VILNIUS UNIVERSITY
FACULTY OF MATHEMATICS AND INFORMATICS
INSTITUTE OF COMPUTER SCIENCE
DEPARTMENT OF COMPUTATIONAL AND DATA MODELING

Information Technologies 4th year Final Bachelor Thesis

# Automation of Computer Networks

**Kompiuterinių tinklų automatizavimas**

Done by:
Aras Urbonas

Supervisor:
Lekt. Eduardas Kutka

Vilnius
2024

# Contents

# Abstract

Currently, computer networks are expanding, mainly due to the surging demand for online services. This makes configuration of networks by traditional means increasingly challenging. Better solutions for network management are needed. In such an environment, network automation seems enticing, promising to reduce the amount of human hours required, as well as reducing chances for errors. Thus, the purpose of this work was to analyze the possible ways to automate common network management tasks. This includes the analysis of the methodologies, tools and technologies that have the potential to automate computer networks, as well as providing an example of an implementation using freely available open-source, multi-platform tools to automate a situation, hardly feasible for manual configuration: firewall rule creation for virtual resources.

# Santrauka

## Kompiuterių tinklų automatizavimas

Šiuolaikiniai kompiuterių tinklai sparčiai plečiasi. Vienas iš pagrindinių veiksnių lemiančių šią plėtrą, tai auganti paklausa įvairioms nuotolinėms paslaugoms. Dėl šios plėtros, kompiuterinių tinklų konfigūravimas tradiciniais metodais tampa vis labiau nepraktiškas. Tyrimai rodo, jog didžioji dalis kompiuterinių tinklų valdymo procedūrų vis dar vykdomos rankiniais metodais. Nauji, modernūs sprendimai yra reikalingi tam, kad patobulinti kompiuterinių tinklų patikimumą, saugumą bei sumažinti laiko kiekį, kurį sistemų administratoriai praleidžia pastoviems, pasikartojantiems konfigūravimo darbams. Taigi, šio darbo tikslas yra analizuoti galimus kompiuterinių tinklų automatizavimo būdus, bei apžvelgti technologijas bei metodus, skirtus šiai užduočiai įgyvendinti. Be to, šiame darbe taip pat pateikiamas pavyzdys, naudojantis atviro kodo automatizacijos bei virtualizacijos įrankius tam, kad pademonstruoti kompiuterinių tinklų automatizavimo galimybes naudojant nemokamai ir lengvai prieinamas technologijas. Užduotis pasirinkta šiam pavyzdžiui: virtualių resursų automatinis diegimas, bei ugniasienės taisyklių priskyrimas jiems.

# Introduction

In the current era of information technologies, computer networks are becoming more and more sophisticated. The demand for cloud services is rising. Many traditionally offline services are moving online. Networks are expanding in scope and complexity, due in large part to the rise of virtual networks and virtualized networking equipment. This presents an issue - configuration of modern, corporate computer networks is time consuming and error prone. This is especially relevant as simple mistakes in network configurations often lead to downtime and thus, loss of business continuity.

One possible solution to alleviate some of these shortcomings is the automation of computer networks. Potential tasks that could be automated include switching and routing configurations, network device configuration backups, inventorization as well as monitoring for errors, among others. Fortunately, most networking devices, particularly ones in the enterprise sphere, are accessible via a secure shell connection, are configurable via command-line and support open management protocols. Thus preexisting open source tools can often be used to automate networking equipment configuration changes. Furthermore, technologies such as software defined networking promise network management centralization via protocols such as OpenFlow. Thus providing the ability to control networks from a single point, allowing to easily manage network devices and apply changes automatically. One of the objectives for this work is to analyze the possible technologies and methodologies using which, networks can be automated.

Firewall and access control list rules are particularly enticing to automate. Their functions are especially crucial, due to their role in network security. Along with that, they can bring down connectivity between networks when improperly configured. A modern network often contains not one, but many firewalls and access control lists. Thus configuring access for a host to a server may require connecting to multiple networking devices and configuring firewall rules on each one. With this manual process being time consuming and human-error prone, the automation of such access rules is particularly relevant. Additionally, virtualization of hosts and network devices allows to automate not only configuration changes, but also to automate deployment. Virtualization platforms may also often have built-in firewalls which integrate well with automation. This enables the possibility to automate the entire process of host deployment, as well as configuration of firewalls and access control lists to give access to the exact network resources required for the hosts. This has the ability to improve network security, while also removing the need for manual human intervention each time a virtual host is created. Thus, a second goal of this work is to provide an example use case and minimal implementation for automation of firewall rules in a virtual environment.

# 1 Analysis

## 1.1 Approaches to Computer Network Management

Many of the networks which form the backbone of numerous organisations have been formed gradually. Over time expanding and changing to meet the changing needs of their parent organizations. Changes most often being configured manually. Multiple contemporary sources state that up to 95% of networking configuration tasks are done manually [12] [18], often via static configurations, often joined by the use of dynamic routing protocols such as OSPF, BGP, among others.

As a network grows in size and complexity it becomes harder to manage manually. Relatively simple tasks, such as adding hosts and defining which resources they are allowed to access becomes a time-consuming task. It can require not only the configuration of layer 2 switch ports, but also connection to numerous devices to configure firewalls and access-control lists. As well as potentially adding or appending static routes.

There are some proposed solutions to these shortcomings, a major one being Software-Defined-Networking (SDN). [1] [11]. It promises automation and simplification of management, though implementing it may require rather major changes to existing network infrastructure.

The use of open-source automation tools for including automation in workflows is also relevant. Appending existing processes, such as firewall and ACL configurations can bring the benefits of reduced human workload and increased scalability and security without the need for to up-root large parts of existing network infrastructure.

## 1.2 Existing Solutions for Network Automation

Utilizing modern technologies such as software-defined network to implement network automation may be beneficial, as the centralized control plane would mean that configuration changes would only need to be done in a single place. This would mean that changes to the way network devices forward data could be applied automatically, without the need to manually connect to numerous devices.

Though, network automation does not necessarily have to involve the most contemporary technologies, languages or protocols. Tools such as shell scripts can be used to achieve a certain level of automation. However, utilizing more modern declarative scripting tools and template engines may often prove to be more efficient in many scenarios. Using a combination of declarative scripts, templates and shell scripts could make it feasible to automate the process of configuring multiple servers and network devices, via a single script or playbook. This approach would not require purchasing any new equipment or software as all the mentioned technologies are open-source and widely compatible with existing network equipment hardware, firmware and software. While this approach would not constitute as true software-defined networking, it would be a good example of how infrastructure-as-code tools can be achieved a certain level of network automation, without the need to overhaul the network topology.

Additionally, in virtualized environments, platforms have tools and features that lend themselves well to automation. For example, in some virtualization platforms, firewall configurations for a particular container or virtual machine are applied immediately when a file, that is synchronized across a cluster is changed. Thus a simple shell script could be used to make changes to firewall rules that apply to one or more virtual machines.

## 1.3 Network Automation using Software-Defined Networking

One of the possible ways to implement computer network automation involves the use of 'Software-Defined Networking' - commonly shortened to SDN. This option of automating computer networks will be explored further in this subsection.

### 1.3.1 SDN Concepts and Principles

One of the main principles of software-defined networking is the separation of the data and control planes. [4] The data plane is responsible for the transmission of packets using lower level protocols. Conversely, the control plane is tasked with managing and controlling the network using higher level protocols for switching and routing, such as MPLS, BGP and OSPF, among others.

In order to better understand the significance of separating the aforementioned planes, it is important to firstly observe the data and control planes in a traditional network structure. In traditional networks, the data and control planes exist on the same physical network device (be it router, switch, firewall, et cetera). Thus, the distinction between the data and control planes in a traditional network is mostly a logical divide, with the planes coexisting and operating on the same hardware, with the respective operating systems and other software of the devices handling the responsibilities of both planes. With this in mind, the diagram below illustrates the location of data and control planes in a traditional hierarchical network, containing routers, layer 3 switches along with layer 2 switches.

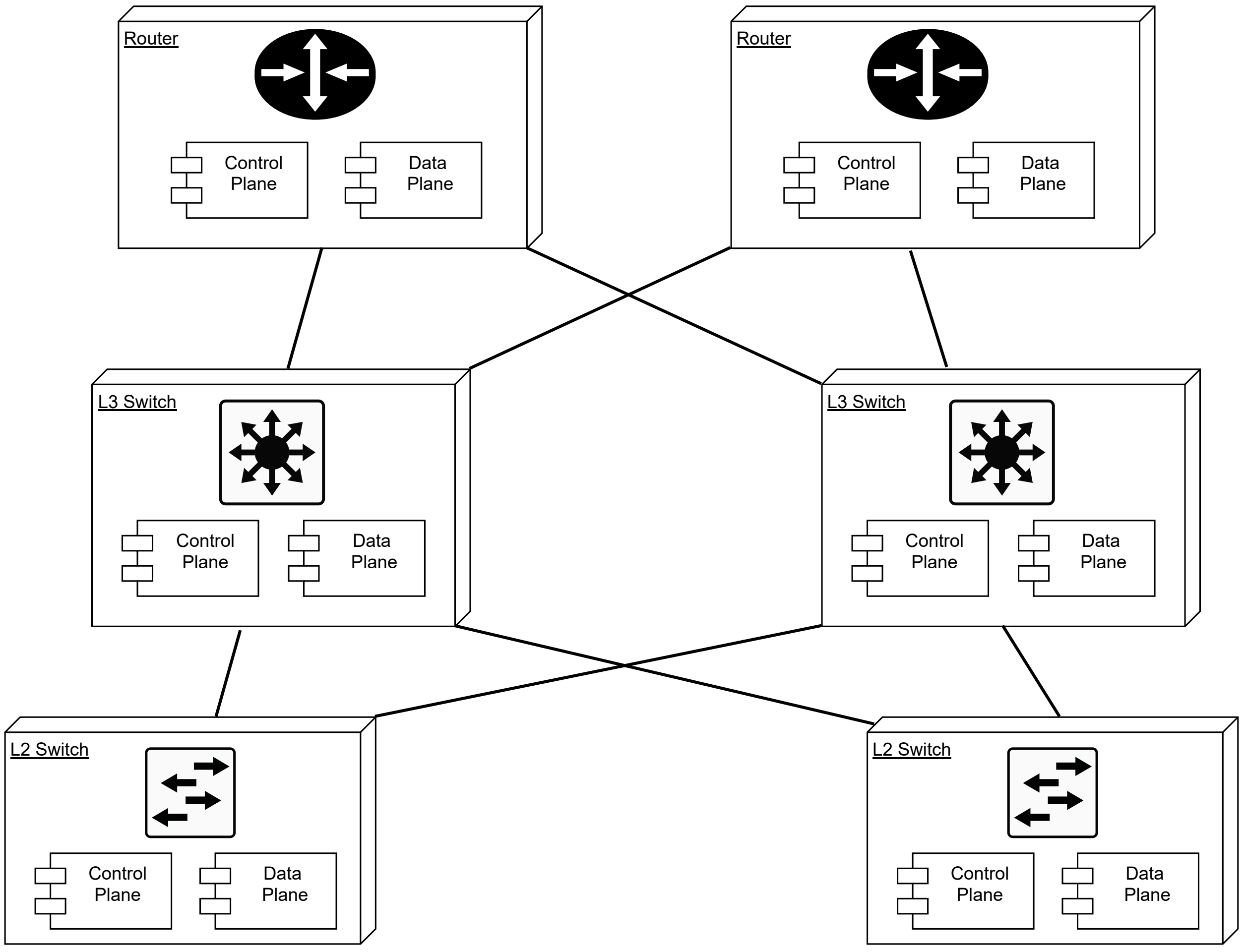

Figure 1. Diagram With The Locations of Data and Control Planes in a Traditional Network

In software-defined networking these planes are separated into different devices. Along with this decoupling, the control plane is to be centralized to a single point, device or entity, thus enabling the centralized control of networking devices. The diagram below illustrates this separation in a software-defined network structure separation via a similar hierarchical network to the one shown in the traditional network diagram.

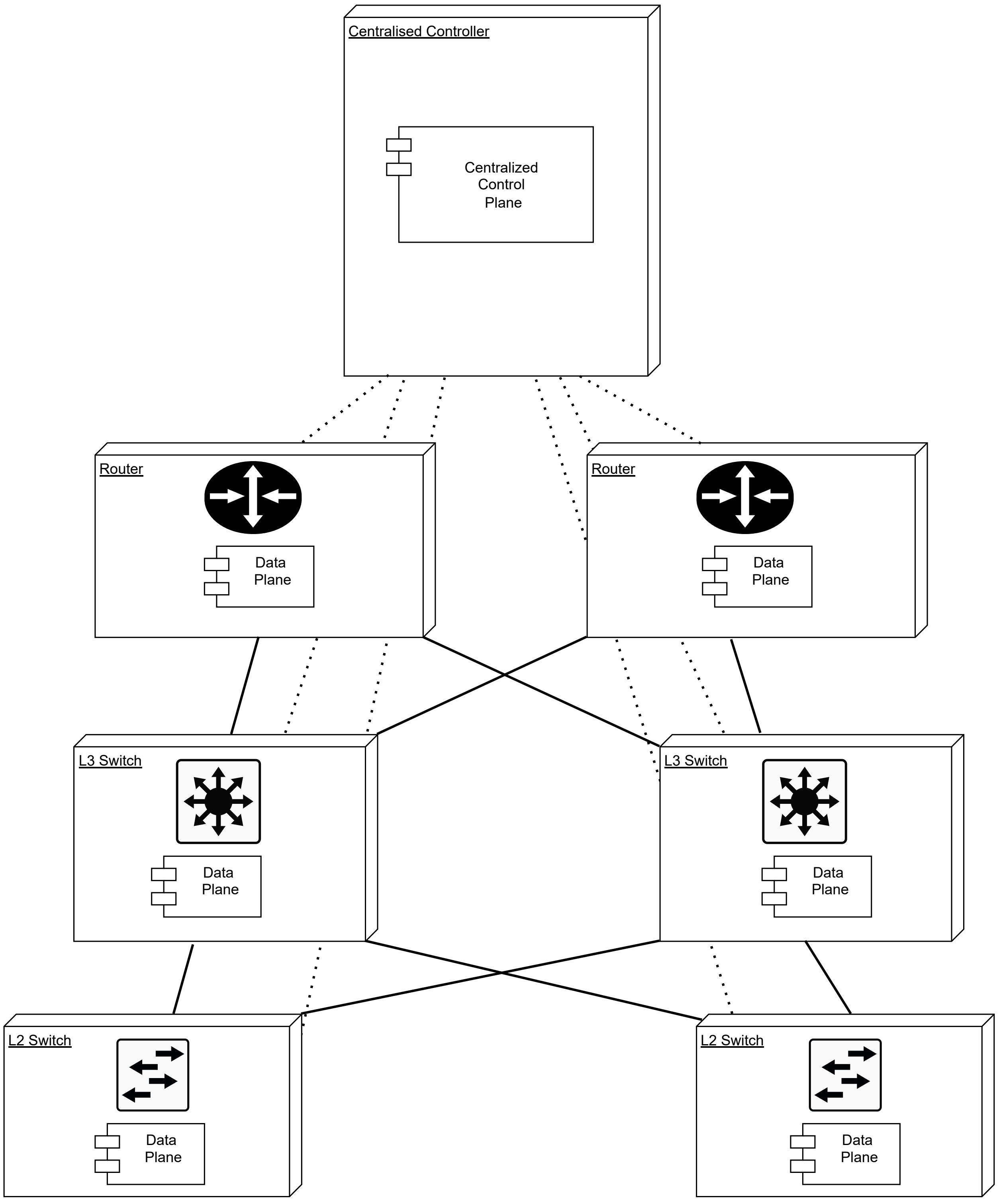

Figure 2. Diagram With The Locations of Data and Control Planes in a Software-Defined Network

Since the control plane is the entity that is tasked with regulating, segmenting and routing network traffic, centralizing it into a single point could make automating network tasks easier, since, changes would only have to be made in one centralized point (the centralized control plane) and would not have to be made across many separate devices. Thus saving time and reducing the chances for errors.

### 1.3.2 SDN Models and Their Advantages

Software-defined networking is a rather broad set of principles, denoting the ways for software based controllers to interface with the hardware. Thus multiple models of SDN are recognized. [2] Including, but not limited to:

- Open SDN - characterized by the use of open-source protocols, such as OpenFlow to control network devices.
- SDN by APIs - application programming interfaces are used to control the data plane of network devices or for communication with higher level applications.
- SDN Overlay Model - this model runs a virtual network on existing hardware, using tunnels to create connections with virtual or physical sites.
- Hybrid SDN - combines a software-defined network with traditional network infrastructure. This allows to introduce SDN gradually, in stages, minimizing disruptions to traffic.

In this subsection, these software-defined networking approaches will be analyzed, comparing each of their advantages. It is important to note, that these models more closely resemble guidelines and principles, and not concrete implementations. Thus there may be a fair amount of overlap between them. As of writing, there does not seem to be a consensus for what constitutes a specific model, as aspects of certain models can be integrated with the implementations of other models. Having acknowledged this, it is important to further analyze the aforementioned models.

The main characteristic of Open SDN is the use of open-source standards, protocols and technologies to implement a software-defined network architecture. The most commonly used protocol to achieve this is OpenFlow. OpenFlow is an open standard maintained by the non-profit Open Networking Foundation. OpenFlow version 1.0 was released in 2009, with limited functionality. With the release of OpenFlow 1.1 in 2011, gaining full VLAN and MPLS support, among other improvements. [3] Currently, in the SDN sphere, OpenFlow is the de facto standard for communication between the centralized control plane and the hardware of network devices (commonly referred to as southbound communication). As OpenFlow is not a proprietary standard, the main advantage is that it can be used and integrated into the various network devices, regardless of the vendor or manufacturer. Additionally, it is important to note that, despite OpenFlow being the commonly accepted standard for southbound communication, protocols such as NETCONF and RESTCONF can also be used as southbound interfaces between the centralized control plane and network devices.

In SDN implementations, APIs are commonly seen alongside OpenFlow, rather than in place of it. APIs in software-defined networks are seen as better suited for northbound communication. That is, interfacing between the centralized control plane and applications via higher level protocols. Common use cases for such northbound APIs is connecting application layer firewalls, load balancers and various security appliances to the SDN controller. Furthermore, southbound protocols such as RESTCONF can enable the communication between the centralized controller and network hardware via REST APIs. Ths use of such northbound APIs enables integration with other automation tools, as well as integration with other services, for example: network monitoring systems.

The Diagram below shows an example where OpenFlow is used for southbound communication and APIs for northbound integration.

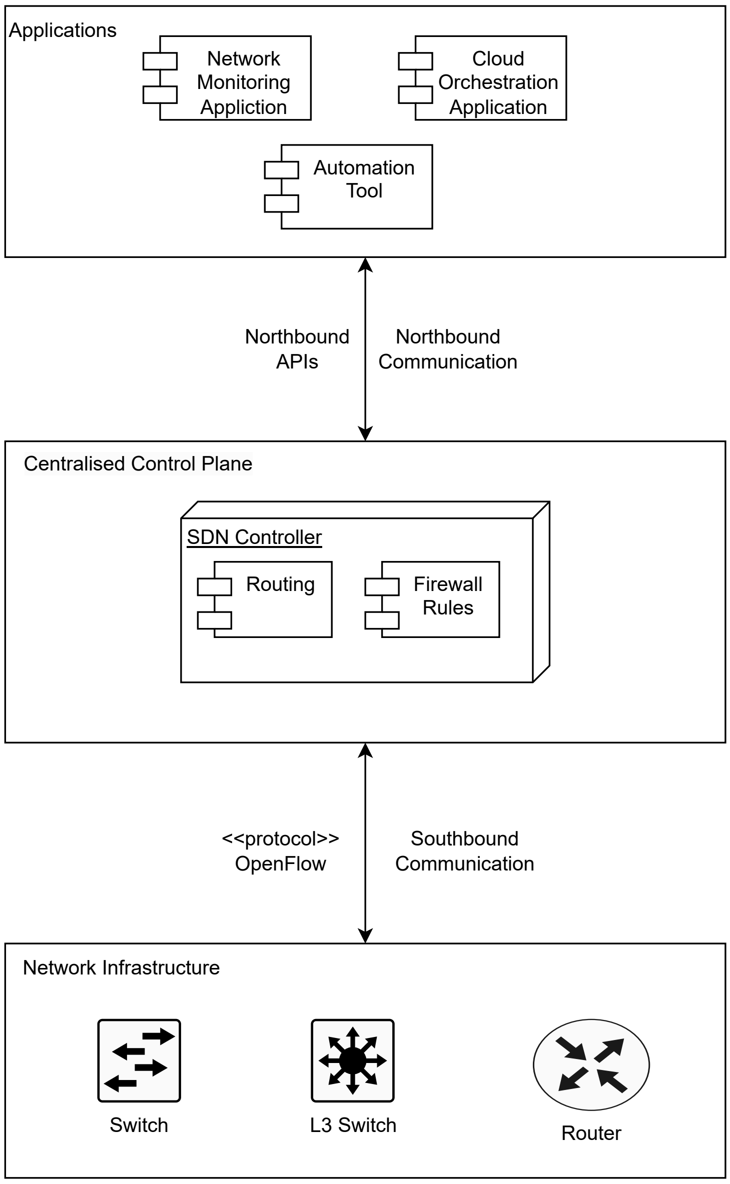

Figure 3. Example of an SDN Implementation with OpenFlow and Northbound APIs

The SDN Overlay Model is characterized by having an additional virtual network over of existing network hardware and infrastructure. The physical network devices remain unchanged, only the data flowing through the network acquires additional tags, denoting their participation in the new overlay network. This can be implemented using the Virtual Tunnel Endpoint (VTEP) technology. Multiple vendors, mainly Juniper and VMware currently offer such a solution for SDN using some of their proprietary tools. The advantages of such an implementation is that existing network hardware remain unaffected, both the devices and their configurations can stay unchanged. This reduces overhead both in terms of hardware cost and labour associated with re-configuration. Additionally, it enables the use of centralized automation, as well as the ability to automate the creation of new virtual networks. The downside being that such an overlay model will likely involve using proprietary software and the remaining infrastructure will still have to be maintained in a manual way, when changes to the physical hardware are required.

A hybrid SDN approach uses a combination of traditional network infrastructure and adds SDN enabled network devices and structure in a gradual way. This approach allows for a more gradual move a software-defined network, while keeping some of the existing legacy hardware running. This means that downtime can be contained to parts of the network during the move to SDN. This can enable a more organic and natural move towards the new network structure, without the need to uproot all existing network infrastructure all at once.

Thus to conclude this part, it is apparent that different approaches and models of software-defined networking have their own unique strengths and advantages. It as also evident that the models often have overlap between themselves, which can, at times, make it difficult to distinguish which particular model would be best to use. Fortunately, many of the models can be freely combined and used alongside each other, as open protocols such as OpenFlow allow their use across different hardware and software vendors, in different scenarios.

### 1.3.3 SDN Challenges and Drawbacks

Having gone through the analysis of different models of software-defined networks and their advantages, it is important to consider the potential challenges and drawbacks of implementing an SDN network structure.

One potential challenge in implementing SDN is obtaining network hardware that support the required software-defined networking protocols, such as OpenFlow. Even tough OpenFlow is an open standard, for which specifications and technical data is available for all vendors of network equipment, not all have seemed to fully implement support for said protocol. For example, the networking giant Arista Networks lists OpenFlow capability in the data sheet for their 7050SX Series of data center switches. [14] However, the latest supported on said switches version is listed as OpenFlow 1.3, which lacks some more advanced features from newer versions. Additionally, other models of switches from certain other vendors may only support OpenFlow 1.0, which is fairly limited, or not have any OpenFlow support whatsoever. Thus requiring the replacement of such hardware in order to implement an Open SDN network model. An organization which seeks to implement SDN, will incur significant costs to procure and replace existing network hardware. As well as having to consider the human hours required for the configuration of the new network structure.

Performance is another consideration of the move to a software-defined network. Some models of SDN, such as the overlay model have an obvious downside, as they rely on additional virtualization as well as tagging, which runs over the existing network infrastructure, introducing performance overhead. Other models may also negatively impact performance, since many network devices have traditionally highly integrated hardware, which had the control and data planes combined in one. The separation of the aforementioned planes may thus negatively impact performance, as vendor may not yet have perfected this concept.

An additional challenge to consider when implementing SDN is reliability of the network. All models of software-defined networking involve some kind of centralization of the control plane. This introduces a single point of failure. Traditional networks normally seek to avoid such cases by segmenting the network and using redundant devices to eliminate or at least reduce the impact of the failure of a single device. In a software-defined network, the loss of the centralized control plane would have far reaching consequences for IT resource availability and by extension business continuity. This makes the SDN controller a particularly vulnerable part of the network. Furthermore, this vulnerability extends to network security. As a breach of the central controller would mean an attacker would likely have vast and far reaching control over the network. These challenges would have to be carefully considered and rectified before the move to a new network structure.

Having analyzed some potential challenges and drawbacks of software-defined networks it is clear that the move to such a structure will require significant resources, as well as careful considerations in terms of network security and reliability.

### 1.3.4 Feasibility of Implementing SDN in a Corporate Network

Having considered the advantages and challenges of SDN, is it now possible to discuss the feasibility of implementing a software-defined network in a organizations IT infrastructure.

In terms of costs, there may be major variations depending on the size and complexity of the network in question. In the case of a medium sized network, where the majority of network equipment is relatively modern and from a well supported vendor, the costs may not be major, as there may be little to no need to replace equipment, in case that it supports Open SDN protocols such as OpenFlow. In a medium sized network, that is well designed, uses a hierarchical network structure, implementing SDN may not require many human hours to reconfigure the equipment either. Of course, a proof-of-concept environment for testing would still be required, as networking is especially crucial to the business continuity of an organization. Thus, any changes would need to be well planned, and many different scenarios and various contingency plans would need to be put in place before any making any changes to the existing network.

Conversely, if a network is of a larger scale, is made up of large amounts of legacy equipment or has a more chaotic structure implementing software-defined networking will require significant resources. In such a case, replacing the legacy network hardware will be necessary. Reconfiguration of the network may also take a lot longer, especially in the case of redundant loops, which had been left intentionally or by human error. Other considerations, such as spanning-tree protocol mismatches may also prove the network to be more fragile when attempting changes, which may make some SDN models, such as the hybrid model more difficult to implement.

Having considered these factors, it is important to observe examples of a successful move to a software-defined network in certain organizations. The notable one being the multinational technology giant Google. The company adopted software-defined networking rather early, as they had already presented details about their SDN network in the 2013 Open Networking Summit. They described using an Quagga open-source software alongside OpenFlow. In terms of hardware, Google was not satisfied with any of the network hardware available at the time. This led the giant to design their own switches using existing silicon to satisfy their needs. [15] A significant amount of time has passed since then, though, as was seen in the example of Arista 7050SX series datacenter network switches, currently hardware still may not support the latest protocols and features.

This example has shown that a move to a software-defined network is possible, even for a very large organization such as Google. Though, as was seen regarding the lack of existing hardware, among other challenges shows that implementing SDN may not be optimal for most organizations. Thus, exists a need for implementations of network automation, which do not require such major changes to network infrastructure. Even after more than a decade from the release of the first OpenFlow versions, many organizations nevertheless use a traditional network structure, where the data and control planes exist on the same hardware devices.

## 1.4 Network Automation using Open Source IT Automation Tools

After having analyzed software-defined networking, it is evident that it not a solution that is suitable for all organizations. This may be due to a variety of factors, which had been discussed in the previous subsections. Thus, other kinds of solutions are necessary to implement computer network automation.

The UML use case diagram below pictures some potential use cases for open-source automation tools for network automation.

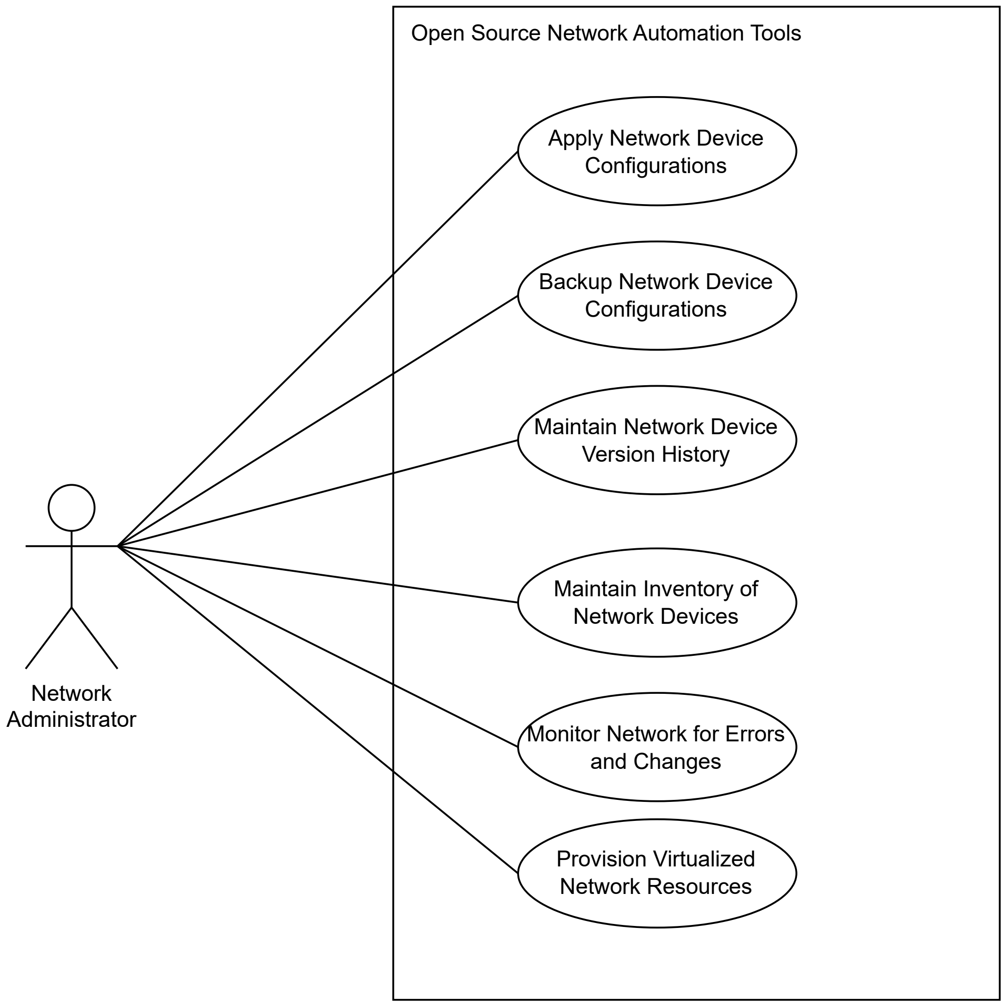
Open Source Network Automation Tools
Apply Network Device Configurations
Backup Network Device Configurations
Maintain Network Device Version History
Maintain Inventory of Network Devices
Monitor Network for Errors and Changes
Provision Virtualized Network Resources
Network Administrator

Figure 4. UML Use Case Diagram for using Open Source IT Automation Tools to Implement Network Automation

Additionally, the analysis of software-defined network had made apparent the existence of networks, which may use SDN for some parts, but still may require other solution for managing certain parts of network infrastructure. For example, when an overlay model is used, leaving the need continue managing the control plane of the hardware devices, or in a hybrid model where parts of the network remain using traditional networking plane structure. For the aforementioned reasons, the following analysis will discuss and evaluate the suitability of various open-source IT automation tools.

### 1.4.1 Overview of Open Source Automation Tools

In order to understand the capabilities and aspects of automation that can be achieved using open-source automation tools. Firstly it is imperative to establish categories of automation tools according to their common use cases in the sphere of computer network management. Analyzing tools according to their category is appropriate, since tools in same category may either have a major overlap in features, thus having similar use cases. Conversely, some tools in same category can have complementary features and integrate well with each other. Thus, the paragraphs below will analyze categories of open-source tools suitable for network automation, as well as the specific tool therein.

**Network Configuration Management Tools** These types of tools are commonly described as suites of software tools, designed for automating configuration tasks, often using declarative scripts, which have to ability to be configure multiple devices using a single run of a script. The use of such scripts, in combination with the use of version control systems for storing various imperative and declarative scripts, code snippets and configuration files creates the basis for Infrastructure as code (IaC). IaC being a set of processes that use the automated management of IT infrastructure, including networking, as a basis for improving the deployment times, reliability and scalability of IT infrastructure. Examples of such tool suites include Ansible, Puppet, Chef and Saltstack. All of the aforementioned tools have similar use cases and considerable communities around them. All of these tools support the most popular devices from major vendors such as Cisco and Arista. Alongside that, the nature of these tools allows wide support with Linux and UNIX systems, thus many of them can be used with popular open-source networking operating systems such as VyOS and Pfsense. Such these tools use a principle of having a central control node, that connects to network devices via protocols such as SSH (or Telnet for legacy devices) and can run certain sets of commands via scripts, upload configuration files. In such a way allowing for automated configuration of network devices, without the need to connect to each device manually and run commands by hand. Though evidently, this requires writing or obtaining these configuration scripts in the first place. That, combined with the setup time and expertise required, means that implementing automation of a network using these tools will require an investment of human hours by operations teams. the number of hours of course dependent of the proficiency and experience of network administrators with the specific automation tool at hand. The setup time required may also depend of the size and structure of a specific network.

For example: an Ansible master playbook could be used, in conjunction with Jinja templates and plug-ins for different device operating systems. While running the playbook, the control node would connect to each network device and make the required changes. For example, adding a firewall or access control list rule in each device in such a way as to allow the host virtual machine to gain access to a virtual server located in a server on another network. This playbook would only need to be written once and via the use of template variables and parameters could be run many times, when needed, to allow new host virtual machines access to said virtual server.

The UML deployment diagram shown below is an example of a network topology, and how connections to network devices would be made upon running an Ansible playbook or similar script. It demonstrates the use case of such an Ansible playbook for automation of configuration for multiple access-control lists and firewalls.

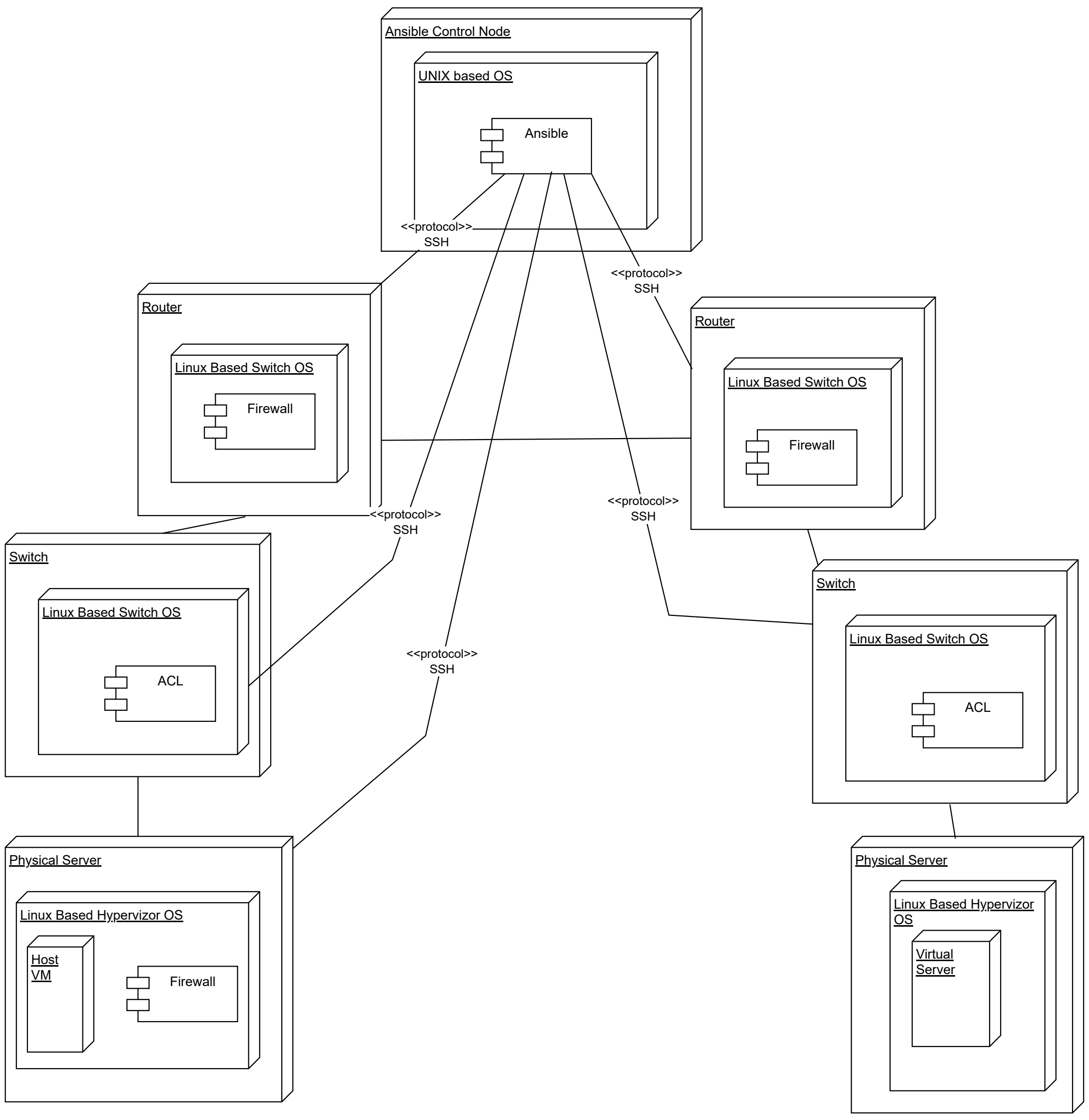

Figure 5. UML Deployment Diagram Showing Use of an Ansible playbook for Configuring ACLs and Firewalls

Additionally, infrastructure as code tools such as Terraform may be used in cloud-native environments, deployed on cloud providers such as Amazon AWS, Microsoft Azure or Google Cloud.

There also exists a variety of proprietary, vendor specific network automation and configuration management tools, such as Cisco DNA Center, Junos PyEZ or HPE Intelligent Management Center. These tools will not be analyzed and discussed, since their nature goes beyond the scope of this paper, which focuses on the analysis of open-source tools, protocols and standards.

**Network Monitoring and Inventory Management Tools** Such types of tools provide automated or partly-automated discovery, monitoring, configuration versioning, among other related features. This category of open-source network automation tools include monitoring systems such as LibreNMS or Grafana as well as configuration versioning tools such as RANCID or Oxidized. Examples of automation in such systems include automatic discovery and inventorization of network devices with tools like LibreNMS and automatically collecting and saving versions of configurations with tools like RANCID or Oxidized. The tools mentioned also have varying levels of integration between each other. For example, Oxidized and LibreNMS have the ability to integrate, forming a system in which automated inventory of network devices is kept and allows system administrators to view device configurations and their versions in a timely and convenient manner via a web interface, without the need to manually connect to each device. LibreNMS also has the ability to integrate with Grafana for altering and data visualization via a back-end database such as Prometheus or Graphite.

The use of the aforementioned tools and the like can save time, as well as aid in troubleshooting, as well as help make network changes. As they provide a reference point, in case of configuration versioning systems. Many of them also may provide altering in cases of configuration changes, errors or outages. Such features of the tools mentioned have the ability to greatly help network administrator to improve the reliability of the networks at hand.

**Network Automation in Virtual Environments** Network automation can be especially important in the case of virtual environments. This is due to the ability for vast scalability and rapid deployment of new virtual machines or containers. This means that manual configuration may not even be a viable option in cases where hundreds of virtual hosts are deployed. Fortunately, virtualization platforms can provide support for automating certain features of the networks therein. For example the open-source virtualization platform Proxmox Virtual Environment has a rather user friendly firewall solution based on iptables. [6] While it may not provide sufficient levels of automation in and of itself, due to the Debian Linux based nature of the Proxmox Virtual Environment platform, existing automation systems could be used to automate aspects of firewall deployment and firewall rules.

### 1.4.2 Advantages of Open Source Automation Tools

The deployment and use of open-source automation tools brings various benefits to network administrators and the networks within their realm of responsibility. As was described in the overview section, such tools are able to automate common processes of computer network configuration, as well as increase observability of the IT infrastructure in an organization. The observability may even extend beyond the networking sphere, as the aforementioned tools have to ability to monitor servers alongside network devices. This can alert system administrators of outages that may be unrelated to the network, for example, they may alert of unfavorable environmental conditions such as temperatures, as well as monitor the power drawn by devices in a datacenter. Thus, implementing certain systems for the use of network automation has positive effects that extend beyond the management of network devices.

Another important aspect of open-source network automation tools in multi-vendor support. As the tool suites are not managed or maintained by one particular vendor, they support most equipment currently in use in datacenters across the world. The use of protocols, such as SSH and SNMP mean that automation can be achieved not only with the latest hardware, but also with much of the legacy equipment that may be over a decade old. This a key advantage that provides much improved flexibility, compared to some of the alternatives in the network automation sphere.

Additionally, the cost of deploying many of those system may be far lower than fully integrating previously mentioned technologies, such as software-defined networking. Assuming that an organization has existing private cloud infrastructure that allows the deployment of virtual machines, as well as administrators with moderate proficiency, these systems can be deployed at little cost. The open-source tools mentioned often do not require the purchase of licenses, even for commercial use. Some projects such as Observium, may have been commercialized, though open-source forks can often fill the void and provide similar capability at no cost, with the increased transparency that comes with an open-source initiative.

### 1.4.3 Limitations of Open Source Automation Tools

Evidently, open-source projects, developed and maintained by their respective communities may also have limitations. Many of the projects mentioned, such as LibreNMS and Oxidized have rather small teams of people working on them, often part-time. This means that bug fixes, improvements and new features may not be released at a timely manner. Additionally, the lack backing from major organization or vendor means that support will be available only from the community, which uses the same systems. This can sometimes be a benefit, as issues can be identified and documented by multiple persons and in different scenarios. The downside, of course is that when a bug or problem has been identified, it may require waiting a considerable amount of time to be fixed, or require work-arounds which include manual code patching which may also not be ideal for many organizations. An example of this can be seen with open-source the network device configuration versioning tool Oxidized. The latest version of the Pfsense open-source software router/firewall is not compatible with the latest version of Oxidized. This issue has been identified and a manual code-fix provided. [10] The fix has also been merged into the development branch of Oxidized, though, at the time of writing these fixes have not yet been addressed in the latest release of Oxidized (0.29.1). This example highlights a particular disadvantage of using open-source systems. Though, it is pertinent to emphasise that this disadvantage may be relative in comparison to other solutions. An active community, even one where changes to releases are made slowly is still favorable, in comparison to a commercial project that is no longer maintained, which may the cases with some legacy systems still in use.

### 1.4.4 Choosing Suitable Open Source Software for Network Automation

When seeking to choose a set of tools suitable for the network of a particular organization it is important to consider multiple factors at play. When choosing network configuration management tools, an organization needs to first consider the vendors and operating systems of the network devices that are a part of the networks, within the respective scope of management. The set of tools chosen has to have the modules needed to properly interface with and run commands and scripts on the existing network devices. Future growth may also need to be considered, that is, is the chosen set of tools likely to support hardware that is to be integrated into the network in the short to medium term. Another important factor to consider is the experience and expertise of the personnel of the operations team. Since different sets of tools use different scripting languages for configurations, for example, some use domain-specific languages, such as in case of Puppet and Chef, and others, like Ansible and Saltstack use YAML. Familiarity with the underlying programming languages may also need to be considered. Since the familiarity of staff with the programming language that a system in use is written in, may help speed up deployment and allow more in-depth troubleshooting or even creating new modules based on specific needs. Additionally, Chef and Puppet are older and more established, which may be favorable for organizations in which stability is a priority. While Ansible and Saltstack may be favorable in use cases where the tasks are more simple. [17] In terms of capability, none of the systems in question should be discounted, as all may provide robust and extensive support for the automation of various kinds of network configurations. For example, with Ansible, automation of IP address assignment to interfaces, as well as BGP routing configurations can be applied. [7]

Choosing open-source network monitoring and inventory management tools, entails the challenges of finding the projects that have not been commercialized and still have an active community. For example, the once popular network device configuration backup tool RANCID (Really Awesome New Cisco confIg Differ) seems to be getting less popular and receiving less maintenance than its self-proclaimed replacement Oxidized. Based on data from respective Github repositories, RANCID currency having the latest release 3.13 launched in October of 2020 [9], while Oxidized had the latest release 0.29.1 in April of 2023 [16]. Along with the more active community, Oxidized seems to be the rather obvious choice as of writing. Though in cases where there is already an existing system in place which works for the needs of an organization, replacing it with Oxidized, which as mentioned still has compatibility issues with popular networking software systems (such as Pfsense) may not be deemed desirable. Additionally, it is important to keep in mind that network device configuration backups may be performed in a fully or partially automated way using tools such as Ansible. When it comes to network monitoring, alterting and inventory upkeep options such as LibreNMS, Cacti, Nagios, as well as Prometheus and Grafana are available. The choice may depend on the administrators preferences and familiarity with the systems in questions, though LibreNMS, the community based fork of latest GPL-licensed of Observium version, is a strong contender. It has the ability to scan predefined networks and subnets for new devices, that it can then add to its monitoring inventory via configured SNMP credentials. It also has the ability to integrate with many of the aforementioned systems, such as Nagios, Proxmox Virtual Environment, Rancid and Oxidized [5]. This makes LibreNMS a worthy addition to many different organizations network management technology stack, integrating well with other systems to seek network automation. Picking automation tools in environments where extensive virtualization is used may depend on the virtualization platforms and tools used. Many cloud providers, such as Microsoft Azure, Amazon AWS, Google Cloud and others may have their own solutions for implementing network automation, potentially completing many tasks behind the scenes, out of the management scope of the customer, using such cloud services. In the case that an organization uses a self-hosted private cloud, tools such as Ansible, Puppet, Chef and others may be used for provisioning and configuring virtual network resources. Private cloud tool vendors, such as VMware, may also provide proprietary automation tools that may be better suited to their particular platform. Additionally, tools such as Terraform may be well suited for cloud environment as well, though it seems like the organization behind Terraform, like many other creators of open-source automation tools, has taken steps to restrict access to their code base by changing the licensing of Terraform.

This trend provides a bleak outlook and may have long term implications. As more and more useful, widely used tools move away from open-source licensing, access to freely available network automation may be shrinking. This may present future challenges for organizations which do not have the resources to acquire licensing, while additionally reducing transparency. This also makes network management more complex, as tools that are currently open and free to use may at any point become commercialized, with the open-source versions loosing future support, updates for new hardware and software as well as crucial security patches. Fortunately, numerous communities of programmers, system and network administrators exist, providing forks and updates, making the features of formerly open-source network management and automation tools, such as Observium, once again open to all.

# 2 Implementation Example

The following subsections will describe a scenario as well as provide example implementing the use of using open-source automation tools for computer network management. This example will deliberately be minimal, providing automation only for a very limited portion of a wider network. Due to limited resources it has been decided that such a proof-of-concept will be sufficient to illustrate the possibility of network automation, as well as some of the capabilities of open-source automation tools and technologies.

## 2.1 Mini-world Scenario

The potential scenario where this example could be applied is an organization, which has preexisting infrastructure, including private and public networks, subnets and servers. This organization has personnel with the expertise to manage self-hosted cloud infrastructure and are familiar with Linux and some of the common open-source automation tools. A new project is being carried out. It requires the setup of a new cluster for the deployment of virtual resources. It has been decided that a firewall will be needed to restricts the hosts from accessing each other, only having access to a web server, hosted in the same cluster. Since many hosts are planned to be deployed, it was decided that automation of host deployment, as well as the automated configuration of firewall rules will be needed. Thus the organization will need a solution of how to accomplish such tasks, utilizing tools and expertise that is already available.

## 2.2 Technological Considerations

For the implementation of the network automation example, it has been consciously and deliberately decided that exclusively open-source tools will be used. This commitment has been established due to both practical and ideological considerations. The tools chosen had to be freely available and widely used, having robust documentation. The example had to be a model, proving that network automation is achievable without extensive investments or resources, is realistic to accomplish for many organizations around the globe. Additionally, the software chosen had to be vendor-agnostic, meaning that it can be run on a vast majority of hardware, only requiring the support for industry standard instruction sets and virtualization technologies. Thus, this model has to provide the basis, proving that proprietary, commercial solutions and not mandatory to implement network automation.

## 2.3 Tools and Technologies

This subsection will list and describe the specific tools and technologies that were chosen and used for the implementation example. Firstly, the open-source Proxmox Virtual Environment virtualization platform was chosen. It is based on Debian Linux, is lightweight, has robust documentation, an active community. Importantly for this project, it contains a rather user friendly firewall based on iptables. A key point being that firewalls rules and configurations are easily accessible and can be configured both in the web-interface and by manipulating configuration files via scripts or commands. This provides the key ability to set needed firewall rules using a graphical interface, observe the configuration applied in the configuration file and potentially duplicate the rules without having to worry about specific syntax of the firewall configuration files.

Secondly, for the deployment of hosts inside of the Proxmox Virtual Environment, it was decided to utilize LXC Linux containers. Due to the minimal nature of the example, it was imperative to make the system as lightweight as possible, requiring minimal hardware resources to replicate. The performance advantage of LXC, as well as containers comes from the fact that containers share the kernel with the host, while traditional virtual machines such as KVM run a separate, additional kernel for each virtual machine. The performance benefits of containers, including LXC are well proven, especially in terms of disk and network IO throughput. [13]

Furthermore, the open-source automation toolset Ansible was chosen for automating the deployment of LXC containers on Proxmox VE, as well as for applying firewall rules and configurations to the newly created hosts. The decision to choose Ansible was one of personal familiarity, as well as robust community support. An open-source Ansible Role for deploying LXC containers on Proxmox had already been published on Github, thus simplifying the deployment automation process. [8] The containers being deployed were Debian 10 and Debian 11 containers, with Debian having been chosen due to being lightweight as well as famously stable.

Additionally, shell scripting was also utilize to run the Ansible Playbook (which deploys an LXC container and applies firewall configuration) multiple times, with different hostname, VMID and IP parameters. This may be a bit of a crude way to accomplish the deployment of multiple hosts, though for this particular implementation example this was deemed efficient and sufficient.

## 2.4 Structure of the System

In this subsection, the structure of the example system will be provided and described. For the deployment of the system itself, the OpenNebula Sunstone virtualization platform, provided by Vilnius University, Faculty of Mathematics and Informatics. This platform can in essence be substituted for most any virtualization platform, bare-metal server or personal computer, as the system was made with the purpose of multi-platform compatibility in mind. Next, a Debian Linux virtual machiene was used, and Proxmox Virtual Environment was deployed on top. When using other platforms or bare-metal hardware it would be favorable to use the ISO provided by Proxmox and install from scratch, though this option was deemed not feasible for the deployment on OpenNebula of VU MIF. Inside of the Proxmox Virtual Environment (PVE), the iptables-based firewall, included with PVE was setup, allowing the creation of firewall rules via the web-interface, commandline or scripts. Next, a Debian 11 LXC container for hosting the Ansible automation toolset was deployed, in which, the required Ansible Role, Ansible Playbook and shell script were uploaded and constructed. Next, an example web server was created, in order to demonstrate the use of firewall rules. The IP address of which is noted in the variables of the Ansible Playbook. From the Ansible LXC container, the series of imperative and declarative scripts is then executed, deploying multiple hosts, along with their appropriate firewall rules, via a single shell command. During the runtime of the scripts, the Ansible container connects via SSH to the virtual machine on which the Proxmox Virtual Environment is running, as well as contacting the Proxmox API in order to create the aforementioned host containers with the desired network configurations. The UML deployment diagram below demonstrates these interactions, as well as shows the overall structure of the example system.

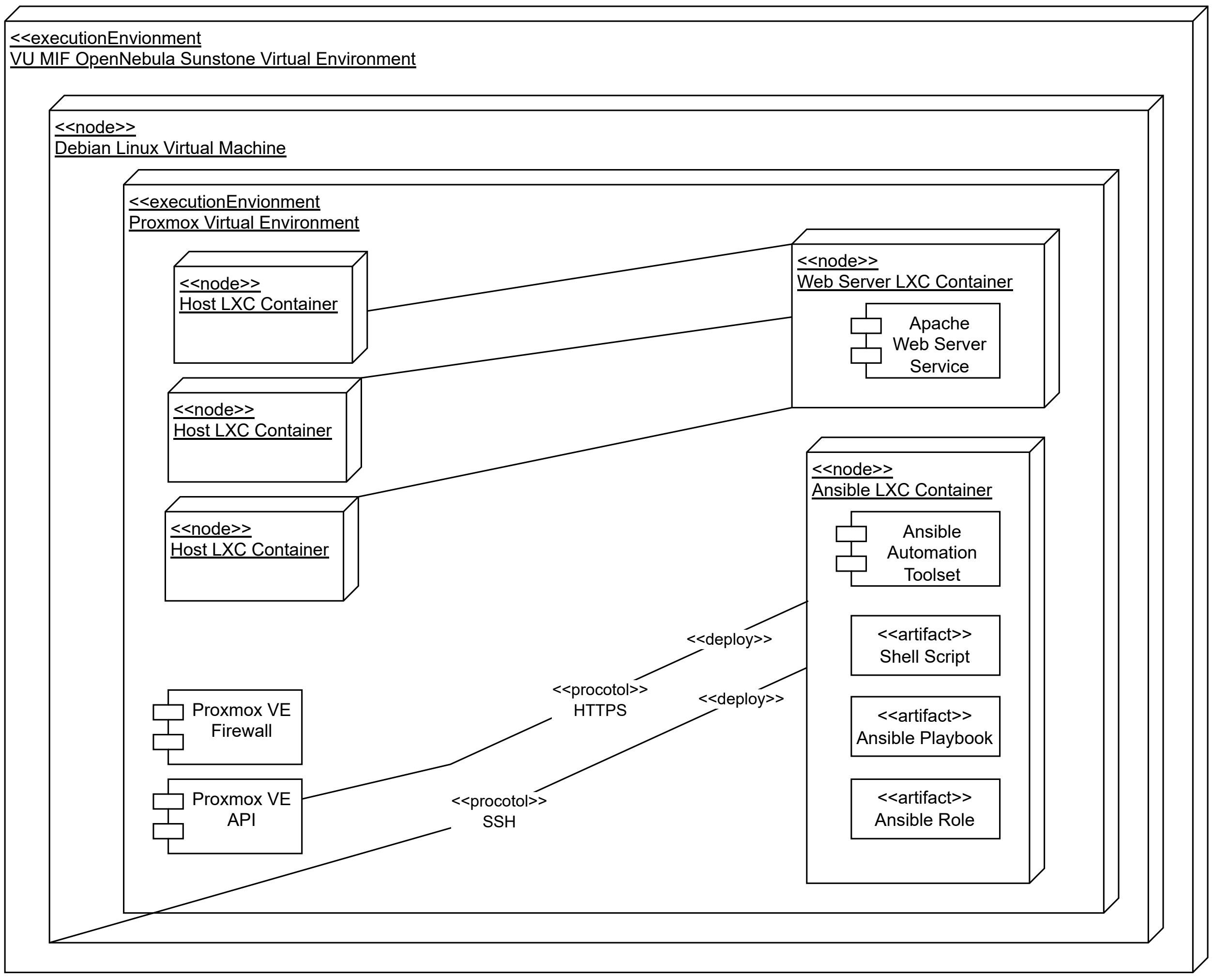

Figure 6. UML Deployment Diagram of the Example System

## 2.5 Environment and Networking Setup

In order to demonstrate the network automation example, an environment for deploying the virtual resources had to first be created and properly configured. Firstly, a Debian 11 Linux virtual machine was created in the OpenNebula Sunstone virtual environment. In order to install the Proxmox Virtual environment on the aforementioned virtual machine, appending the sources list in /etc/apt/sources.list.d/ was required, as well as adding the GPG key for the Proxmox VE repository. Proxmox VE 7.4-16 was then installed using the Debian apt package manager. The port 8006 for the web interface then had to be forwarded via OpenNebula Sunstone, so as to allow the access to the Proxmox VE web interface. Upon accessing the web interface, two Linux bridges were created to act as virtual switches for the example network. One of the Linux bridges (vmbr0), containing the virtual machine interface eth0, as well as having the IP address assigned by OpenNebula: 10.0.0.114, was for access to the 10.0.0.0/8 network of virtual resources in OpenNebula, as well as connecting to the 10.0.0.1 gateway for internet access. The second Linux bridge (vmbr1), was created for internal network of the Proxmox VE for hosts, the web server, as well as the Ansible container. The network address being 192.168.1.0/24. This network does not have any associated interfaces or ports on the virtual machine, thus, it is isolated from the rest of the network of OpenNebula virtual machines. This was done as to not interfere with the wider university network and avoid IP collisions or other disruptions to the faculty network infrastructure. The Proxmox VE firewall also had to be enabled. This had to be done in two places of the web interface: in the 'Datacenter' section, as well as for the specific node 'Debian', setting the default input and output policies to 'ACCEPT'. Furthermore, the container for the Ansible automation suite was deployed, though it, unlike the other containers required access to the internet for installing the required packages as well as pulling the Ansible Role for the automated deployment of subsequent containers. For this, an additional interface was created for the virtual machine in OpenNebula Sunstone, as to get an additional IP address that would be used for the Ansible container and allow for it to connect to the 10.0.0.1 gateway, thus getting access to the internet. For simplicity and stability, static IP assignment was used for the 192.168.1.0/24 network. It is important to note, that extensive security hardening had not been performed. That means that in some places of the example system, root users are used, simple passwords or other practices that may be considered insecure in a production environment. These decisions were taken to make the example system easy to test and troubleshoot, as well as being quicker to set up. Common security guidelines, such as robust passwords were however followed in the parts of the system which face the wider faculty networks and access the internet. Having acknowledged that, the setup continues. The Ansible container was populated with the all required software for the automated deployment of LXC containers and firewall rules. The web server container was deployed using the automated scripts. The IP of the web server was noted and set in the Ansible playbook variables, thus the subsequently deployed host containers are able to access only the web server container, all other outgoing traffic with destination addresses other than the web server IP 192.168.1.250 being rejected.

## 2.6 Declarative and Imperative Scripts

In order to achieve the goals of demonstrating network automation in an example environment, a combination of imperative and declarative scripts was utilized. Declarative meaning that the desired state was described, with the software then seeking to create said state. Conversely, imperative meaning that the steps and their order required to create resources were specified via a set of commands in a script, in a similar way to procedural programming.

For the creation of the LXC containers, a preexisting open-source Ansible Role was utilized. [8]. The parameters for execution being set via variables in the Ansible Playbook titled *pve-ct-deploy-playbook.yaml* and command-line arguments in the shell script *mass-deploy.sh*, which runs the Playbook.

The playbook was also appended with tasks, which create and apply the aforementioned Proxmox VE firewall rules, via the use of a Jinja template, that resides in the *templates* directory and is titled *example.fw*.

Some of the parameters for the Ansible Role, responsible for LXC container creation being:

- **pve_vmid** - the VMID to be used in Proxmox VE to identify the container;
- **pve_hostname** - the hostname of the container;
- **pve_node** - the node on which the container is to be deployed;
- **api_user** - the username and authentication method (Linux PAM or PVE) for the Proxmox VE API ;
- **pve_apipass** - the password for the Proxmox VE API user;
- **pve_api_host** - IP address or domain name for connecting to the Proxmox VE API;

- **pve_template** - the template from which the LXC container is to be created;
- **pve_netif** - network interface configuration for the LXC container;
- **pve_cores** - number of CPU cores assigned to the LXC container;
- **pve_mem** - amount of memory (in megabytes) to be assigned to the LXC container;
- **pve_swap** - the amount of disk space used for the swap memory;
- **pve_guest**_pass - the password for the LXC container;
- **pve_dns** - the DNS server IP for the LXC container;
- **pve_storage** - the storage to be used for deploying the LXC container (for example: local or ceph cluster);
- **pve_disk_size** - size of the disk for the LXC container;
- **pve_ssh** - public SSH key that is to be set in the LXC container;
- **pve_onboot** - specify whether to start the container upon the Proxmox VE node boot.

An additional parameter *web_server_ip* was utilized in the aforementioned firewall rule template *example.fw*. It is defined in the *pve-ct-deploy-playbook.yaml* and denotes the IP of the web server that the host containers need to have access to.

Furthermore, the aforementioned shell script *mass-deploy.sh* was created and used to run the Ansible playbook multiple times with certain incremental parameters via a loop. The incremental parameters being *pve_ct_ip, pve_hostname, pve_vmid*. For simplicity, the values of these three parameters are tied to the VMID. This introduces a limitation on the VMIDs that can be specified, as well as IP addressing. Since the final octet of the container IPv4 address is set to the VMID, the VMID has to be between 1 and 254, while also avoiding any of the address endings already in use, such as .1 for the Linux bridge, .102 for the Ansible container and .250 for the web server. The VMID range is passed to the shell script in 2 arguments: first argument being the VMID of the first container to be created and the second argument being the number of containers to be created. The script will then increment each VMID, IP address and hostname suffix by +1 for each container that is to be deployed.

## Conclusions and Recommendations

In conclusion, the two major goals set out for this work were achieved. One of them being the analysis of network automation methodologies, tools and technologies. During the course of the analysis it was found that software-defined networking will often require major changes to the network infrastructure, requiring major investments in hardware, as well as human hours. An alternative, in the form of open-source automation tools, was found. Tools such as Oxidized, LibreNMS and Ansible were proven to be able to automate some common, yet important tasks, such as network device configuration backup, inventory keeping and updating as well as configuration changes, applying configurations to new physical or virtual devices, hosts. Though a negative trend in the sphere of open-source automation tools was noticed. That being privatization and commercialization of formerly open-source projects. A few notable ones being Observium and more recently: Terraform. Thankfully, open-source forks (such as LibreNMS, being a fork of Observium) are stepping in to provide some of the same features, while having open licensing. Though it is not clear how some of these projects will evolve and if they will be supported in the long term.

Another goal of the project was to give an implementation example, enabling network automation using open-source tools. It was decided to automate the deployment and firewall rule configuration for virtual resources. This was achieved by the use of the Proxmox Virutal Environment, along with Ansible and shell scripts. With the series of scripts being used to deploy LXC containers in Proxmox VE, as well as to apply their respective firewall rules. This example demonstrated that certain network tasks can be automated using freely available, open automation tools in a vendor-agnostic environment, with the example system being widely compatible with different hardware or virtual machine environments.

Overall, this work has clearly shown the importance of open technologies, standards and tools. It is important to contribute to the development and maintenance of open-source tooling, as it provides benefits to all, especially in cases where purchasing proprietary, vendor specific automation products is not feasible. The open source community is providing crucial management and automation tools, helping to configure and maintain networks, that have a major impact on many of our lives.

## Future Work

As the given implementation model is just a minimal example of what is possible with open-source automation tooling. To make it suitable for production use, resources such as a DHCP server and NAT router would have to be deployed for the example network, with additional security hardening having to be performed as well. Moreover, work on maintaining and creating open-source automation tooling is crucial, as some tools that were already available were of great help, making it possible to automate deployment and configurations without having to design, create and program tooling from the ground up.

Additional research and analysis into preexisting open-source network automation tooling would also be of use, as exploring the combinations and integration among numerous suites may uncover particularly useful combinations, allowing to automate many more processes.

## References

[1] Muhammad Awais, Muhammad Asif, Maaz Bin Ahmad, Toqeer Mahmood, and Sundus Munir. Comparative analysis of traditional and software defined networks. Department of Computer Science, Lahore Garrison University, Lahore, Pakistan, 2021. https://ieeexplore.ieee.org/abstract/document/9526236.

[2] VMware by Broadcom. What is software-defined networking (sdn)? Broadcom Inc., 2023. https://www.vmware.com/topics/glossary/content/software-defined-networking.html.html.

[3] Chang Ching-Hao and Dr. Ying-Dar Lin. OpenFlow Version Roadmap. National Yang Ming Chiao Tung University, Taiwan, 2015. http://speed.cis.nctu.edu.tw/~ydlin/miscpub/indep_frank.pdf.

[4] Stuart Clark, Hazim Dahir, Jason Davis, and Quinn Snyder. Automation. Cisco Certified DevNet Professional DEVCOR 350-901 Official Cert Guide, 2022. https://www.ciscopress.com/articles/article.asp?p=3145761&seqNum=4.

[5] LibreNMS contributors. *LibreNMS Docs*, 2024. https://docs.librenms.org/.

[6] Proxmox VE contributors. *Firewall*, 2023. https://pve.proxmox.com/mediawiki/index.php?title=Firewall&oldid=11840.

[7] Anirban Datta1, A. T. M. Asif Imran, and Chinmay Biswas. Network Automation: Enhancing Operational Efficiency Across the Network Environment. Department of Information and Communication Technology, Bangladesh University of Professionals, Dhaka, 2022. https://icrrd.com/public/media/21-03-2023-154115Network%20Automation.pdf.

[8] Noe Gonzalez. Ansible Roles: Proxmox. GitHub, Inc, 2022. https://github.com/engonzal/ansible_role_proxmox.

[9] Github User haussli. RANCiD Tags. GitHub, Inc, 2023. https://github.com/haussli/rancid/tags.

[10] Github User jcheger. Release 0.29.1 breaks pfsense and opnsense config. GitHub, Inc, 2023. https://github.com/ytti/oxidized/issues/2771.

[11] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. Georgia Institute of Technology, 2013. https://ieeexplore.ieee.org/abstract/document/6461195.

[12] Aladhami Mahmood Mazin, Ruhani Ab Rahman, Murizah Kassim, and Abd Razak Mahmud. Performance analysis on network automation interaction with network devices using python. Faculty of Electrical Engineering, Universiti Teknologi MARA, Malaysia, 2021. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9431823&tag=1.

[13] Roberto Morabito, Jimmy Kjällman, and Miika Komu. Hypervisors vs. lightweight virtualization: a performance comparison. Ericsson Research, NomadicLab, Jorvas, Finland, 2015. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7092949.

[14] Arista Networks. *7050SX Series 10/40G Data Center Switches Data Sheet*, 2020. https://www.arista.com/assets/data/pdf/Datasheets/7050SX-128_64_Datasheet_S.pdf.

[15] Brent Salisbury. Inside Google's Software-Defined Network. Informa PLC, London, United Kingdom, 2013.
https://www.networkcomputing.com/networking/inside-googles-software-defined-network.

[16] Alexander Schaber. Oxidized Releases. GitHub, Inc, 2023.
https://github.com/ytti/oxidized/releases.

[17] Deeksha Srivastava. Chef vs. Puppet vs. Ansible vs. Saltstack: A Complete Comparison. Successive Digital, 2021. https://medium.com/successivetech/chef-vs-puppet-vs-ansible-vs-saltstack-a-complete-comparison-9af8f1790c0d.

[18] Cisco Systems. Use network automation to design, provision, and manage your network for improved business efficiency. Cisco Systems, Inc., 2023.
https://cisco.com/c/en/us/solutions/automation/network-automation.html#~why-cisco.