Complex internship: computer networks

Motivation

This internship aims to investigate the potential benefits for replacing VyOS, an open-source network operating system, with a NixOS configuration. Background for this is that building VyOS is like fighting an uphill battle. The requirered packages and dependencies to build VyOS are endless and need a specialist to solve. NixOS on the other side has it advantages in its reproducibility of packages, which means that packages are isolated from another and have no dependencies on each other. The question to ask now is: can we use the advantages of NixOS and reproduce the functionality of VyOS with it?

To resolve this we are going to analyse the networking functionality of VyOS. Then we derive 3 real world scenarios which include sub-functions of VyOS. These scenarios will then be build by using NixOS. To complete our proof-of-concept we will write a translator which converts a VyOS configuration, with the functinoality of the 3 scenarios, to the nixos configuration, so that afterwards both systems have the same scope of functions.

Background

VyOS

VyOS, a versatile network operating system and offers a wide range of use cases we will list some of them here.

- VyOS supports various routing protocols such as OSPF, BGP, RIP, and static routing
- It supports VPN technologies including IPsec, OpenVPN, L2TP, and PPTP, enabling secure remote access and site-to-site connectivity
- VyOS offers firewall functionality, allowing administrators to control traffic flow and enforce security policies
- It supports Network Address Translation (NAT)
- VyOS can function as a DHCP server, providing automatic IP addresses to client devices on the network.
- The system supports Virtual LAN (VLAN) functionality, enabling network segmentation
- VyOS supports IPv6.
- The operating system offers high availability features such as VRRP (Virtual Router Redundancy Protocol), ensuring network resilience and redundancy.
- VyOS can be deployed on various hardware platforms, including physical servers, virtual machines, and cloud environments, providing flexibility and scalability.
- It supports Multi-WAN load balancing and failover, distributing network traffic across multiple Internet connections for improved performance and reliability.
- VyOS offers a command-line interface (CLI) and a web-based management interface, providing multiple options for configuration and administration.

The full list of configuration options for VyOS can be found in the documentation: https://docs.vyos.io/en/latest/configuration/index.html#configuration-guide

While VyOS offers several advantages, it comes at a price that goes beyond monetary considerations. Although it is an open-source solution, utilizing VyOS effectively can be quite expensive in terms of knowledge and expertise. Building VyOS from source code is a cumbersome process due to its reliance on various packages, not to mention the complexities involved in customizing the system to suit specific needs. This requires a significant investment of time and effort to gain a comprehensive understanding of VyOS.

NixOS

NixOS is a Linux distribution known for its unique package management and system configuration approach. It is based on the Nix package manager, offering a declarative way to manage software and system settings. One of its key advantages is the ability to manage packages independently, meaning each package and its dependencies are isolated and self-contained. This approach avoids conflicts and ensures that upgrading or removing a package does not impact other parts of the system. Upgrades in NixOS are designed to be performed as a whole, ensuring the entire upgrade process is reliable and can be easily reversed if needed. With NixOS, the entire operating system is defined by a single configuration file, allowing for reproducible setups. To conclude NixOS is famous due to its reproducibility, reliability, flexibility, and the independent nature of its packages.

Task

To demonstrate the advantages of NixOS and show its ability to provide similar functionality as VyOS, we will replicate three specific use cases in both VyOS and NixOS configurations. Additionally, we will develop a generic transformer that takes a VyOS configuration as input and generates a corresponding NixOS configuration.

The transformer will automate the process of migrating from VyOS to NixOS for the specified use cases, ensuring a smooth transition and replication of functionality.

While the initial transformer implementation will focus on the specified use cases, it can be extended by mapping additional files and configurations to cover more scenarios. This flexibility allows for customization and adaptation of the transformer to suit specific needs beyond the initial use cases. For further information see Transformer.

By successfully replicating the use cases and providing a tool to transform VyOS configurations into NixOS configurations, we aim to highlight NixOS's ability to offer comparable functionality to VyOS while using its unique features such as package isolation, reproducibility, reliability.

Requirement analysis

This section lists all requirements to complete this internship.

- R1: Analyse the functionality of VyOS and NixOS
- R2: Derive three use cases of VyOS and build the scenarios using NixOS
- R3: Develop a transformer to convert VyOS configurations to NixOS configurations
- R4: Assess the advantages and disadvantages of replacing VyOS with NixOS

Use cases

For our proof of concept we choose some realistic use cases, where we provide the VyOS functionality by NixOS. While defining our goal we divided our use cases by komplexity.

The first one rebuilds a DHCP server with three clients that automatically receive an ip-addresses. Additionally to the DHCP we configured a bonding interface between two NixOS systems.

In the second scenario we configure a wireguard VPN which simulates a remote worker to be connected to an onsite network.

The third scenario addresses the routing functionality of a VyOS device. Therefore we configure three routers automatically exchanging network topology information as autonomous systems.

Further description and explanation about the scenarios can be found in the according folders in this repository.

Approach

NixOS installation

To set up the scenarios, we choose VirtualBox as our virtualization tool. VirtualBox is a free and open-source software that enables the creation and operation of multiple virtual machines on a single physical computer. This choice helps minimize research costs since no specialized equipment is required.

To begin, we downloaded the minimal ISO of NixOS from the official website: https://nixos.org/download.html#nixos-iso

We selected the minimal ISO because it offers lighter system requirements, which is beneficial when running four virtual machines simultaneously. There is also no the need for graphical software in our scenarios. Because of that, the command-line interface suits our purposes.

Next, we installed the ISO onto a newly created virtual machine, following the instructions provided on the NixOS website: https://nixos.org/manual/nixos/stable/index.html#sec-installation-manual-partitioning.

Once the virtual machine was operational, we proceeded to create virtual duplicates of it. These duplicates served as the starting point for our scenarios.

VyOS to NixOS transformer

What it does?

To complete our proof of concept we designed a translator which takes a VyOS configruation as an input and gives you a NixOS configuration as an output. This is possible for our four use cases:

1. DHCP server
2. Bonding
3. Wireguard configuration
4. BGP router

How to use it?

1. To obtain the VyOS ocnfiguration of a running system you can utilize one of the following commands:

```
show configuration > config.json
show configuration json pretty >config.json # only from version 1.3
available
```

(Before proceeding, ensure that Python 3 is installed on your NixOS system.)

2. Next, transfer the 'config.json' file along with the '/transformer/*' directory from this repository to your active NixOS system. Ensure that the 'config.json' file is located next to the 'transformer.py' file.

3. Proceed by running the 'transformer.py' script. After completion, it will generate a 'configuration.nix' file.

4. Copy the 'confiuration.nix' file to '/etc/nixos/configuration.nix' and apply it to the system by executing:

```
nixos-rebuild switch
```

How does it work?

The algorithm converts VyOS configuration files in JSON format to Nix configuration files in Nix format. It accomplishes this by taking into account the VyOS configuration and mappings, which are created for each transformation scenario. The VyOS configuration is extracted from an active system, allowing the algorithm to work directly with real-world data. On the other hand, the mappings are manually designed and stored in the "/transformer/mappings/" directory. These mappings follow a specific syntax, explained in the Mapping syntax section. The mapping files can be expanded to cover additional services or functionalities, providing flexibility for future extensions.

The transformer itself is divided into three parts.

The preprocessor is responsible for collecting the data and processing the representation of it. Notably, it analyses the network interface names of the NixOS system to ensure that they are configured with the appropriate name in the nix configuration.

The mainprocessor is responsible for applying the specified rules of the mapping files to the extracted information from vyos and translate it to nixos config. This involves an iterative algorithm that scans for matches from the mapping file within the VyOS data. The algorithm supports two types of placeholders. The first type, $X (where X is any positive integer starting from 0), stores the value of the VyOS configuration at that place in a list indexed correspondingly, which later supplement the Nix path. The second type involves the use of regular expressions. Here, the algorithm compares the value from the VyOS configuration at that place with the regex. If a match is found, the algorithm continues comparing the paths from VyOS with the mapping path until it verifies a full match. The use of regular expressions also allows for group matches with round brackets. These groups are added separately to the list of stored values and can be utilized within the NixOS configuration path of the mapping. For each match, the configuration is immediately translated and stored in the NixOS configuration.

The postprocessor, is designed to handle special configuration files for the use cases DHCP and BGP. These configurations, while included in the configuration.nix file, follow a different syntax and require special considerations during creation. The mapping for these transformers is integrated into the code and relies on templates equipped with placeholders. These templates are stored in the following dictionaries in this repository: '/bgpTemplates' and '/dhcpTemplates'. When information is detected, it is immediately injected into the template. To guarantee a functioning syntax for the services, all lines lacking values are removed at the end of the process. Similar to the main transformer, these templates require a functional VyOS configuration.

The reason why we splittet these cases from the mainprocessor are the different mapping language requirements for these configurations. For instance, unlike pure Nix config syntax, DHCP and BGP syntax can contain repeated sections requiring unique mapping treatment and programming logic like loops in a mapping language. Furthermore the needed values for DHCP need to change the representation style of the subnet mask from '/24' to '255.255.255.0' or calculate a broadcast address from the given network address and the subnet mask.

After all the processing stages, the main file combines all the configurations into a single string and exports it to the 'configuration.nix'.

Mapping syntax

--> Describe in JSON format with this syntax we create the opportunities to map X requirements to 1 nixos configuration. 1 to 1 configuration checking for certain values with regex expressions and therefore have some kind of if statement insode the mapping language.

Comparisons

This section will discuss the benefits of using NixOS for VyOS functionality.

Project ideas rise from problems. One problem of using VyOS as a router software is to create your own executable. Despite VyOS is open source it is very hard to build your own version of it. The reason for this is that VyOS has a lot of dependencies and packages which are not easy to install. The VyOS documentation provides a guide to build your own version of VyOS. This guide is very long and requires a lot of knowledge about the VyOS system. Building your own version of VyOS basically has two main adavnatages for network administrators. First you can customize the VyOS system to your needs. Second and more important you can audit the code for security vulnerabilities and be sure that no changes have been made to the code before compiling.

Based on this motivation we now want to conclude our internship with a comparison of VyOS and NixOS. We will compare the two systems by the following criteria:

Work effort to build nixos and vyos from source code

As described above building VyOS is cumbersome. NixOS on the other side rebuilds itself every time you apply the configuration. In NixOS it is even possible to build an ISO with your own configuration, which can then be used for later installations. This is a clear advantage for NixOS.

Usability of a running system and learning curve

This section will discuss the usability while using each system once they are running. This includes the learning curve to get familiar with the syntax and the complexity of the syntax.

Starting with NixOS the syntax for the Nix intern configuration is easy to understand. The complexity starts at the point when you want to add aditional services that are not directly included and need special configurations for the deamons. These configurations follow their own syntax. So for each service you need to learn a new one. VyOS on the other side has a unified and simplified syntax for all services. This makes it easy to learn and understand. Setting up the scenarios for this project was three times faster for VyOS than for NixOS. Comparing the lerning curves of both systems, they both start with the same rise but at some point VyOS leaves NixOS behind beause the syntax stays the same. Applaying a new configuration is easy and fast for NixOS and VyOS. Therefore you just need to use one/two commands. Looking at the functionality of both systems, VyOS probably has the same range of functions as NixOS. But using all functions in NixOS requires far more knowledge in different services and deamons. This is because VyOS is specialized on networking and NixOS is a general purpose operating system. To conclude this section, VyOS has a clear advantage in usability and learning curve.

Maintainance cost and security aspects

As discussed above NixOS has it advantages in the building process and VyOS in the usability. But there are also other aspects to consider. One of them is the maintainance cost. This includes the effort to update the system and the security aspects. Updating NixOS is automatically done every time you rebuild the system with the new configuration. VyOS on the other side requires a manual update. This is a slightly advantage for NixOS. Becuase both distributions are open source the security risks of malicius code are very low. But the risk of security vulnerabilities is higher for NixOS. This is because NixOS is a general purpose operating system and therefore has more attack vectors. VyOS on the other side is specialized on networking and therefore has less attack vectors. In this comparison they are both almost equal.

Summary

Our internship project entailed an analysis and comparison of NixOS and VyOS, two distinct Linux distributions, in an attempt to optimize router software deployment. Each system carries its own distinct strengths and weaknesses.

The complexity of building VyOS from source is a clear disadvantage. On the other hand, NixOS offers a more streamlined and automated building process, rebuilding itself every time a configuration is applied. Additionally, NixOS can generate an ISO with user-specific configurations.

In terms of usability and the learning curve, VyOS takes the lead. Despite the simplicity of NixOS's internal configuration syntax, its complexity spikes when incorporating additional services. In contrast, VyOS maintains a unified and simple syntax across all services, enabling quicker set up and reducing the learning curve.

However, the versatility of NixOS and VyOS might be restricted in certain contexts. NixOS is a general-purpose operating system that can handle various tasks but may require more extensive knowledge and time to configure specific functionalities. VyOS, on the other hand, being a specialized networking OS, is more straightforward in its networking functions but might not be as flexible.

In the end, NixOS and VyOS both leverage the Linux kernel, so they have similar capabilities and can run on the same machines with similar drivers.

However, given the specialized nature of VyOS, we do not recommend replacing it with NixOS, despite NixOS's more manageable building process. The time, that can be saved setting up the environment can not compete with the time saved with the more easy configruation and modification in the running process. Therefore we rate the range of functions and the usability higher than the trustworthiness you gain by building the system by your own with NisOS. People who have a high priority for security and trusworthiness should decide on the size of the project if they use NixOS or try to build VyOS by themselves. The second option is recomended for projects with multiple service requirements and a chance of increasing network capabilities in the future.

Future research in this field could take two directions. The first involves creating a NixOS configuration capable of building the VyOS ISO from source code, effectively bypassing the cumbersome manual building process of VyOS while reaping the benefits of both operating systems.

Alternatively, researchers could consider the utilization of a JSON processor like "https://jqlang.github.io/jq/" or an XSLT transformer "https://www.w3.org/TR/xslt-30/#json" to engineer a generic transformer for all configuration files