays a critical role in illustrating the multi-vendor environment that our system supports.

Table 2. Company X Network Devices and Roles

Device Type	Model	IP Addresses	Role
Control Server	HP - EliteDesk 5	10.0.10.252/24	Central management node for network automation, running Jenkins Server, Ansible, Prometheus, and Grafana
MikroTik Routers	Mikrotik - 3011UiAS	10.0.0.2/24 10.0.10.2/24 10.0.20.2/24 10.0.30.2/24	Main router on the setup with VRRP ability to provide redundant routing between subnets and connection with the network.
	Mikrotik - hAP-ax2	10.0.0.3/24 10.0.10.3/24 10.0.20.3/24 10.0.30.3/24	Redundant router on the setup with VRRP ability to provide redundant routing between subnets and connection with network.
Cisco Switches	Cisco - WS-C2960X-24TS-L	10.0.10.10/24	Access layer connectivity, VLAN segmentation
	Cisco -	10.0.10.11/24	

	WS-C2960S-48TS-L		
	Cisco - WS-C2960S-48TC-L	10.0.10.12/24	
	Cisco - WS-C2960X-24TS-L	10.0.10.13/24	
Juniper Router (is being used as switch)	Juniper SRX100	10.0.10.9/24	Juniper router, which is used as a switch, to provide ability to test JunosOs automatization setups

4.4 Automation Strategy

The automation strategy designed to enhance network operations across Company X's multi-vendor environment will utilise a sophisticated setup involving Ansible for configuration management, Prometheus and Grafana for monitoring, and GitHub Actions with Jenkins for orchestrating continuous integration and deployment (CI/CD) workflows.

4.4.1 Ansible Setup and Configuration

The framework will leverage an organised directory structure within the Ansible environment, defining the roles among such include: **cisco**, **juniper**, **mikrotik** to handle the network device configurations. The master orchestrator is an **infra.yaml** playbook which will set up the environment and initialise with the network infrastructure based on roles encapsulating provided device responsibilities like backups, configurations and updates.

4.4.2 Monitoring with Prometheus and Grafana

Monitoring operations will be achieved with Prometheus, which in this case will scrape and store the network's metrics, and with Grafana, which will visualise by the help of custom dashboards. This combination facilitates real-time network monitoring, focusing on key metrics essential for maintaining optimal network performance.

The specific metrics to be collected via SNMP are described in Table 3.

Table 3. Key Metrics for Network Monitoring with Prometheus and Grafana

Metric Category	Description
Device Availability	Refers to the up/down status of all network devices meant to be connected, and so describable as enabling real-time notifications for outages on a network
Interface Metrics	Network interface metrics show the throughput, packet loss, errors, and discards occurring on the network interface to monitor the efficiency and integrity of the transmitted data
Device Health	Monitor the CPU utilisation, memory consumption, and

	temperature to prevent overuse and overheating
Network Latency	Measures the time data takes to travel from source to destination, crucial for identifying network congestion.
Bandwidth Utilisation	It entails the amount of bandwidth that has been utilised, basically within a given timeframe, so that proper capacity planning can be carried out, which is necessary to avoid network congestion

These types of metrics are very important in ensuring that the Network Operations team picks the performance problems and resolves them in the least time possible in the best possible way, which ensures the network remains healthy and reliable.

4.4.3 CI/CD with GitHub Actions and Jenkins

GitHub Actions enables the automatic validation and integration of code changes within the network automation framework in Continuous Integration. Below are some of the responsibilities carried out by GitHub Actions when changes have been pushed to the repository:

- **Validate the code:** During the runtime it will check for syntax, integrity, quality, and best practices in all YAML files and Ansible playbooks.
- **Review and Approve:** Enforced manually to branch protections, which include rules requiring changes to go through human review and approval from the NetOps team after compliance and oversight has occurred.
- **Feedback Loop:** Sends notifications of code status and changes, ensuring transparency and traceability of edits.

Jenkins plays a paramount role in the continuous deployment process, orchestrating the whole deployment pipeline from code integration with GitHub actions, up to deployment configurations through network infrastructure. Jenkins will be taking the deployment pipeline process ahead by executing a series of major activities, such as:

- **Retrieve Changes:** Auto-fetch validated and approved code changes from GitHub to make sure that the deployment process is being worked upon with updated and approved codebase.

- **Automated deployment:** It will use Ansible playbooks in the process of applying the network configurations across the defined inventory to guarantee a systematic and reliable deployment of configurations across the network. This involves deployment of monitoring setups to keep the monitoring infrastructure in line with the network state.
- **Monitoring and Notifications:** Monitors deployments by logging the status of every task running. Provides a real-time feedback mechanism through the deployment process - such as success notifications or failure alerts, among others - to raise visibility during operations.
- **Post-Deployment Verification:** Optionally, Jenkins can trigger scripts or playbooks that verify the successful application of configurations on network devices, ensuring the expected changes are correctly implemented.

This CI/CD framework, leveraging both GitHub Actions for integration and Jenkins for deployment, providing an effective, open, resilient system for the management of setup management in network configuration and monitoring.

5 Implementation

This chapter focuses on the practical implementation part of the theoretical concepts that have been discussed in the sections above. It details the project's setup, including the directory structure and configuration settings, and describes how these elements support the implementation of network automation across multiple vendor devices. The focus is on the practical steps taken to translate "Experimental Design" into a functional system within Company X's network environment.

5.1 Project Structure and Setup

The project's directory structure was organised to facilitate clarity and ease of access to the various components essential for network automation. At the core of the project is the '**ansible**' directory, which houses the configuration files, inventory, and roles designed to manage different network devices across vendors such as Cisco, Juniper, and MikroTik.

Ansible Configuration (ansible.cfg): Configured to optimise Ansible's operation within the project, including setting the inventory path and disabling host key checking to streamline playbook runs.

For clarity on the Ansible configuration employed in the project, Figure 3 showcases the ansible configuration file:

A screenshot of a text editor window titled 'ansible.cfg' with the path '~/.ansible.cfg'. The window contains the following configuration text:

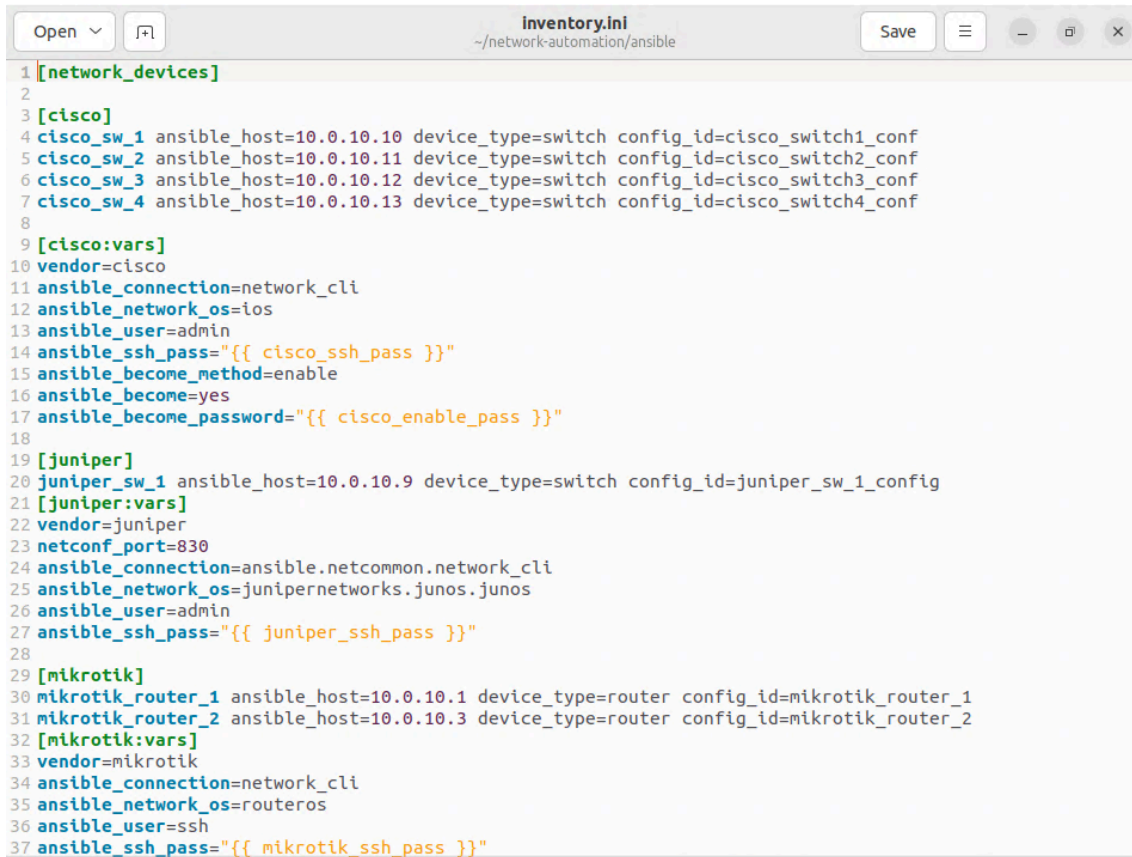
```
1 | defaults|
2 inventory = ./inventory.ini
3 vault_password_file = ~/.ansible/password/vault-pass.txt
4 ansible_python_interpreter = /usr/bin/python3
5 host_key_checking = False
6
7
8 [persistent_connection]
9 connect_timeout = 120
10 command_timeout = 500
11
12 [diff]
13 always = yes
```

The status bar at the bottom indicates 'Plain Text', 'Tab Width: 8', 'Ln 1, Col 1', and 'INS'.

Figure 3. Ansible configuration file

Inventory (inventory.ini): Defines the network devices under management, categorised by vendor, and specifies connection parameters to enable Ansible to communicate with each device securely and efficiently.

The structure of the Ansible inventory, which organises the network devices, is detailed in Figure 4:

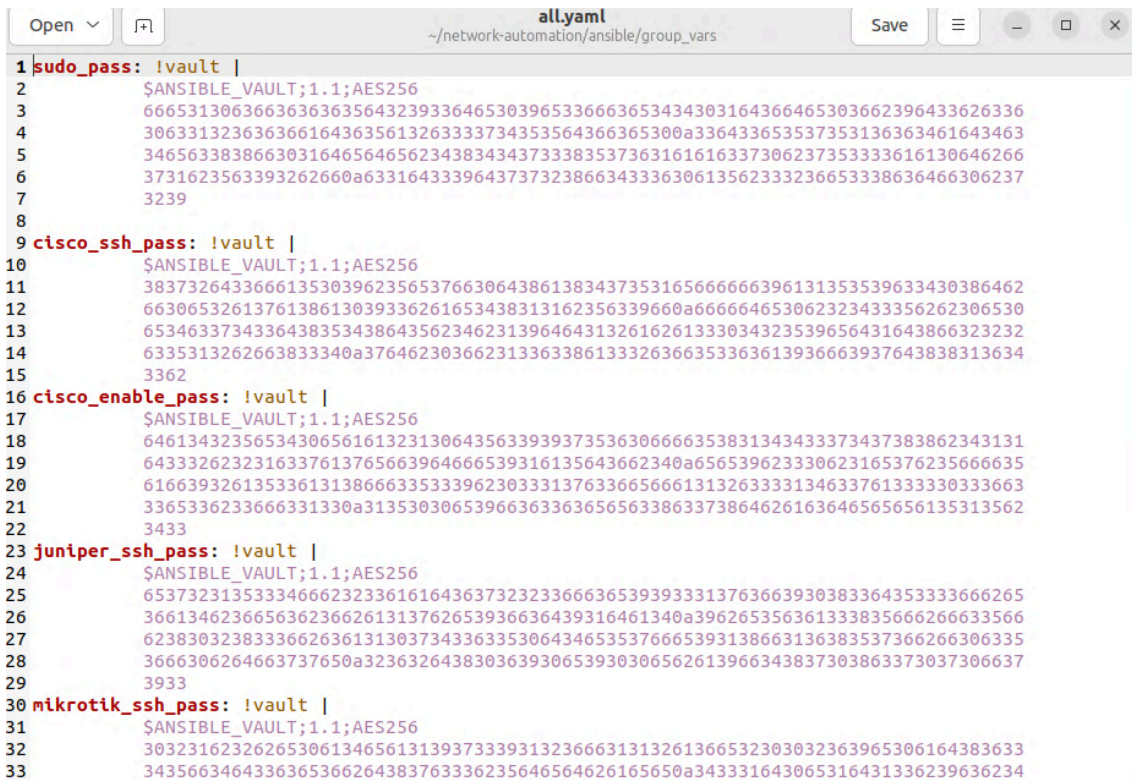


```
1 [network_devices]
2
3 [cisco]
4 cisco_sw_1 ansible_host=10.0.10.10 device_type=switch config_id=cisco_switch1_conf
5 cisco_sw_2 ansible_host=10.0.10.11 device_type=switch config_id=cisco_switch2_conf
6 cisco_sw_3 ansible_host=10.0.10.12 device_type=switch config_id=cisco_switch3_conf
7 cisco_sw_4 ansible_host=10.0.10.13 device_type=switch config_id=cisco_switch4_conf
8
9 [cisco:vars]
10 vendor=cisco
11 ansible_connection=network_cli
12 ansible_network_os=ios
13 ansible_user=admin
14 ansible_ssh_pass="{{ cisco_ssh_pass }}"
15 ansible_become_method=enable
16 ansible_become=yes
17 ansible_become_password="{{ cisco_enable_pass }}"
18
19 [juniper]
20 juniper_sw_1 ansible_host=10.0.10.9 device_type=switch config_id=juniper_sw_1_config
21 [juniper:vars]
22 vendor=juniper
23 netconf_port=830
24 ansible_connection=ansible.netcommon.network_cli
25 ansible_network_os=junipernetworks.junos.junos
26 ansible_user=admin
27 ansible_ssh_pass="{{ juniper_ssh_pass }}"
28
29 [mikrotik]
30 mikrotik_router_1 ansible_host=10.0.10.1 device_type=router config_id=mikrotik_router_1
31 mikrotik_router_2 ansible_host=10.0.10.3 device_type=router config_id=mikrotik_router_2
32 [mikrotik:vars]
33 vendor=mikrotik
34 ansible_connection=network_cli
35 ansible_network_os=routeros
36 ansible_user=ssh
37 ansible_ssh_pass="{{ mikrotik_ssh_pass }}"
```

Figure 4. Ansible Inventory file

Group Variables (group_vars/all.yaml): Contains variables applicable to all hosts, including sensitive information such as passwords encrypted using Ansible Vault for enhanced security.

To provide context on the grouping of variables used across all hosts, Figure 5 depicts the Ansible group variables file:



```
all.yaml
~/network-automation/ansible/group_vars

1 sudo_pass: !vault |
2   $ANSIBLE_VAULT;1.1;AES256
3   6665313063663636363635643239336465303965336663653434303164366465303662396433626336
4   3063313236363661643635613263333734353564366365300a3364336537353136363461643463
5   34656338386630316465646562343834343733383537363161616337306237353333616130646266
6   3731623563393262660a633164333964373732386634333630613562333236653338636466306237
7   3239
8
9 cisco_ssh_pass: !vault |
10  $ANSIBLE_VAULT;1.1;AES256
11  38373264336661353039623565376630643861383437353165666666396131353539633430386462
12  6630653261376138613039336261653438313162356339660a666664653062323433356262306530
13  65346337343364383534386435623462313964643132616261333034323539656431643866323232
14  6335313262663833340a376462303662313363386133326366353363613936663937643838313634
15  3362
16 cisco_enable_pass: !vault |
17  $ANSIBLE_VAULT;1.1;AES256
18  64613432356534306561613231306435633939373536306666353831343433373437383862343131
19  6433326232316337613765663964666539316135643662340a656539623330623165376235666635
20  6166393261353361313866633533962303331376336656661313263333134633761333330333663
21  3365336233666331330a313530306539663633636565633863373864626163646565656135313562
22  3433
23 juniper_ssh_pass: !vault |
24  $ANSIBLE_VAULT;1.1;AES256
25  65373231353334666232336161643637323233666365393933313763663930383364353333666265
26  3661346236656362366261313762653936636439316461340a396265356361333835666266633566
27  62383032383336626361313037343363353064346535376665393138663136383537366266306335
28  3666306264663737650a323632643830363930653930306562613966343837303863373037306637
29  3933
30 mikrotik_ssh_pass: !vault |
31  $ANSIBLE_VAULT;1.1;AES256
32  30323162326265306134656131393733393132366631313261366532303032363965306164383633
33  3435663464336365366264383763336235646564626165650a343331643065316431336239636234
```

Figure 5. Ansible Group Variables file

5.2 Ansible Implementation

This section delves into the implementation of Ansible, detailing how it automates network configuration and deployment across the multi-vendor environment. Ansible plays a main role in the designed system, managing the configuration files, inventory updates, and the execution of roles that ensure consistent and efficient network operations.

5.2.1 Initialization (Init Role)

The initialization role is crucial as it prepares the environment for further automation tasks. It ensures that all necessary system packages and Python libraries are installed and updated on the control node. This step is fundamental to avoid potential issues during the automation process.

System Update and Package Installation: Updates the system's package list and install essential packages, such as python3-pip and sshpass. These packages are necessary for Ansible to communicate with network devices and manage Python dependencies.

Python Library Installation: Install Python libraries like paramiko and ncclient. Paramiko is used for SSH connections to network devices, and ncclient is utilised for managing network configurations via NETCONF.

The initial role in setting up the environment is outlined in Figure 6, showcasing the Ansible init role tasks:



```
1- name: Update apt package list
2  ansible.builtin.apt:
3    update_cache: true
4  become: true
5
6- name: Install system packages
7  ansible.builtin.apt:
8    name:
9      - python3-pip
10     - sshpass
11  state: present
12  become: true
13
14- name: Install Python libraries
15  ansible.builtin.ptp:
16    name:
17      - paramiko
18      - ncclient
19  state: present
20
21- name: Allow Grafana custom port through firewall
22  community.general.ufw:
23    rule: allow
24    port: "[grafana_port]"
25    proto: tcp
26  become: true
```

Figure 6. Ansible init role

5.2.2 Prometheus, Grafana, and SNMP Exporter (Monitoring Roles)

These roles are tasked with setting up the monitoring stack. Prometheus collects metrics from network devices, Grafana provides visualisation dashboards, and SNMP Exporter gathers SNMP metrics for Prometheus. These tools together enable a comprehensive view of the network's performance and health.

Prometheus Role

Prometheus is the open-source monitoring and alerting toolkit that plays a big role in the world of monitoring. It stores metrics from configured targets at configured intervals. The role of Prometheus in this setup involves several key tasks:

- **Installation and Setup:** Prometheus is installed and updated using the system's package manager, ensuring the latest version is deployed for optimal performance and security.
- **Service Management:** The Prometheus service is started and enabled to launch at boot, ensuring continuous monitoring without manual intervention.
- **Configuration Management:** Custom templates (prometheus.yml.j2 and prometheus_defaults.j2) are deployed to configure Prometheus according to the network's specific monitoring needs. This includes defining scrape targets, metrics path, and scrape intervals, tailored to gather relevant data from the network devices.
- **Reload Configuration:** Changes to Prometheus's configuration files trigger a service restart, applying new settings without downtime.

Prometheus' role in delineating the essential tasks for its setup, service management, and dynamic configuration for effective network monitoring is presented in Appendix 1.

Grafana Role

Grafana offers powerful visualisation tools for the metrics collected by Prometheus, enhancing data interpretation through customizable dashboards. In this setup Grafana main tasks include:

- **Repository Setup and Grafana Installation:** Grafana's official repository is added to the system, and the software is installed, ensuring access to the latest features and fixes.
- **Configuration Customization:** A custom grafana.ini configuration file is deployed to tailor Grafana settings, including security options, data source configurations, and general settings that optimize performance and user experience.
- **Data Source and Dashboard Provisioning:** Default data source and dashboard configurations are established using templates (sample.yaml.j2, dashboard.yaml.j2) and JSON files (mikrotik.json, juniper.json, cisco), automating the integration with Prometheus and the setup of network-specific monitoring views.
- **Service Management:** The Grafana service is enabled and started, ensuring its availability for user access and interaction.

Appendix 2 depicts the Ansible tasks for Grafana's setup, detailing the automated steps from installation to dashboard provisioning, vital for visualising network metrics.

SNMP Exporter Role

SNMP Exporter exposes the capability of Prometheus to collect metrics from devices, using the Simple Network Management Protocol (SNMP), a critical component for comprehensive network monitoring. This role encompasses:

- **Binary Management:** The SNMP Exporter binary is downloaded, extracted, and made executable, placed in a designated directory for operational readiness.
- **Configuration Template Deployment:** A customised SNMP configuration (snmp.yml.j2) is applied, defining which metrics to collect from network devices, tailored to the specifics of the monitored environment.
- **Service Creation and Management:** A systemd service file for the SNMP Exporter is created and deployed, ensuring the exporter is properly managed by the system's service manager, including starting at boot and maintaining runtime consistency.
- **Cleanup and Security:** Post-installation cleanup is performed to remove temporary files, and permissions are set to secure the SNMP Exporter's operational environment, aligning with best practices for system security.

Appendix 3 outlines the SNMP Exporter role, highlighting the automated steps for downloading, configuring, and securing the SNMP Exporter for Prometheus:

5.2.3 Network Device Configuration (Cisco, Juniper, Mikrotik Roles)

Cisco Role

Backup Configuration: Backs up the current configuration of Cisco devices as a safety measure before applying any changes.

Configuration Deployment: Applies new configurations to Cisco routers and switches from Jinja2 templates, tailored to Company X's network architecture.

Configuration Verification: Notifies the system to save the running configuration to the startup configuration, ensuring changes persist after a reboot.

To illustrate the Ansible tasks used for the Cisco role, focusing on backup, deployment, and verification processes, Appendix 4 provides a detailed view.

Juniper Role

NETCONF Check and Activation: Verifies if NETCONF is enabled on Juniper devices and activates it if necessary, facilitating remote management.

Configuration Management: Similar to the Cisco role, it backs up configurations and applies new settings using Jinja2 templates. The role also sets the connection method to NETCONF, optimised for Juniper devices.

The process of ensuring NETCONF is enabled on Juniper devices and applying configuration changes is depicted in Appendix 5, highlighting the steps involved in the Juniper role.

MikroTik Role

The MikroTik role is designed to automate the configuration management of MikroTik routers and switches. This role encompasses several key tasks:

Backup Current Configuration: Creates a binary backup of the MikroTik device's current configuration, ensuring that a recovery point is available in case of misconfiguration or other issues.

Export Configuration to .rsc File: Exports the current configuration to a readable .rsc script file, facilitating version control and review of the configurations applied to the devices.

Manual Copy of Backup and Export Files: Utilises sshpass and scp commands to securely transfer the binary backup and exported configuration files from the MikroTik device to the control node. This step is essential for maintaining a repository of device configurations for audit and compliance purposes.

Generate Configuration Script from Template: Leverages Ansible's templating capabilities to generate device-specific configuration scripts from Jinja2 templates. This allows for dynamic creation of configurations based on variables defined in Ansible's inventory and group_vars, ensuring that configurations are tailored to the specific needs of Company X.

Upload and Apply Configuration Script: Securely uploads the generated configuration script to the MikroTik device and applies it using RouterOS commands. This step transforms the templated configuration into the active configuration on the device.

Cleanup of Temporary Files: Ensures that temporary configuration scripts are removed from the control node after they have been successfully applied to the MikroTik device, maintaining a clean working environment.

For the MikroTik role, Appendix 6 visualises the sequence of tasks from backing up configurations to applying changes, demonstrating the comprehensive approach taken for MikroTik devices.

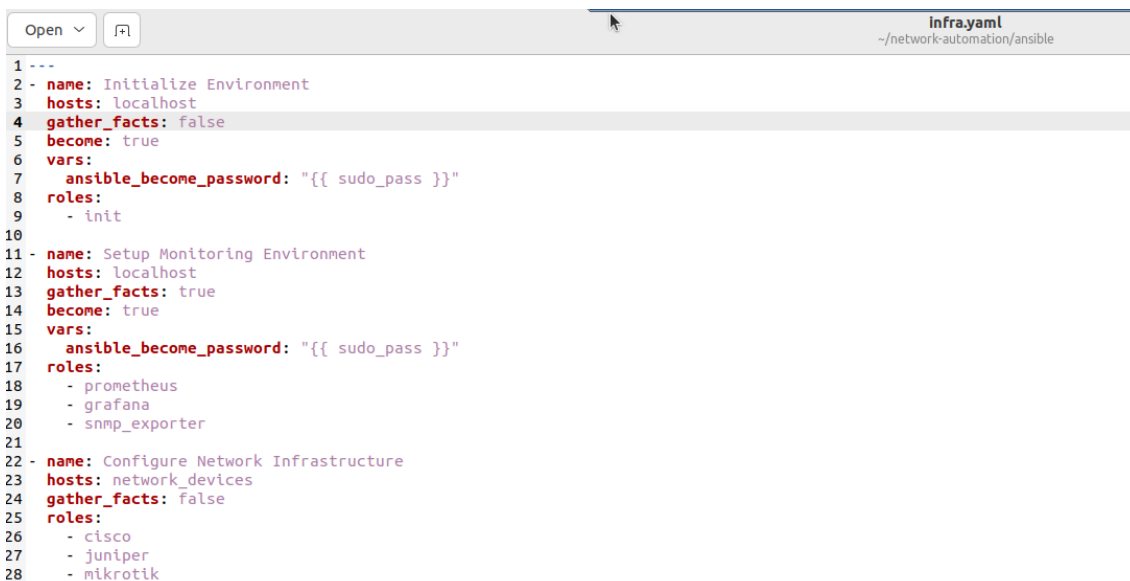
5.2.4 Network Infrastructure Configuration (Master Playbook: infra.yaml)

The master `infra.yaml` playbook serves as the cornerstone for automating the configuration process within Company X's network, delineating the workflow into distinct, well-organised phases.

Initial Setup: This phase focuses on preparing the control node (`localhost`) where Ansible executes. It's pivotal for establishing a solid foundation for the subsequent tasks, ensuring all necessary tools and libraries are in place.

Monitoring Environment Setup: Following the initial preparation, the playbook transitions to configuring the monitoring stack. This critical phase aims to equip Company X with robust monitoring capabilities, enabling proactive network management.

Network Devices Configuration: The final phase is when the playbook applies configurations to the network devices from different vendors. This is the heart of the automation process, where the network's operational parameters are defined and implemented.



```
1 ---
2 - name: Initialize Environment
3   hosts: localhost
4   gather_facts: false
5   become: true
6   vars:
7     ansible_become_password: "{{ sudo_pass }}"
8   roles:
9     - init
10
11 - name: Setup Monitoring Environment
12   hosts: localhost
13   gather_facts: true
14   become: true
15   vars:
16     ansible_become_password: "{{ sudo_pass }}"
17   roles:
18     - prometheus
19     - grafana
20     - snmp_exporter
21
22 - name: Configure Network Infrastructure
23   hosts: network_devices
24   gather_facts: false
25   roles:
26     - cisco
27     - juniper
28     - mikrotik
```

Figure 7. Master playbook `infra.yaml`

5.3 Continuous Integration/Continuous Deployment (CI/CD)

This section involves putting into practice the concepts of Continuous Integration (CI) and Continuous Deployment (CD). This part will delve into how CI/CD is applied to make the reliability and efficiency of the network change enhanced over the infrastructure at Company X. It details the specific tools and workflows used.

5.3.1 Continuous Integration (CI)

Continuous Integration (CI) is an important part of the automated deployment and monitoring system in this project. This is achieved by ensuring all the changes applied to the network configuration are syntactically correct, conforming to the best practices and do not introduce errors, using GitHub Actions and Batfish before the commits are merged into the master branch and deployed.

GitHub Workflows and Branch Protection Rules

GitHub Actions is used to automate the CI process. The above automation is realised through two major workflows: "Network CI - Ansible Syntax and structure checks" and "Network CI - Multi-Vendor Configuration Syntax Check."

The first workflow tests the Ansible playbook and YAML file syntax to follow convention and be error-free. It depicts the steps for setting Python up, the installation of dependencies (Ansible, ansible-lint, yamllint), and doing checks on syntax, linting, and YAML. For a detailed depiction of the workflow used for Ansible syntax and structure checks, please see Figure 27 in Appendix 7.

The second workflow, essential in the validation of multi vendor network configurations, leverages Batfish. This step is designed to analyse the rendered network configurations for different vendors, checking for errors or potential issues that could impact the network's functionality or security. For a detailed depiction of the workflow used for Batfish network configuration analysis, please see Figure 28 in Appendix 7.

To maintain the integrity of the master branch and ensure that only validated changes are merged, branch protection rules are implemented. These rules require that changes undergo review and approval by the NetOps team or designated code owners. This approach guarantees that all modifications are scrutinised, further enhancing the reliability of the network automation process.

Batfish Analysis

Batfish was integrated as a central to the CI process, ensuring that it is also doubled in power as an effective network configuration analysis tool. It ensures the automatic verification of the configurations, which has to meet the sought after policies and be ready for expected behaviour after deployment. For that, the author wrote a special script in Python. It is designed to expand Batfish analysis capabilities using Jinja2 templates for multi-vendor device-type configuration files with caution.

The analysis conducted by Batfish extends to a variety of configuration aspects, offering an in-depth assessment of:

- **Node Properties:** Basic configuration elements, ensuring nodes are correctly identified within the network.
- **Interface Properties:** Verifying the correct setup of interface-level attributes.
- **BGP and OSPF Process Configurations:** Reviewing the critical routing protocol configurations for correctness and compliance with network design.
- **VLAN Properties:** Confirming VLAN assignments and configurations against the network's segmentation strategy.

In addition, Batfish can help detect human error by finding the configuration mistakes, discovering potential security vulnerabilities, and confirming conformance to the organisation's policies. For instance, Batfish checks for proper validation within the **HSRP Properties** in high-availability setups, maps **IP Owners** for proper address allocations, and analyses **Named Structures** like access-lists for proper referencing and definition. It also brings out **Unused Structures**, which maybe are no longer required, making it very easy for the network to operate and thus easy to emanate to network outages.

By integrating Batfish into the CI pipeline, the project removes a barrier that was possibly stopping pre-merge problems from reaching discovery further upstream in development cycles. Such proactivity is of paramount importance for the network to be kept and operated in accordance with the organisational policies and operational requirements that demand resiliency and security without introducing disturbances in the production environment.

The diagram displayed on the Figure 8 below provides a visual representation of the Continuous Integration workflow, mapping the journey from code push or pull request creation by a developer, through the various automated checks, to the eventual merging of code into the master branch upon successful completion and review.

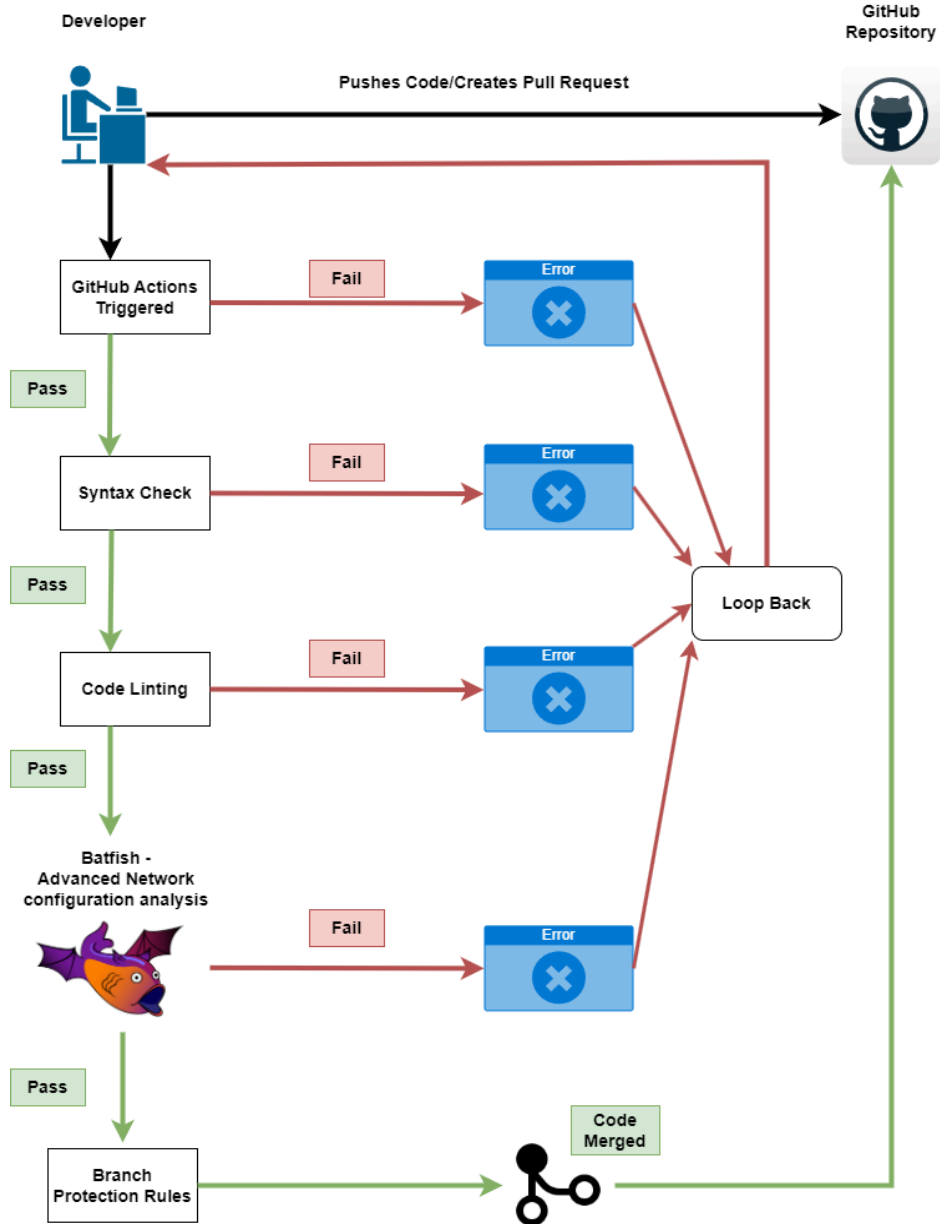


Figure 8. Continuous Integration Workflow Diagram for Network Automation

5.3.2 Continuous Deployment (CD)

The Continuous Deployment (CD) process in this project is meticulously designed to automate the deployment of validated changes from the repository to the production environment. This way, the CD pipeline is configured so that any change that has been made in the master branch is deployed into the production environment automatically and securely without human intervention. The following flow chart will explain the operation process of the CD pipeline.

Jenkins: The Automation Core

Jenkins orchestrates the deployment process itself, interfacing directly with GitHub to pull the latest changes and managing deployment workflow. Its widely built plug-in ecosystem allows interfacing smoothly for both GitHub as source control and Ansible for deployment automation.

GitHub: Source Control Integration

The CD pipeline initiates in GitHub, where there is source code kept together with the configuration files. Integration with Jenkins takes place by the use of webhooks, one of the attributes in GitHub. They are automatically invoked when changes to the master are merged, hence triggering deployment.

ngrok: Secure Connectivity

Ngrok plays a critical role in securely exposing the Jenkins server to the internet, making it accessible to receive webhook notifications from GitHub. This is especially crucial when Jenkins is running in a secure or local environment, allowing for real-time, automated triggers without compromising security.

Ansible: Configuration and Deployment

Ansible is tasked with automating the deployment and application of network configurations. It translates the changes from GitHub into actionable updates across the network, ensuring that the deployments are consistent, repeatable, and scalable, thanks to its agentless architecture and idempotent operations.

To better visualise the CD process described, a flowchart in Figure 9 is provided below. It illustrates the sequence from the initiation of a change in the GitHub repository, through the Jenkins automation sequence, to the final deployment facilitated by Ansible.

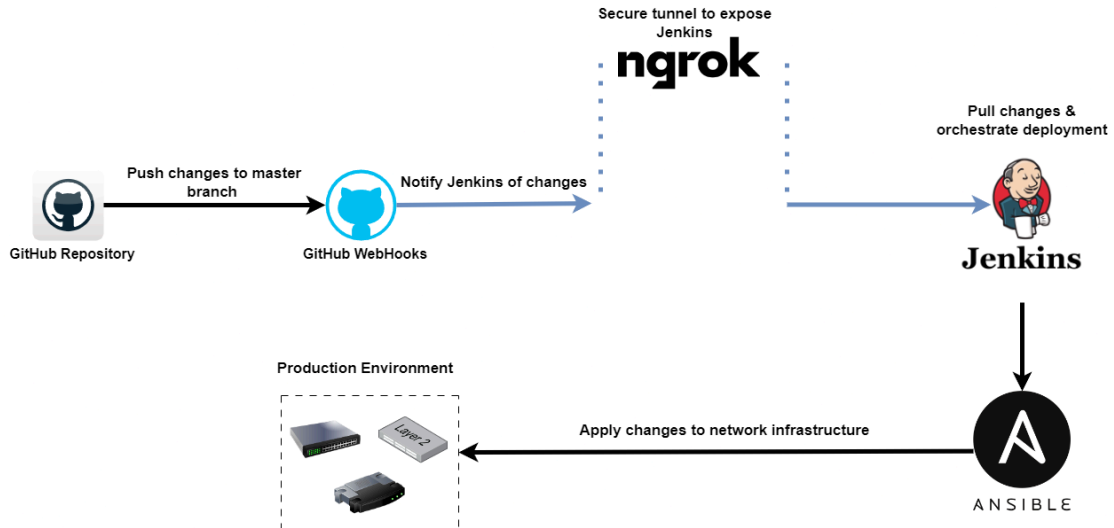


Figure 9. Continuous Deployment Process Diagram

This diagram that is shown in Figure 9 elucidates the step-by-step process of the CD pipeline, capturing the intricacies of each component's role and their interconnections.

5.4 Monitoring Setup

The monitoring setup for this project leverages the dynamic configuration capabilities of Prometheus and the visual excellence of Grafana to ensure scalability and flexibility in a multi-vendor network environment. Using Ansible, the deployment of these monitoring tools, ensuring that our monitoring infrastructure can scale with our network.

5.4.1 Prometheus Configuration

Utilising the **prometheus.yml.j2** template, Prometheus is configured to dynamically scrape metrics from network devices across various vendors, including Cisco, Juniper, and Mikrotik. This approach fosters scalability, as adding new devices to the inventory automatically updates Prometheus's scraping targets without manual intervention.

For a detailed view of the dynamic Prometheus configuration setup, please refer to Figure 18 in Appendix 8. This template demonstrates the use of Jinja2 to dynamically generate job configurations for each host within the Ansible inventory, grouped by vendor. This method allows Prometheus to scrape SNMP metrics tailored to the MIBs supported by each vendor, enhancing the monitoring granularity and relevance.

Directly following the dynamic configuration outlined in Appendix 8 - Figure 19, Figure 10 is a screenshot of the Prometheus targets interface, exemplifying the active and up-to-date status of various network devices. This visual confirmation affirms the successful application of the **prometheus.yml.j2** template, demonstrating Prometheus's capability to dynamically scrape and monitor metrics from multiple devices in a multi-vendor environment.

The screenshot shows the Prometheus Targets page with a table of active monitoring targets. The table has columns for Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error. There are five groups of targets, each with a 'show less' link. The first group is 'snmp_cisco_cisco_sw_1 (1/1 up)', the second is 'snmp_cisco_cisco_sw_2 (1/1 up)', the third is 'snmp_cisco_cisco_sw_3 (1/1 up)', the fourth is 'snmp_cisco_cisco_sw_4 (1/1 up)', and the fifth is 'snmp_juniper_if_mib_juniper_sw_1 (1/1 up)'. Each target is in a 'UP' state and has a 'Last Scrape' time and 'Scrape Duration' listed.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://127.0.0.1:9116/snmp module="if_mib" target="10.0.10.10"	UP	instance="10.0.10.10" job="snmp_cisco_cisco_sw_1"	4.268s ago	490.1ms	
http://127.0.0.1:9116/snmp module="if_mib" target="10.0.10.11"	UP	instance="10.0.10.11" job="snmp_cisco_cisco_sw_2"	21.135s ago	776.5ms	
http://127.0.0.1:9116/snmp module="if_mib" target="10.0.10.12"	UP	instance="10.0.10.12" job="snmp_cisco_cisco_sw_3"	38.112s ago	913.9ms	
http://127.0.0.1:9116/snmp module="if_mib" target="10.0.10.13"	UP	instance="10.0.10.13" job="snmp_cisco_cisco_sw_4"	31.019s ago	379.8ms	

Figure 10. Active Monitoring Targets in Prometheus

5.4.2 SNMP Configuration Generation

The SNMP configuration was carefully crafted using the SNMP Exporter Config Generator, a tool that parses MIBs with NetSNMP and produces configurations for the SNMP exporter. This step was pivotal in ensuring that the SNMP Exporter could understand and translate SNMP data from the devices into Prometheus metrics, based on OIDs and MIBs obtained from vendors' official documentation.

The process involved building the generator from source and generating device-specific SNMP configurations for. This step is crucial for translating SNMP data from network devices into metrics that Prometheus can understand and store.

5.4.3 Grafana Dashboards and Configuration

Grafana was configured to serve as the visual interface for monitoring, with dashboards created for each device type - Cisco, Juniper, and Mikrotik - tailored to the specific metrics relevant to each vendor. These dashboards (**mikrotik.json**, **cisco.json**, **juniper.json**) were integrated into the Ansible playbook, ensuring they are automatically deployed and updated alongside the monitoring infrastructure.

The grafana.ini configuration was adjusted to fit the custom setup, including modifications to the default port to prevent conflicts with other services. Furthermore, a **sample.yaml** file was used to define data sources and deletion policies within Grafana,

ensuring Prometheus is correctly linked as the primary data source for the dashboards. Detailed configurations for the Grafana data source setup can be found in Appendix 9, Figure 30

To exemplify the practical application of the configuration, Figure 11 provides a snapshot of a Grafana dashboard created for MikroTik devices, showcasing the ability to monitor and visualise key metrics such as temperature, voltage, interface status in real-time and others:



Figure 11. Grafana Mikrotik Dashboard

6 Test Scenario: Implementing Network Changes through the CI/CD Pipeline

This test scenario demonstrates the practical application of the Continuous Integration and Continuous Deployment (CI/CD) pipeline developed for Company X. It illustrates the workflow involved in identifying, updating, and deploying network configurations to address both security enhancements and operational needs. By following a real-life inspired scenario, this section aims to showcase the efficiency, security, and agility of the CI/CD pipeline in handling dynamic network changes across diverse network devices.

6.1 Scenario Overview

In this scenario, a NetOps engineer at "Company X" would identify the need to update network configurations, the update would include enhancing the configuration of the network for both security requirements and new operational needs. The required changes were a variety of them, from updating VLANs to firewall rules, in configurations of Mikrotik and Cisco routers. This scenario will follow the workflow right from the initiation of a change by the engineer to the deployment and will, therefore, demonstrate how a CI/CD pipeline implemented within the project ensures efficiency, security, and agility.

6.2 Step 1: Initiating the Change

The NetOps engineer prepares the network changes in a local development environment. After testing the changes locally, they are ready to be pushed to the repository. The changes include:

Cisco Switch Configurations: Adjustments in VLAN settings and SSH configurations across multiple switches to enhance network segmentation and security.

Mikrotik Router Configurations: Updates to firewall rules, DHCP settings, and the addition of new VLANs for improved network management and security.

The engineer commits the changes to a new branch off the development branch, ensuring they follow best practices for version control and peer review. Following Figure 12 shows Git commit summary.

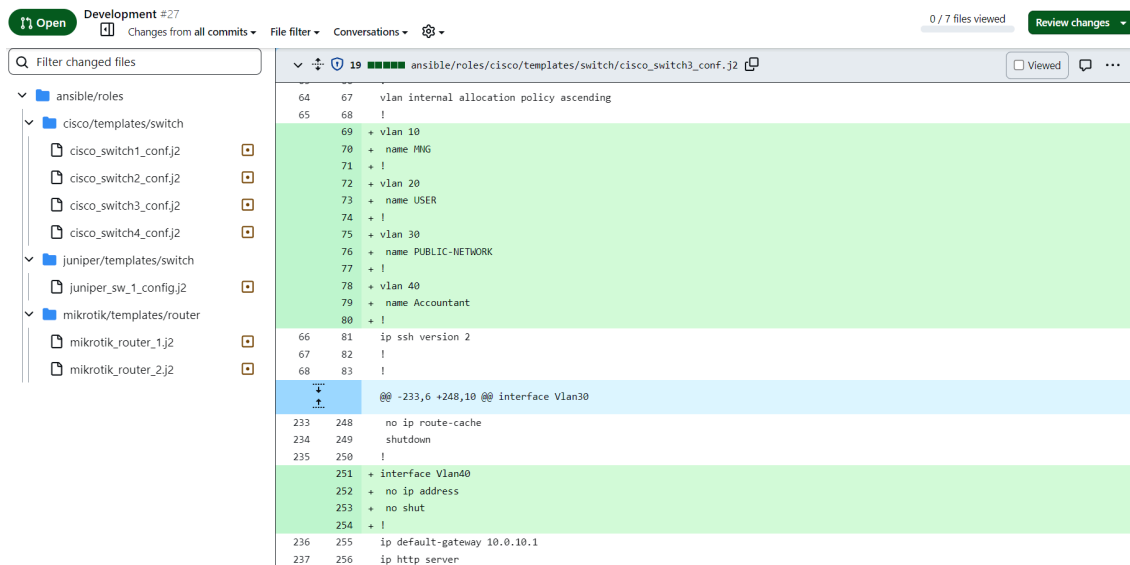


Figure 12. Git commit summary

6.3 Step 2: Continuous Integration Process

Once the changes are pushed to the remote repository, GitHub Actions trigger the automated CI processes. It includes:

Validation of syntax and configuration: Automated processes validate the syntax of the Ansible playbook and configuration files for Cisco and Mikrotik devices. For a visual representation of these syntax and lint checks, refer to Figure 13: GitHub Actions CI Workflow for Syntax and Lint checks.

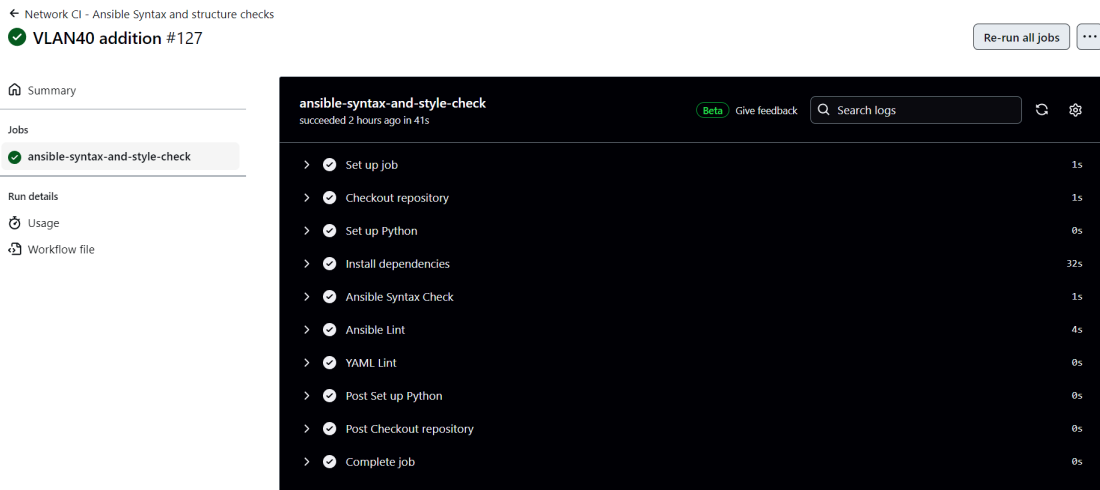


Figure 13. GitHub Actions CI Workflow for Syntax and Lint checks

Automated Testing with Batfish: The network configurations are tested against a virtual network model to ensure they don't introduce any connectivity or security issues. Figure 14: GitHub Actions CI Workflow for Multi-Vendor Configuration Validation provides a detailed view of this testing process.

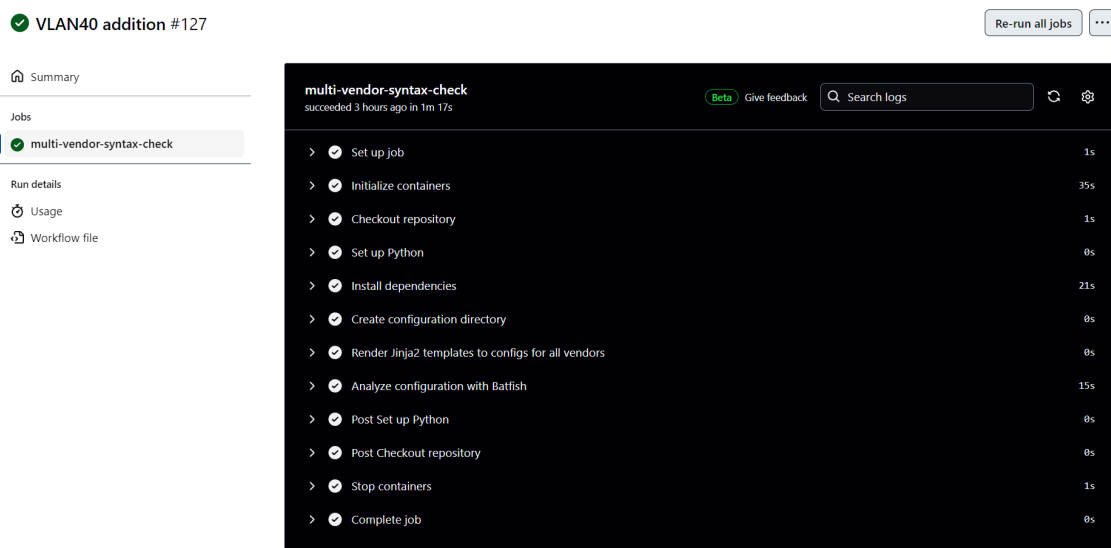


Figure 14. GitHub Actions CI Workflow for Multi-Vendor Configuration Validation

And the following Figure 15 captures the results of the CI checks performed on the multi-vendor network configuration updates, verifying successful validation and readiness for deployment.

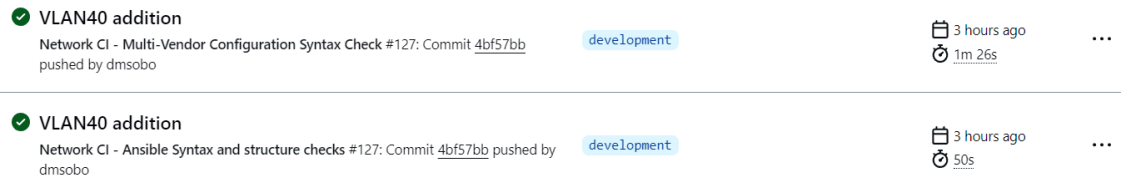


Figure 15. Results of CI workflows

As part of the continuous integration process, the NetOps engineer's push to the development branch initiates a series of events. GitHub webhooks are HTTP callbacks that react to events in the GitHub repository. When the engineer commits changes to a branch, a webhook sends a POST request to a pre-configured URL, which in this scenario is the Jenkins server.

However, since Jenkins is very commonly secured within a private network, ngrok does serve as a secure tunnel to the Jenkins server. It exposes local servers behind NATs and firewalls to the public internet over secure tunnels. It would actually reach Jenkins through ngrok on a webhook without directly exposing it to the public internet.

200 OK

The response was decoded for display. [Download original response](#)

Summary Headers Raw Binary

195 bytes application/json;charset=utf-8, gzip encoded (245 bytes decoded)

```
{
  "jobs": {
    "network-automation": {
      "regexFilterExpression": "^refs/heads/master$",
      "triggered": false,
      "resolvedVariables": {
        "branchRef": "refs/heads/development"
      },
      "regexFilterText": "refs/heads/development",
      "id": 0,
      "url": ""
    }
  },
  "message": "Triggered jobs."
}
```

Figure 16. Webhook Response for Development Branch Commit

The Figure 16 illustrates the webhook response from GitHub after the engineer's commit to the development branch. It clearly shows that the job named **"network-automation"** is not triggered (**triggered: false**). The reason is that it's configured to be triggered only if the commit is made to the master branch. This is a security measure to prevent unauthorised changes from being deployed without proper review and testing.

6.4 Step 3: Continuous Deployment

After passing CI checks, the changes are reviewed and merged into the master branch. At this point the webhook interaction becomes crucial once again. The merge triggers another POST request from GitHub to Jenkins via ngrok. This time, since the commit is on the master branch, Jenkins recognizes the event as a deployment trigger.

200 OK

The response was decoded for display. [Download original response](#)

Summary Headers Raw Binary

197 bytes application/json;charset=utf-8, gzip encoded (249 bytes decoded)

```
{
  "jobs": {
    "network-automation": {
      "regexFilterExpression": "^refs/heads/master$",
      "triggered": true,
      "resolvedVariables": {
        "branchRef": "refs/heads/master"
      },
      "regexFilterText": "refs/heads/master",
      "id": 18,
      "url": "queue/item/18/"
    }
  },
  "message": "Triggered jobs."
}
```

Figure 17. Jenkins Job Triggered by Master Branch Commit

Figure 17 here shows the webhook's payload, specifically indicating a **true** value for **"triggered"**, confirming that Jenkins has acknowledged the event and has queued the job to roll out the changes to the production environment.

Subsequent to the webhook reception, Jenkins executes a pre-configured job. The first step of this job is to synchronise the Jenkins workspace with the master branch, ensuring the latest committed changes are ready for deployment. Figure 28 illustrates this step, showing the Jenkins job in progress as it prepares to deploy the new configurations.

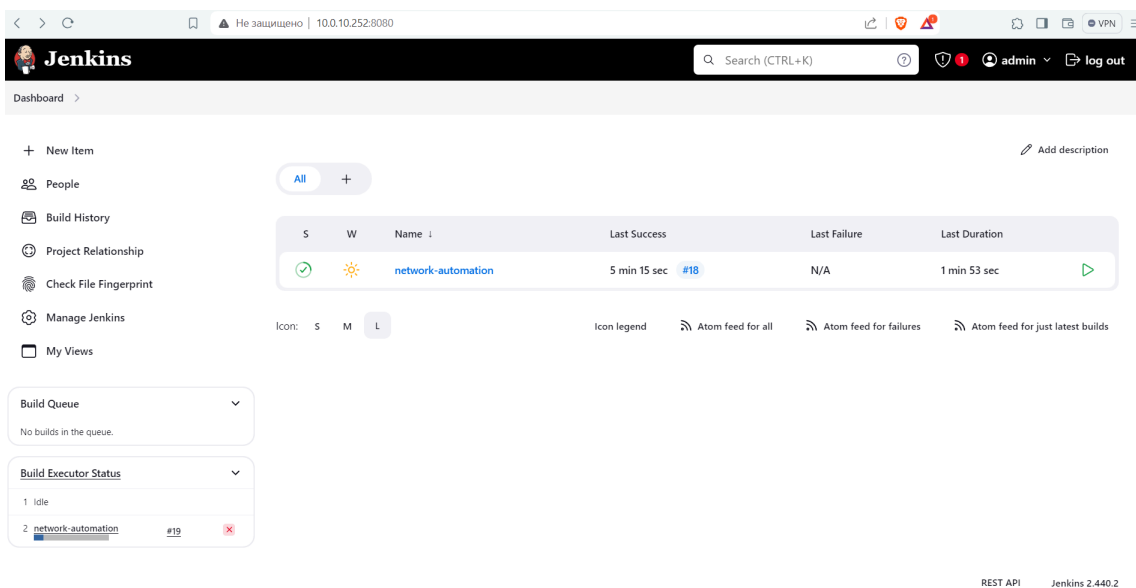


Figure 18. Jenkins job in progress

After the deployment job is initiated, Jenkins executes the Ansible playbook, which is a crucial step in the deployment process. The playbook contains a set of tasks that are run on the network devices to apply the new configurations. Each task in the playbook is designed to be idempotent, meaning that it only makes changes if the desired state does not match the current state. This ensures that the network configuration is always in the expected state after the playbook runs.

```

TASK [mikrotik : Remove temporary configuration scripts] *****
skipping: [cisco_sw_1]
skipping: [cisco_sw_2]
skipping: [cisco_sw_3]
skipping: [cisco_sw_4]
skipping: [juniper_sw_1]
changed: [mikrotik_router_1]
changed: [mikrotik_router_2]

RUNNING HANDLER [cisco : Save running config to startup config (Cisco)] *****
ok: [cisco_sw_1]
ok: [cisco_sw_3]
ok: [cisco_sw_2]

PLAY RECAP *****
cisco_sw_1      : ok=3  changed=2  unreachable=0  failed=0  skipped=15  rescued=0  ignored=0
cisco_sw_2      : ok=3  changed=2  unreachable=0  failed=0  skipped=15  rescued=0  ignored=0
cisco_sw_3      : ok=3  changed=2  unreachable=0  failed=0  skipped=15  rescued=0  ignored=0
cisco_sw_4      : ok=2  changed=1  unreachable=0  failed=0  skipped=15  rescued=0  ignored=0
juniper_sw_1    : ok=4  changed=1  unreachable=0  failed=0  skipped=13  rescued=0  ignored=0
localhost       : ok=26  changed=4  unreachable=0  failed=0  skipped=1   rescued=0  ignored=0
mikrotik_router_1 : ok=8  changed=2  unreachable=0  failed=0  skipped=9   rescued=0  ignored=0
mikrotik_router_2 : ok=8  changed=2  unreachable=0  failed=0  skipped=9   rescued=0  ignored=0

Finished: SUCCESS

```

REST API Jenkins 2.440.2

Figure 19. Jenkins Console Output - Ansible Playbook Execution

Figure 19. displays the results of the Ansible playbook execution. This output is a testament to the power of automation within the CI/CD pipeline. As shown in Figure 19, tasks are executed across various network devices, with a summary indicating the number of changes applied and confirming the successful completion of the job with no failures. This final step ensures that the intended changes are correctly implemented in the production environment.

The job's completion marks the end of the Continuous Deployment phase. At this point, the NetOps team can be confident that the new network configurations are live. Any adjustments made to VLAN settings, firewall rules, or other network parameters are now active, shaping the traffic and security posture as per the committed changes.

6.5 Step 4: Post-Deployment Verification

The post-deployment verification step is critical to ensure the applied changes reflect the expected state of the network. In this case, the verification focuses on confirming that new VLAN configurations have been successfully implemented across the network.

Verification Process:

VLAN Status Check: Review the VLAN configurations to ensure that new VLANs are active and operational as per the deployment plan.

Monitoring System Check: Inspect the monitoring dashboard to verify that it reflects the current operational status of the VLANs, with all newly implemented VLANs showing 'UP' status.

Non-Operational VLANs: Confirm that VLANs that are meant to remain non-operational are indeed not active, ensuring they are correctly excluded from the active network configuration.

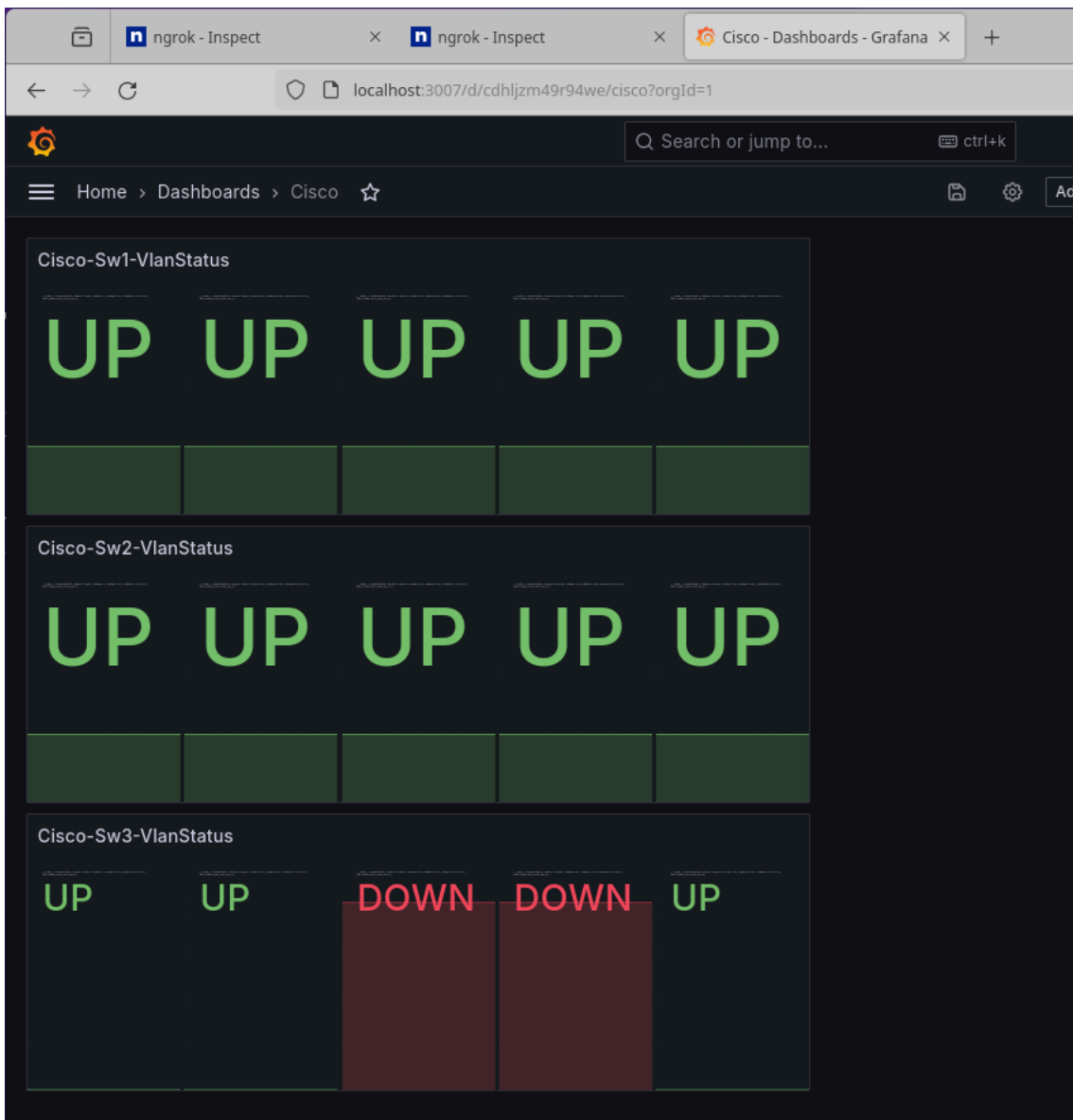


Figure 20. VLAN Implementation Status Post-Deployment

The attached Figure 20 demonstrates the successful addition of the VLANs with their status displayed as 'UP' in the network monitoring dashboard, except for those VLANs that are intentionally set to a non-operational state. This visual evidence from the monitoring tool confirms that the network changes have been properly implemented and are functioning as expected.

7 Results and discussion

This thesis embarked on the development of the automated system of deploying and monitoring changes within the network in a multi-vendor environment of "Company X" in an aim to achieve the least manual intervention in taking into consideration the benefit of operational efficiency and rapid adaptability of the network. In light of the above, the following results are discussed in light of the first research questions and objectives in general, with particular regard to the challenges and limitations.

7.1 Addressing Research Questions

Challenges in Manual Network Deployment and Monitoring Processes: The research further ascertains that manual operations are, to say the least, inefficient in nature and have a high susceptibility to errors. This means they are unable to react in a timely manner to the demand hence made on the network. These, among other factors, contribute highly to operational bottlenecks, hence heightening downtimes and increasing operational cost.

Transformation through Automation: Automation emerged as a pivotal solution, enabling quicker deployments, more accurate monitoring, and a reduction in manual errors. The CI/CD pipeline thus brought in an agile way for better management with dynamic and efficient strategies in network configurations and monitoring which would be aligned with the requirement of agile contemporary network infrastructures.

Constituting an Effective Automated System: An effective automation system was realised by integrating Ansible for automation, alongside Prometheus and Grafana for monitoring, and GitHub Actions with Jenkins for CI/CD workflows. This setup ensured compatibility across multi-vendor devices and scalability to accommodate network growth, establishing a robust framework for ongoing optimization.

7.2 Achievement of Objectives and Observations:

Technological Exploration and Evaluation: Incorporating Jenkins and GitHub Actions significantly enriched the project's CI/CD pipeline. While Jenkins was instrumental in automating the continuous deployment (CD) aspect, ensuring the execution of Ansible playbooks and other tasks upon code commits, GitHub Actions played a pivotal role in the continuous integration (CI) phase. It automated deployment and testing workflows directly within the GitHub repository, simplifying the management of CI pipelines and enhancing source control integration. Together, these tools created a robust automation flow that was critical for the project's success. Ngrok's secure tunnelling capabilities enabled seamless and secure interactions between GitHub Actions and the Jenkins server, mitigating potential security risks. Additionally, the SNMP Exporter and Batfish further supported the project by enhancing network visibility and providing a pre-deployment testing mechanism, respectively, which contributed to the comprehensive monitoring and testing capabilities in a multi-vendor environment.

Design and Implementation Challenges: The thesis successfully designed and implemented a tailored automated deployment and monitoring system. However, simulating a multi-vendor network environment posed significant challenges due to the complexity of accurately representing diverse network devices and the restrictions imposed by commercial licensing. The lack of vendor-provided simulation models or virtual images, coupled with licensing restrictions, made it difficult to create a fully representative virtual staging environment. This limitation underscored the complexity of deploying and monitoring in a real-world multi-vendor context.

Validation of Effectiveness Amidst Limitations: Despite these challenges, the system's effectiveness in reducing manual intervention and operational risks was validated. A noteworthy limitation was the automation of Mikrotik devices, where the lack of comprehensive Ansible modules necessitated reliance on less efficient methods, such as extensive use of the Ansible shell module. This aspect highlighted the need for more sophisticated tool support for certain vendors within the automation framework.

7.3 Conclusions and Future Directions

The automation of network operations at Company X has effectively met the thesis's goals, overcoming identified challenges to foster more efficient, reliable, and agile network operations. The solution in place is also scalable and adaptive, therefore guaranteeing that it can be expanded with the growth of the network and implementation in the future.

However, the limitations encountered, particularly the challenge of simulating a comprehensive multi-vendor environment and the specific issues with automating Mikrotik devices, point to areas needing further development and refinement. Future research could focus on overcoming these limitations through the development of more advanced simulation tools or exploring partnerships with vendors to ease licensing restrictions for educational and testing purposes. Additionally, enhancing the automation framework to include broader vendor support and integrating AI and ML for predictive analytics could significantly advance the field of network operations.

In summary, this thesis has provided critical insights into practical solutions for automation of network operations in a multi-vendor environment and lights the paths for organisations targeting modernization of their network operations. The methodologies and systems developed here lay a print foundation for innovative efficiency to be emulated in the management of the network.

References

- [1] Broadcom Academy, "Fujitsu Central Europe Reduces TCO by 75% with Expanded 'Human-Centric' Approach to NetOps," in Broadcom Network Operations Case Studies, Broadcom, 2023. [Online]. Available: <https://academy.broadcom.com/network-operations/fujitsu-netops-case-study>
- [2] Jay Ashok Shah and Dushyant Dubaria, "NetDevOps: A New Era Towards Networking & DevOps," [Online]. Available: <https://ieeexplore.ieee.org/document/8992969>. [Accessed 10 March 2024].
- [3] Kyle Rankin, DevOps Troubleshooting: Linux Server Best Practices, 1st ed., Addison-Wesley Professional, 2013.3
- [4] Cisco DevNet, "Various Cisco Blogs on Cisco Live," [Online]. Available: <https://blogs.cisco.com/tag/cisco-live>. [Accessed 10 March 2024].
- [5] Aladhami, Mahmood Mazin, Ruhani Ab Rahman, Murizah Kassim, and Abd Razak Mahmud, "Performance Analysis on Network Automation Interaction with Network Devices Using Python," in Proceedings of the 2021 IEEE 11th Symposium on Computer Applications & Industrial Electronics (ISCAIE), IEEE, 2021, doi: 10.1109/ISCAIE51753.2021.9431823. [Online]. Available: <https://ieeexplore.ieee.org/document/9431823>. [Accessed 11 March 2024].
- [6] Caroline Chappell, "DevOps for Network Engineers: The Implications for Network Automation," in DevOps for NetEng White Paper, prepared by Heavy Reading on behalf of Cisco, June 2016. [Online]. Available: <https://community.cisco.com/t5/crosswork-automation-hub-knowledge-articles/devops-for-neteng-white-paper/ta-p/3642983>. [Accessed 11 March 2024].
- [7] Juniper Networks, "The 2020 State of Network Automation: The Annual Report on Key Network Automation Trends," 2020. [Online]. Available: <https://www.juniper.net/us/en/forms/2020-state-of-network-automation-report.html>. [Accessed 11 March 2024].

- [8] Red Hat, "Ansible Network Automation," [Online]. Available: <https://www.redhat.com/en/technologies/management/ansible/network-automation>. [Accessed 12 March 2024].
- [9] Ansible by Red Hat, "Network MOPs as Automated Workflows," [Online]. Available: <https://www.ansible.com/blog/network-mops-as-automated-workflows>. [Accessed 12 March 2024].
- [10] Red Hat, "Red Hat Advances Enterprise and Network Automation with New Ansible Offerings," [Online]. Available: <https://www.ansible.com/press-center/press-releases/red-hat-advances-enterprise-and-network-automation-new-ansible-offerings>. [Accessed 12 March 2024].
- [11] Broadcom Academy, "Network Automation: Top Use Cases and Benefits," [Online]. Available: <https://academy.broadcom.com/blog/network-operations/network-automation-top-use-cases-and-benefits>. [Accessed 13 March 2024].
- [12] TechTarget, "NetSecOps Best Practices for Network Engineers," 07 Feb 2024. [Online]. Available: <https://www.techtarget.com/searchnetworking/tip/NetSecOps-best-practices-for-network-engineers>. [Accessed 13 March 2024].
- [13] Deanna Darah, "Challenges & Best Practices for Network Operations Management," in SearchNetworking, TechTarget, 21 June 2022. [Online]. Available: <https://www.techtarget.com/searchnetworking/feature/Challenges-best-practices-for-network-operations-management>. [Accessed 13 March 2024].
- [14] Grafana Labs, "Intro to Prometheus," in Grafana Documentation, Grafana Labs. [Online]. Available: <https://grafana.com/docs/grafana/latest/fundamentals/intro-to-prometheus/>. [Accessed 22 March 2024].
- [15] GitHub, "GitHub Actions," [Online]. Available: <https://github.com/features/actions>. [Accessed 17 April 2024].

[16] Jenkins, "Jenkins," [Online]. Available: <https://www.jenkins.io/>. [Accessed 17 April 2024].

Appendix 1 - Configuration Details for the Prometheus Role

- name: Install Prometheus
 ansible.builtin.apt:
 name: prometheus
 state: present
 update_cache: true

- name: Start and enable Prometheus
 ansible.builtin.service:
 name: prometheus
 state: started
 enabled: true

- name: Configure Prometheus
 ansible.builtin.template:
 src: prometheus.yml.j2
 dest: /etc/prometheus/prometheus.yml
 mode: '0644'
 notify:
 - Restart Prometheus

- name: Adjust Prometheus default configuration
 ansible.builtin.template:
 src: prometheus_defaults.j2
 dest: /etc/default/prometheus
 mode: '0644'
 notify:
 - Restart Prometheus

Figure 21. Prometheus Role Ansible Tasks

Appendix 2 - Detailed Ansible Tasks for Grafana Setup

- name: Import Grafana GPG key
ansible.builtin.apt_key:
 url: "https://packages.grafana.com/gpg.key"
 state: present

- name: Add Grafana repository
ansible.builtin.apt_repository:
 repo: "deb https://packages.grafana.com/oss/deb stable main"
 state: present

- name: Install Grafana
ansible.builtin.apt:
 name: grafana
 state: present
 update_cache: true

- name: Deploy custom grafana.ini configuration
ansible.builtin.template:
 src: grafana.ini.j2
 dest: /etc/grafana/grafana.ini
 mode: '0644'
notify: Restart Grafana

- name: Setup Default datasource
ansible.builtin.template:
 src: sample.yaml.j2
 dest: /etc/grafana/provisioning/datasources/sample.yaml
notify: Restart Grafana

- name: Copy conf for dashboards
ansible.builtin.template:
 src: dashboard.yaml.j2
 dest: /etc/grafana/provisioning/dashboards/sample.yaml

- name: Setup dashboard Mikrotik
ansible.builtin.copy:
 src: mikrotik.json
 dest: /etc/grafana/provisioning/dashboards/mikrotik.json
notify: Restart Grafana

- name: Setup dashboard Juniper
ansible.builtin.copy:
 src: juniper.json
 dest: /etc/grafana/provisioning/dashboards/juniper.json
notify: Restart Grafana

- name: Setup dashboard Cisco
ansible.builtin.copy:
 src: cisco.json
 dest: /etc/grafana/provisioning/dashboards/cisco.json
notify: Restart Grafana

```
- name: Start and enable Grafana service
  ansible.builtin.systemd:
    name: grafana-server
    state: started
    enabled: true
```

Figure 22. Grafana Role Ansible Tasks

Appendix 3 - Detailed Ansible Tasks for SNMP Exporter

Setup

```
---
- name: Download SNMP Exporter binary
  ansible.builtin.get_url:
  url:
  "https://github.com/prometheus/snmp_exporter/releases/download/v0.25.0
  /snmp_exporter-0.25.0.linux-amd64.tar.gz"
  dest: "/tmp/snmp_exporter.tar.gz"
  mode: '0644'

- name: Extract SNMP Exporter binary
  ansible.builtin.unarchive:
  src: "/tmp/snmp_exporter.tar.gz"
  dest: "/opt"
  remote_src: true
  creates: "/opt/snmp_exporter-0.25.0.linux-amd64/snmp_exporter"

- name: Ensure SNMP Exporter binary is executable
  ansible.builtin.file:
  path: "/opt/snmp_exporter-0.25.0.linux-amd64/snmp_exporter"
  mode: '0755'

- name: Copy SNMP Exporter configuration
  ansible.builtin.template:
  src: "{{ role_path }}/templates/snmp.yml.j2"
  dest: "/opt/snmp_exporter-0.25.0.linux-amd64/snmp.yml"
  mode: '0644'

- name: Create SNMP Exporter systemd service file
  ansible.builtin.template:
  src: "{{ role_path }}/templates/snmp_exporter.service.j2"
  dest: "/etc/systemd/system/snmp_exporter.service"
  mode: '0644'

- name: Reload systemd daemon
  ansible.builtin.systemd:
  daemon_reload: true

- name: Enable and start SNMP Exporter service
  ansible.builtin.service:
  name: snmp_exporter
  state: started
  enabled: true

- name: Set permissions for SNMP Exporter directory
  ansible.builtin.file:
  path: "/opt/snmp_exporter-0.25.0.linux-amd64"
  state: directory
  mode: '0755'
  recurse: true
```



```
when: ansible_facts['distribution'] == 'Ubuntu'

- name: Remove downloaded SNMP Exporter tar.gz file
  ansible.builtin.file:
    path: "/tmp/snmp_exporter.tar.gz"
    state: absent
```

Figure 23. SNMP-Exporter Role Ansible Tasks

Appendix 4 - Detailed Ansible Tasks for Cisco Role

- name: Backup current configuration
cisco.ios.ios_config:
 backup: true
when: vendor == 'cisco'
register: backup_config

- name: Apply complete configuration to Cisco routers
cisco.ios.ios_config:
 src: "templates/router/{{ config_id }}.j2"
 save_when: modified
when: vendor == 'cisco' and device_type == 'router'
notify: Save running config to startup config (Cisco)

- name: Apply complete configuration to Cisco switches
cisco.ios.ios_config:
 src: "templates/switch/{{ config_id }}.j2"
 save_when: modified
when: vendor == 'cisco' and device_type == 'switch'
notify: Save running config to startup config (Cisco)

Figure 24. Cisco Role Ansible Task

Appendix 5 - Detailed Ansible Tasks for Juniper Role

- name: Check if NETCONF is enabled on Juniper devices
junipernetworks.junos.junos_command:
 commands:
 - show configuration system services | match netconf
 register: netconf_check
 when: vendor == 'juniper'

- name: Enable NETCONF if not already enabled (CLI)
junipernetworks.junos.junos_netconf:
 netconf: true
 when:
 - netconf_check is defined
 - netconf_check.stdout_lines is defined
 - netconf_check.stdout_lines | join('') | regex_search('netconf')
is none
 - vendor == 'juniper'

- name: Set connection method to NETCONF for Juniper devices
ansible.builtin.set_fact:
 ansible_connection: "ansible.netcommon.netconf"
 ansible_port: "{{ netconf_port }}"
 ansible_become: false
 when: vendor == 'juniper'

- name: Backup current configuration on Juniper devices (NETCONF)
junipernetworks.junos.junos_config:
 backup: true
 when: vendor == 'juniper'

- name: Apply configuration to Juniper routers (NETCONF)
junipernetworks.junos.junos_config:
 src: "templates/router/{{ config_id }}.j2"
 when: vendor == 'juniper' and device_type == 'router'
 notify: Save configuration on Juniper devices

- name: Apply configuration to Juniper switches (NETCONF)
junipernetworks.junos.junos_config:
 src: "templates/switch/{{ config_id }}.j2"
 when: vendor == 'juniper' and device_type == 'switch'
 notify: Save configuration on Juniper devices

Figure 25. Juniper Role Ansible Task

Appendix 6 - Detailed Ansible tasks for Mikrotik Role

- name: Backup current configuration on MikroTik device
community.routeros.command:
 commands:
 - "/system backup save name={{ inventory_hostname }}_backup"
when: vendor == 'mikrotik'
register: backup_config

- name: Export configuration to an .rsc file
community.routeros.command:
 commands:
 - "/export file={{ inventory_hostname }}_backup"
when: vendor == 'mikrotik'
register: export_config

- name: Manually copy the binary backup file from MikroTik device
ansible.builtin.shell: >
 sshpass -p '{{ ansible_ssh_pass }}' scp -o
StrictHostKeyChecking=no
 ssh@{{ ansible_host }}:{{ inventory_hostname }}_backup.backup
 {{ playbook_dir }}/roles/mikrotik/backup/{{ inventory_hostname
}}_binary_backup.backup
 changed_when: false
 become: false
 delegate_to: localhost
 when: vendor == 'mikrotik'

- name: Manually copy the configuration export file from MikroTik device
ansible.builtin.shell: >
 sshpass -p '{{ ansible_ssh_pass }}' scp -o
StrictHostKeyChecking=no
 ssh@{{ ansible_host }}:{{ inventory_hostname }}_backup.rsc
 {{ playbook_dir }}/roles/mikrotik/backup/{{ inventory_hostname
}}_config_backup.rsc
 changed_when: false
 become: false
 delegate_to: localhost
 when: vendor == 'mikrotik'

- name: Generate MikroTik configuration script from template
ansible.builtin.template:
 src: "router/{{ config_id }}.j2"
 dest: "{{ playbook_dir }}/roles/mikrotik/temp/{{
inventory_hostname }}_config.rsc"
 mode: '0644'
 delegate_to: localhost
 when: vendor == 'mikrotik'

- name: Upload configuration script to MikroTik device via SCP
ansible.builtin.shell: >

```

    sshpass -p '{{ ansible_ssh_pass }}' scp -o
StrictHostKeyChecking=no
    {{ playbook_dir }}/roles/mikrotik/temp/{{ inventory_hostname
}}_config.rsc
    ssh@{{ ansible_host }}:/
    changed_when: false
    become: false
    delegate_to: localhost
    when: vendor == 'mikrotik'

- name: Apply configuration script on MikroTik device
  community.routeros.command:
    commands:
      - "/import file-name={{ inventory_hostname }}_config.rsc"
    when: vendor == 'mikrotik'

- name: Remove temporary configuration scripts
  ansible.builtin.file:
    path: "{{ playbook_dir }}/roles/mikrotik/temp/{{
inventory_hostname }}_config.rsc"
    state: absent
    delegate_to: localhost
    when: vendor == 'mikrotik'

```

Figure 26. Mikrotik Role Ansible Task

Appendix 7 - Detailed GitHub CI Workflows



```
1 name: Network CI - Ansible Syntax and structure checks
2
3 on:
4   push:
5     branches-ignore:
6       - master
7   pull_request:
8     branches:
9       - '*'
10
11 jobs:
12   ansible-syntax-and-style-check:
13     runs-on: ubuntu-latest
14     steps:
15       - name: Checkout repository
16         uses: actions/checkout@v4
17
18       - name: Set up Python
19         uses: actions/setup-python@v5
20         with:
21           python-version: '3.x'
22
23       - name: Install dependencies
24         run: |
25           pip install ansible ansible-lint yamllint
26
27       - name: Ansible Syntax Check
28         run: ansible-playbook --syntax-check ./ansible/*.yaml
29
30       - name: Ansible Lint
31         run: ansible-lint ./ansible/*.yaml
32
33       - name: YAML Lint
34         run: 'yamllint ./ansible/ -d \"{extends: default, rules: {line-length: {max: 120}}}\"'
35
36
```

Figure 27. Workflow for Ansible Syntax and structure checks

```
Open [icon] network-ci-conf-files-syntax.yaml
~/network-automation/github/workflows

1 name: Network CI - Multi-Vendor Configuration Files Check
2
3 on:
4   push:
5     branches-ignore:
6       - master
7   pull_request:
8     branches:
9       - '*'
10
11 jobs:
12   multi-vendor-syntax-check:
13     runs-on: ubuntu-latest
14     services:
15       batfish:
16         image: batfish/allinone
17         ports:
18           - 9996:9996
19     steps:
20     - name: Checkout repository
21       uses: actions/checkout@v4
22
23     - name: Set up Python
24       uses: actions/setup-python@v5
25       with:
26         python-version: '3.x'
27
28     - name: Install dependencies
29       run: |
30         pip install setuptools
31         pip install pybatfish jinja2
32
33     - name: Create configuration directory
34       run: mkdir -p configs
35
36     - name: Render Jinja2 templates to configs for all vendors
37       run: python scripts/render_jinja_templates.py
38       working-directory: ${github.workspace}
39
40     - name: Analyze configuration with Batfish
41       run: python scripts/analyze_with_batfish.py
42
```

Figure 28. Workflow for Batfish Network Configuration Analysis

Appendix 8 - Detailed Configuration of Dynamic Prometheus

```
global:
  scrape_interval: 60s

scrape_configs:
{% for host in groups['cisco'] %}
  - job_name: 'snmp_cisco_{{ host }}'
    scrape_interval: 60s
    metrics_path: /snmp
    params:
      module: [if_mib]
    static_configs:
      - targets:
        - "{{ hostvars[host].ansible_host }}"
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: {{ snmp_exporter_server }}:9116 # SNMP
Exporter's address
{% endfor %}

{% for host in groups['juniper'] %}
  - job_name: 'snmp_juniper_JunMib_{{ host }}'
    scrape_interval: 60s
    metrics_path: /snmp
    params:
      module: [juniper_operating]
    static_configs:
      - targets:
        - "{{ hostvars[host].ansible_host }}"
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: {{ snmp_exporter_server }}:9116 # SNMP
Exporter's address
{% endfor %}

{% for host in groups['mikrotik'] %}
  - job_name: 'snmp_mikrotik_{{ host }}'
    scrape_interval: 60s
    metrics_path: /snmp
    params:
      module: [mikrotik]
    static_configs:
      - targets:
```



```

    - "{{ hostvars[host].ansible_host }}"
relabel_configs:
  - source_labels: [__address__]
    target_label: __param_target
  - source_labels: [__param_target]
    target_label: instance
  - target_label: __address__
    replacement: {{ snmp_exporter_server }}:9116 # SNMP
Exporter's address
{% endfor %}

{% for host in groups['mikrotik'] %}
  - job_name: 'snmp_mikrotik_if_mib_{{ host }}'
    scrape_interval: 60s
    metrics_path: /snmp
    params:
      module: [if_mib]
    static_configs:
      - targets:
        - "{{ hostvars[host].ansible_host }}"
    relabel_configs:
      - source_labels: [__address__]
        target_label: __param_target
      - source_labels: [__param_target]
        target_label: instance
      - target_label: __address__
        replacement: {{ snmp_exporter_server }}:9116 # SNMP
    Exporter's address
{% endfor %}

```

Figure 29. Dynamic Prometheus Configuration

Appendix 9 - Detailed Grafana Data Source Configuration

```
apiVersion: 1

deleteDatasources:
  - name: Prometheus
    orgId: 1

datasources:
- name: Prometheus
  type: prometheus
  access: proxy
  orgId: 1
  url: http://{{ prometheus_server }}:{{ prometheus_port }}
  password:
  user:
  database:
  basicAuth: false
  basicAuthUser:
  basicAuthPassword:
  withCredentials:
  isDefault: true

version: 1
editable: true
```

Figure 30. Dynamic Grafana Data Source Configuration

Appendix 10 – Non-exclusive licence for reproduction and publication of a graduation thesis¹

I Leonid Peskov

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Streamlining Network Operations: Implementing Automated Deployment and Monitoring in a Multi-Vendor Environment”, supervised by Mohammad Tariq Meeran
 - 1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
 - 1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

09.05.2024

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.