SSoT Based Network Service Deployment

Bachelor Thesis

Department of Computer Science

OST – University of Applied Sciences
Campus Rapperswil-Jona

Semester: Spring 2023

Version: 1.0
Date: 2023-06-16 14:53:53+02:00
Git Version: Not available

Project Team:	Dejan Jovicic
	Dominic Walther
Project Advisor:	Urs Baumann
External Co-Examiner:	Patrick Mosimann, Cisco
Internal Co-Examiner:	Mitra Purandare, OST

OST
Eastern Switzerland
University of Applied Sciences

School of Computer Science
OST Eastern Switzerland University of Applied Sciences

Acknowledgements

We would like to express our sincere gratitude to all those who have supported us throughout this study.

In particular, we would like to express our appreciation to our supervisor, Urs Baumann. During our weekly meetings, he always took enough time to answer our questions and through his pexerience and expertise he provided invaluable insights that aided our comprehension and problem-solving. We are especially grateful for his accessibility and prompt assistance through the entire duration of this project, even beyond our scheduled meetings.

Abstract

In the field of network automation, a fully *Single Source of Truth* (SSoT) based deployment is considered something of a holy grail, as it promotes reliable, repeatable, and documented procedures. While many proprietary solutions exist in this space, many of the open-source alternatives currently lack in capability, ease of use, or code quality.

At its core, NetBox is an open-source project composed of a pre-defined database structure and a Django-based graphical interface. Its purpose is to document network infrastructure, covering both the hardware itself and its configuration. However, it lacks a mechanism for deploying said documented network in an automated manner, where the documentation dictates the network configuration, rather than the inverse.

The goal of this project is to expand the capabilities of NetBox by adding support for network services through a plugin and implementing a mechanism for generating and deploying the corresponding configuration to the relevant network devices.

We began by researching multiple network services before settling on MPLS L3 VPN, which encompasses the LDP, VRF, and BGP protocols, of which only VRF was already part of NetBox. After comparing various tools for network device configuration, we decided to use Napalm for device interactions and configuration management, with Nornir serving as a parallelization layer. During development, emphasis was placed on keeping the project extensible for future additions and making the deployment process as simple and intuitive as possible.

The resulting software is comprised of two parts: the Argos-NetBox plugin, which extends NetBox and adds support for the aforementioned protocols, and Argos-NAC, which queries the necessary data from NetBox and handles the generation and deployment of the various device configurations. Argos-NetBox can be used standalone and is vendor-independent, making Argos-NAC an optional addition to it. The protocols covered by this project can be used separately and are designed to be interchangeable, should alternatives to them be added in the future.

It is worth noting that, in addition to this paper, a separate MkDocs has been created which outlines aspects of the project relevant to its future development and offers an introduction to the dependencies upon which the project relies.

Lay Summary

Initial Situation

The concept of "Single Source of Truth" (SSoT for short) is crucial for IT companies as it ensures consistent use of a unified source for employees and automation tools alike. When it comes to network documentation, smaller companies often rely on tools like Excel sheets or other Microsoft applications to document networks. Unfortunately, this often results in multiple sources of truth, as employees create copies of the main document or write their own separate notes.

This is where NetBox comes into play. NetBox is an open-source web application specifically designed for documenting computer networks. It serves as the "Single Source of Truth" tool, encompassing all aspects of network documentation, from physical components like switches and routers to more abstract IP addresses and routing information. With NetBox, companies can ensure centralized and accurate network documentation, fostering better collaboration and consistency among employees.

Typically, network engineers using NetBox still follow the traditional approach of configuring devices first and then documenting the changes made. In an ideal Single Source of Truth environment, network engineers would reverse this process, documenting services and configurations first, which would then be automatically deployed to the devices. Since NetBox does not offer this functionality out of the box, external tools are required.

Currently, no such tools exist for network services, which is why we have been assigned the task of creating one. The objective is to expand NetBox by adding support for network services through a plugin and develop a solution that allows users to deploy documented configurations from NetBox directly onto network devices, streamlining the process and aligning it with the SSoT approach.

Approach

Put simply, a network service is a collection of protocols which, in conjunction, serve a common goal.

After some consideration, we settled on covering the MPLS L3 VPN service. The goal of this network service is to connect disjointed customer networks using a shared hardware infrastructure (consisting of both routers and cables) with minimal overhead and without the different customers' traffic interfering with each other. From the point of view of each customer, their traffic magically flows from one location to another, without needing to concern themselves with the routing specifics and at significantly reduced cost compared to dedicated infrastructure.

Most of the required protocols are not part of NetBox out-of-the-box. For this reason, Argos-NetBox was developed - a NetBox plugin which adds support for the LDP and BGP protocols, as well as tying all the relevant protocols together as part of an MPLS network.

With the help of Nornir and Napalm, a user's NetBox documentation is turned into actionable device configurations and sent to the respective devices as part of the Argos-NAC command-line utility.

During development of both Argos-NetBox and Argos-NAC, we focused on making our solution as extensible as possible to future additions, as well as simplifying the onboarding process for future developers and and the workflow for the end-users.

Results

Through the creation of Argos-NetBox and Argos-NAC, we have laid the initial foundation for a potentially expansive community-driven project. These two components enable users to effectively document a network service in NetBox and streamline its deployment to network devices, whether through an interactive or fully autonomous procedure.

In addition to this paper, we have prepared a comprehensive documentation using MkDocs which clarifies key aspects of the project essential for its future development. It also serves as an introduction to the project's dependencies, highlighting their significance and role in the overall framework.

Further Work

The primary objective of this project is to provide network engineers with an extendable foundation for documenting a broad range of network services, while also enabling the automation of their deployment. Our future plans involve expanding the scope of supported network services, as well as adding support for additional vendors.

Table of Contents

Listings

List of Figures

List of Tables

Part I

Technical Report

Chapter 1

Vision

1.1 Problem Statement

NetBox is a powerful documentation tool that combines IP address management (IPAM) and datacenter infrastructure management (DCIM) functionalities into a single application. It also provides an extensive collection of community plugins, and integrated APIs which allow network automation using NetBox data. However, despite its widespread adoption and active community, NetBox currently lacks the capability to document network services such as MPLS L3 VPN, VPLS, pseudowire and many other services. Additionally, there is no built-in mechanism to automate the deployment of the documented services within NetBox.

These limitations force network engineers to manually document network settings in NetBox and separately configure these settings on the corresponding network devices. Consequently, NetBox falls short of its potential as a fully reliable Single Source of Truth (SSoT) tool. Presently, it primarily serves as a documentation software for IPAM and DCIM purposes rather than fulfilling its envisioned role as the central hub for comprehensive network service management.

1.2 Our Solution

Our project seeks to extend NetBox by creating a plugin that enables the modeling of network services such as MPLS L3 VPN, VPLS, pseudowire and others. Furthermore, we are developing an external component that utilises the documented information in NetBox to deploy these services onto network devices. This component will incorporate vendor-specific details to ensure the configuration aligns with their requirements, while NetBox serves as the single source of truth for network services. With the intention of transforming the project into a community-driven endeavor, we aim to design the plugin and external component to be easily extensible for external developers.

In summary, the focus of this bachelor thesis is to create a NetBox plugin which network engineers can use to document network services before they are deployed and then using an external tool to deploy these documented services onto the network devices, making our NetBox plugin the single source of truth.

Chapter 2

Technical Framework

2.1 Network Automation Tools

In this section we will examine various automation tools we investigated as possible options for our project.

Command-Line Interface

The Command-Line Interface (CLI) is often the primary method junior network engineers learn to use for configuring network devices, due to its simplicity. In terms of automatability, however, it is often not the preferred choice.

One reason for this is that by using the CLI directly, the only way to determine whether an operation was successful is to anticipate how the device will respond to any command both in case of success and failure and compare this prediction against the actual reply received from the device. This approach is highly susceptible to changes in reply messages, as any changes to the formatting or phrasing of a reply may break its detection. Furthermore, some commands may not have an associated reply in cases of success, adding further difficulties in detection.

Additionally, when writing a script which uses the CLI directly, it is difficult to keep an overview of the state of the device being configured. Certain commands can only be run given a pre-condition. Others may need to be run in a particular context, for example on an interface, requiring the script to determine beforehand whether it has successfully entered the right context, whether further actions are required or if the context change itself has failed. This often requires complex state tracking and management in order to work reliably.

For these reasons, developing automated procedures using the CLI alone for anything but the most trivial procedures is inadvisable if other options are available.

Ansible

Ansible is an open-source automation tool that enables users to configure endpoint systems, deploy software and orchestrate workflows with playbooks. It is written in Python, which is an advantage as it is easy to learn and customize for individual use cases. The community is extensive, with plenty of documentation, modules and examples available for Ansible.

One of the key benefits of Ansible is that it does not require any software to be installed on the endpoints. It uses standard SSH and Python to execute tasks remotely, making it a lightweight and easy-to-deploy solution.

Ansible operates with an inventory of systems intended for connection and automation. The inventory file can be written in YAML, which contributes to the ease of use of Ansible, as it is straightforward to write and read. With the inventory file in place, Ansible can be employed to automate tasks through the use of playbooks. Listing 2.1 contains a sample inventory, consisting of groups of devices, each containing a hostname mapped to IP addresses. As shown, groups can even be nested, allowing for arbitrary granularity when targeting groups in playbooks.[1]

```yaml
all:
  controllers:
    hosts:
      ubuntu:
        ansible_host: 10.0.0.1
  routers:
    children:
      edge:
        hosts:
          R1:
            ansible_host: 10.0.0.2
      test:
        hosts:
          R2:
            ansible_host: 10.0.0.3
```

Listing 2.1: Ansible Inventory Example

Playbooks are a collection of one or more tasks that define the desired state of a system in a declarative way. They are written in YAML, just like the inventory, and allow users to define and execute several tasks. Listing 2.2 shows an example of such a playbook. In it, first the targets on which the play should be executed are specified via the "*hosts*" keyword. What follows is a list of tasks to perform on all selected devices, which in this case consists of a single instruction to ensure "apache2" is installed. Importantly, the task of deciding whether the application yet needs to be installed on any particular host is left to Ansible, avoiding unnecessary actions.

```yaml
---
- name: Install Apache
  hosts: web_servers
  become: true
  tasks:
  - name: Install Apache
    apt:
      name: apache2
      state: present
```

Listing 2.2: Ansible Playbook Example

The biggest disadvantage of Ansible is that, as the playbooks grows in complexity, they can become hard to understand and maintain. Additionally, while a lot of the functionality Ansible can provide is compartmentalized neatly into plugins, often even trivial operations require the employment of multiple plugins, further complicating the resulting playbook.

Nornir

Nornir is a user-friendly, Python-based automation framework that doesn't require proficiency in a domain-specific language, simplifying troubleshooting and debugging. The framework's ability to provide additional context whenever a task fails, further compounds this advantage. Nornir has extensive documentation and a very active community which has developed numerous plugins, making the framework even more versatile.

For network automation, Nornir supports plugins like Napalm, Netmiko, NetBox and NETCONF, which makes it very suitable for our project as a tool. Napalm, for instance, provides support for sending CLI commands to network devices, making it a valuable tool in various projects and scenarios. Just like Ansible, Nornir does not have to be installed on the remote node and makes use of inventory files written in YAML. An example of which can be seen in Listing 2.3. [4]

```yaml
ubuntu:
  groups:
  - controllers
  hostname: 10.0.0.1
R1:
  groups:
  - routers
  - edge
  hostname: 10.0.0.2
R2:
  groups:
  - routers
  - test
  hostname: 10.0.0.3
```

Listing 2.3: Nornir Inventory Example

StackStorm and SaltStack

StackStorm and SaltStack are both powerful automation tools, but they have different strengths and use cases.

StackStorm is primarily used to connect different services and automate workflows. For example by provisioning Docker containers on demand. However, it is not designed for endpoint configuration. Additionally, the complexity of StackStorm can be challenging to get into due to large number of options it provides. While it is possible to use it in conjunction with Ansible playbooks, for the purposes of our project, most of the functionality we would need it for could be replaced with simple cron jobs. [8]

On the other hand, SaltStack is a tool that is mainly used to maintain remote nodes, install software or ensure that services are running. It consists of a "master" node which communicates with its "minions" - agents that run a reduced version of SaltStack for local execution and communication back to the "master". This need for additional software to be installed on the endpoints (consisting of routers in our project) makes SaltStack a non-starter for our project.[7]

2.2 MPLS L3 VPN

In this section, we will take a closer look at the protocols and services necessary that form an MPLS L3 VPN service at a conceptual level. We start with an introduction to MPLS and the prerequisite terminology, then move from the backbone of the network towards the client's router, covering the various protocols we encounter and explaining their purpose in the network. An overview of the complete network (Figure 2.1), as well as a pre-configured lab were provided by our advisor Urs Baumann during the early stages of the project.

2.2.1 Introduction to MPLS

Multiprotocol Label Switching, or MPLS for short, is conceptually similar to VLAN, in that it allows for multiple, separated networks to co-exist on a shared physical network through label based packet routing. Where it differs from VLAN (beyond its implementation) is in its scalability and its support for Quality of Service (QoS) for more fine-grained resource management. It is considered to be a layer 2.5 technology, meaning that it inserts itself between the data-link layer, typically covered by switches, and the networking layer which is often dominated by routers and IP addresses. In this position, it inherits some properties of both layers - labels and tags, for example, are typically associated with switches and offer low-overhead packet forwarding, however MPLS uses exclusively router based protocols and often involves a mesh of routers as the backbone of the network.

The routers involved in MPLS can be divided into three different roles: Customer Edge (CE), Provider Edge (PE), and Provider (P).

A **CE** router is a router typically owned by the customer, and often has no direct knowledge of MPLS being employed in the network. It marks the boundary of the customer's infrastructure and is typically directly connected to a PE router.

A **PE** router is a router that shares a link with at least one CE router. Its main function is to provide services at the edge of the MPLS VPN, including MPLS-internal BGP and Virtual Routing and Forwarding (VRF) tables.

A **P** router is a router that does not have a direct link to a CE router, as it is part of the backbone of the network. As a Label Switching Router (LSR), it can therefore focus on forwarding only labeled packets, without needing to consider IP addresses or VPNs.

MPLS is a network service built on top of a variety different protocols and services, often with several options available to choose from for a given task. We will cover these tasks and protocols later in this section.

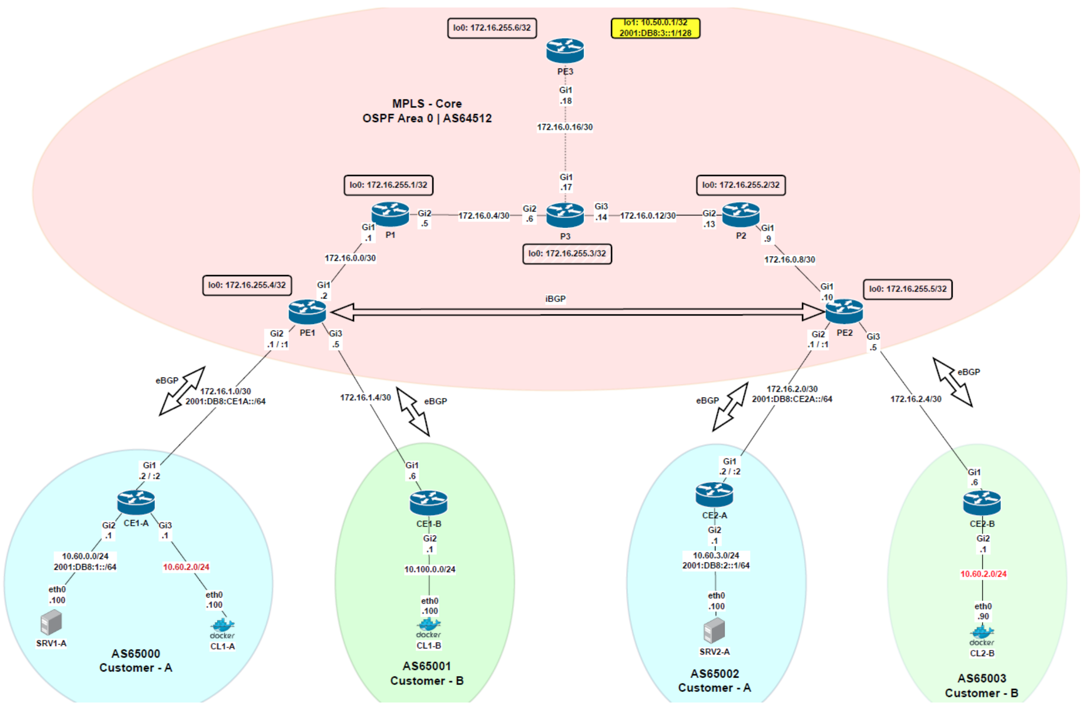


Figure 2.1: Overview MPLS L3 VPN Lab - provided by Urs Baumann

2.2.2 Tasks and Protocols

Let us take a closer look at MPLS by subdividing it into the various tasks that need to be covered for it to function, starting with the backbone, then moving outwardly until we reach the customer's own network.

P Routers

The backbone of an MPLS network consists entirely of a network of P routers, often assembled in a mesh-like topology. Its purpose is to route traffic between the various CE routers in a fast and efficient way. As such, each P router needs two tasks to be fulfilled in order to take on this role:

- It needs to know the topology of the backbonenetwork it finds itself in
- It needs to understand labels and how to relay them towards their destination

Regarding the first task, this is typically accomplished by an **Interior Gateway Protocol** (or IGP), allowing each router to learn the topology of the network by exchanging information with its direct neighbours through a protocol such as OSPF, IS-IS or RIP. This "Neighbourhood" forms what is called an **Autonomous System** (AS), whose members share a common routing policy.

Once a router knows where it can find the other members of the network, it needs to understand **labels**. Labels are a way to indicate the so-called "next-hop" a packet needs to be forwarded to in order to eventually reach its final destination. The job of a P router is to read the incoming label, look up which neighbour the packet is intended for next, replace the label with one known to said neighbour (in a process called **label swapping**), then forward the packet. This allows the router to play its part in delivering the packet to its destination while only needing partial knowledge of the labels used throughout the network. It can even participate passively in QoS without knowing of it, as different labels can correspond to different paths to the same destination for the purposes of congestion avoidance, but from the point of view of any individual P router these labels may have nothing in common. The task of label distribution is covered by the aptly named **Label Distribution Protocol**, which establishes various paths through the network, called Labels Switched Paths, or LSPs for short.

Once both of these protocols have been configured and the router is told to play the part of a P router in an MPLS network, it is ready to get to work and begin forwarding packets between various PE routers.

PE Routers

PE routers make use of the P-router-based backbone of the MPLS network and add a layer of abstraction to it. Their task is to direct traffic between fragmented customer networks via the backbone, while keeping unrelated networks separate from each other.

A PE router needs the following tasks covered by various protocols in order to fulfill its role:

- It needs to understand labels and know which labels lead to which PE routers
- It needs to know which customer networks it is responsible for and how to individually address networks
- It needs to know the other PE routers in the network and which customer networks they are responsible for
- It needs to exchange routing information with the CE router
- It needs to provide whatever service is built on top of the MPLS network (Eg. VPN, Pseudowire, QoS, MVPN)

When it comes to labels, the same concepts apply with PE routers as with P routers, in fact they may even share parts of the configuration. Where they differ is that all PE routers each have a range of labels assigned to them. The PE router can use these labels to denote different **Forwarding Equivalence Class**es (FEC), each of which are associated with a target PE router, as well as other information relevant for QoS. Multiple FEC may therefore lead to the same PE router, but through different paths in order to avoid congestion or improve latency. Additionally, PE routers are in charge of attaching an appropriate label to packets coming from the client's network (ingress) and removing the tag if they receive a packet from another PE router, before forwarding it on to the appropriate customer networks (egress). This determines the path the packet will take ahead of time with minimal overhead, as the entire path can be described with a single label which is swapped out by each hop between P and PE routers.

For a single PE router to be able to handle multiple distinct customer networks and keeping them separate, a single routing table does not suffice, as the different networks may want to use overlapping IP ranges. One approach to solve this issue is to use the **Virtual Routing and Forwarding** (VRF) protocol, which allows for multiple routing tables to be maintained by associating a network-wide unique identifier called a **Route Distinguisher** (RD) to each network or network fragment, and thereby to each routing table. This additional information, in conjunction with the IP address allows for the devices in each customer network or fragment thereof to be addressed separately, regardless of the used IP range. The ability to determine which network a packet originates from and to which network it is addressed allows the creation of **Virtual Private Networks** (VPN), as policies describing which networks may or may not communicate between each other allow for distinct virtual networks utilizing the same physical infrastructure. For this purpose, each PE router must know the unique **Autonomous System Number** (ASN) of each customer network it is directly connected to; this information is later shared with other PE routers to facilitate connections between these networks.

Once each PE router knows which customer networks they are in charge of, this information needs to be propagated throughout the network of PE routers (via the backbone) in order to allow fragmented customer networks to communicate between each other. This is the task of an **External Gateway Protocol** which, unlike its counter-part, the "Interior Gateway Protocol", is in charge of exchanging routing information *between* separate Autonomous Systems (AS). An example of such a protocol is the **Border Gateway Protocol** (BGP), which works by having each participating PE router advertising the AS' they known and the devices therein. Each PE router then takes note of this information in their routing tables, effectively associating an IP of a customer with a particular AS and the PE router managing it. This describes the internal process necessary for cross-AS communication and is typically called **iBGP** in an MPLS network.

This covers the internal process for discovering the provider's network topology. Now the PE routers must exchange the relevant knowledge they have gathered with the CE routers they are connected to and vice-versa, in a process called **Route Injection**. This also occurs via an External Gateway Protocol and, in case BGP is being used, is often referred to as **eBGP**. The end-goal is that the CE router knows whether a given IP range is reachable through the PE router, and the PE router in turn knows the customer network well enough to advertise the customer's IP addresses to the other PE routers in order to render this network reachable across the MPLS network.

2.2.3 Summary

Once all of these steps mentioned above have occurred, the network is able to offer its services to the customers. The concrete service being offered (Eg. Pseudowire, L2/L3 VPN, etc.) depends on the specific protocols which have been employed and the way they are configured. In summary, the core aspects which make MPLS are as follows:

- The provider's device communicate between each other via labels to reduce overhead
- The provider's topology, as well as which customer networks are present in the network is exchanged dynamically between the relevant devices
- The customer's devices have little to no knowledge of MPLS being employed, from their point of view the provider's network may as well consist of a single device
- The interchangeability and configuration options of the various protocols offer great customizability

2.3 MPLS L3 VPN on Cisco Routers

In this section we will cover the concrete steps for configuring MPLS L3 VPN on Cisco Routers, according to Figure 2.1 ”Overview MPLS L3 VPN Lab - provided by Urs Baumann”.

Label Distribution Protocol

In an MPLS network, each packet is assigned a label that is used to direct it through the network. To assign a label range to a particular PE router, one can run the command `mpls label range xxxx - xxxx`.

To enable MPLS on a router’s interface, the command `mpls ip` must be added to the interface configuration. This enables MPLS to create a label-switched path for IP packets, allowing the router to forward traffic based on labels rather than traditional IP routing tables. The router then starts advertising labels for IP prefixes to other routers in the network through the label distribution protocol (LDP). Once an IP packet arrives on an interface with MPLS enabled, the router uses the label to determine the next hop for the packet.

Virtual Routing and Forwarding

When creating a VRF, a unique Route Distinguisher (RD) must be specified to distinguish between different networks which may use overlapping IP ranges. The RD typically consists of two parts. The first part should be either an IPv4 address or an Autonomous System Number (ASN) and the second part can be arbitrarily chosen.

The RD should follow one of these two formats:

- **4-byte-int:2-byte-int** - ASN with identifier
- **4-byte-dotted-decimal:2-byte-int** - IPv4 address with identifier

MPLS also uses Route Targets (RTs) to determine into which VRFs a PE places iBGP-learned routes. RTs are advertised in BGP updates as BGP Extended Community path attributes. RT values follow the same basic format values of an Route Distinguisher. Most configurations use a single RT value, which is imported and exported by each VRF for a customer.

When creating a VRF, the line `address-family ipv4` is used to specify that the VRF will support IPv4 routing. Similarly, to add IPv6 support, `address-family ipv6` can be used. An example of a VRF config can be found in Listing 2.4.

```
vrf definition Cust_A
  rd 172.16.255.4:65000
  route-target export 1:1
  route-target import 1:1

  address-family ipv4
  exit-address-family
```

Listing 2.4: VRF Config Example Cust_A

Open Shortest Path First

Open Shortest Path First (OSPF) or any other IGP is required to establish IP connectivity between the routers. To configure it on a network interface, OSPF needs to be enabled globally on the router and given an OSPF ID. OSPF can then be enabled on the relevant network interfaces by specifying the OSPF ID and OSPF area ID for each. A sample configuration for interface-based OSPF is provided in Listing 2.5.

```
router ospf 1

interface GigabitEthernet1
  ip address 172.16.0.2 255.255.255.252
  ip ospf 1 area 0
```

Listing 2.5: OSPF Config Example

Border Gateway Protocol

Border Gateway Protocols (BGP) can be used both for exchanging routing information between the various PE routers (iBGP), and between PE and CE routers (eBGP), both of these require BGP to be enabled via the `router bgp <ASN>` command. With `no bgp default ipv4-unicast`, the automatic advertisement of IPv4 unicast routes to BGP neighbors can be disabled, which is enabled by default. The `address-family ipv4` command is used as a way of importing the ”ipv4” address-family into the BGP configuration. This forms the basis of the BGP configuration (Listing 2.6).

```
router bgp 64512
  bgp log-neighbor-changes
  no bgp default ipv4-unicast

  address-family ipv4
  exit-address-family
```

Listing 2.6: iBGP Config Example 1

IBGP is configured by first specifying the router’s neighbours, along with the interface it should use for the routing information exchange process, with the `neighbor <neighbor-ip> remote-as <ASN>` and `neighbor <neighbor’s IP> update-source <interface>` commands. As is typical with iBGP networks, a loopback interface is used as the ”updated-source” interface. Additional properties of the relationship between iBGP neighbours can be specified in the `vpnv4` and `vpnv6` address families, as seen in Listing 2.7.

```
router bgp 64512
  ...
  neighbor 172.16.255.4 remote-as 64512
  neighbor 172.16.255.4 update-source Loopback0
  neighbor 172.16.255.6 remote-as 64512
  neighbor 172.16.255.6 update-source Loopback0

  address-family vpnv4
    neighbor 172.16.255.4 activate
    neighbor 172.16.255.4 send-community extended
    neighbor 172.16.255.6 activate
    neighbor 172.16.255.6 send-community extended
  exit-address-family

  address-family vpnv6
    neighbor 172.16.255.4 activate
    neighbor 172.16.255.4 send-community extended
    neighbor 172.16.255.6 activate
    neighbor 172.16.255.6 send-community extended
  exit-address-family
```

Listing 2.7: iBGP Config Example 2

As we are also using BGP for the routing exchange between PE and CE routers, eBGP also needs to be configured here.

The lines `address-family [ipv4 | ipv6] vrf <vrf name>`, `neighbor <CE's IP> remote-as <remote ASN>` and `neighbor <CE-IP> activate` are used once for each combination of customer VRF and supported address family (Listing 2.8). This way, every PE router is informed about the VRFs and AS'es it is responsible for, which is later shared through iBGP with its BGP neighbours.

```
router bgp 64512
  ...
  address-family ipv4 vrf Cust_A
    neighbor 172.16.2.2 remote-as 65002
    neighbor 172.16.2.2 activate
  exit-address-family

  address-family ipv6 vrf Cust_A
    neighbor 2001:DB8:CE2A::2 remote-as 65002
    neighbor 2001:DB8:CE2A::2 activate
  exit-address-family

  address-family ipv4 vrf Cust_B
    neighbor 172.16.2.6 remote-as 65003
    neighbor 172.16.2.6 activate
  exit-address-family
```

Listing 2.8: eBGP Config Example

2.4 Pynetbox

Pynetbox is a Python API client library for NetBox under the *netbox-community* umbrella. Its primary purpose in the scope of this project is to act as an interface to NetBox in order to retrieve data from the underlying database.

2.4.1 Implementation Issues

Pynetbox is composed of a series of data structures and helper functions designed to simplify the interaction with the underlying REST API. Unfortunately, the way it is implemented comes with a long list of downsides.

Firstly, due to Pynetbox needing to mirror NetBox's underlying data structure, any additions made by plugins are not covered natively. While this may not be an issue for a lot of applications, this severely limits its usefulness in our case. Fully relying on Pynetbox for our data querying needs would require either the extension of the project or crafting custom requests, thereby foregoing much of the library's functionality.

A further issue is the mixed availability of type information when using it. While the Integrated Development Environment (IDE) is able to list all available children of a class, there often are no descriptions. Additionally, there is no type information for the indirect children of the primary classes such as `Api` and `App`. The reason for these shortcomings is the combination of several factors:

- complete lack of type-hints
- complete lack of docstrings for class fields
- lack of forward declarations for fields in cases where their initialization happens at a later point in time
- heavy and combined use of class composition and general-purpose classes

The lack of type hints could easily be fixed, as type hints were introduced in Python 3.7 and the maintainer of Pynetbox implicitly dropped support for Python versions older than 3.7[3] and the project's CI/CD pipeline only tests Python versions 3.8 and later.

Docstrings for class fields have been around since Python 2.1, but they seem to still be a rather obscure feature in the Python ecosystem. The lack of their appearance in Pynetbox is therefore of little surprise.

The lack of forward declarations appears to be an oversight on part of the developers. The current lack of type hints would actually simplify their addition, as assigning `None` to a field which will be initialized after instantiation of the class does not raise warnings during type deduction if there are no type hints to conflict with.

While class composition is likely the best approach for an API wrapper, its implementation through instances of general-purpose classes instead of through specialized subclasses thereof is not. It actively hinders the addition of type hints as there are no specialized classes to annotate the fields with. Classes such as `App` already behave like wrappers for multiple specializations and creating a wrapper class for each specialized `App` constructor would be a simple way to allow the IDE to distinguish them.

While these shortcomings can be overcome by relying on the NetBox API specification which it mirrors, it poses an unnecessary hurdle to the process of using the library.

A complete rewrite of the project is already in progress in the form of the *netbox-python* project, also under the *netbox-community* umbrella. Unfortunately for us, this project is still in its infancy as of the time of writing and sudden breaking changes are therefore to be expected. However it does seem to address most of the issues outlined above, making it a promising option for future projects.

2.4.2 Use Within Argos-NAC

Due to its downsides, Pynetbox is only used in a very limited fashion, namely to determine the presence and version of the Argos-NetBox in NetBox. By raising and exception if the plugin is not installed or is of an incompatible version, we aim to simplify the debugging process of end users. A well formulated error message early on in the execution of the deployment should prevent the user from being confronted with a much harder to parse GraphQL query or model initialization error.

Part II

Product Documentation

Chapter 3

Requirements

3.1 Storyboard

In this section, we will take a look at the planned workflow surrounding Argos. As it is intended to be used either interactively in a terminal, or without supervision as part of a script, Argos-NAC can be used in both modes of operation, with the choice being up to the user. As shown in Figure 3.1, in either case, data is only queried from the NetBox database once all necessary arguments are provided. This in turn is a prerequisite for establishing connections to the network devices to configure.

Another important thing to note is that checking the existing configuration on the device is necessary for the deprovisioning of certain configuration elements, as explained in subsection 4.1.1.

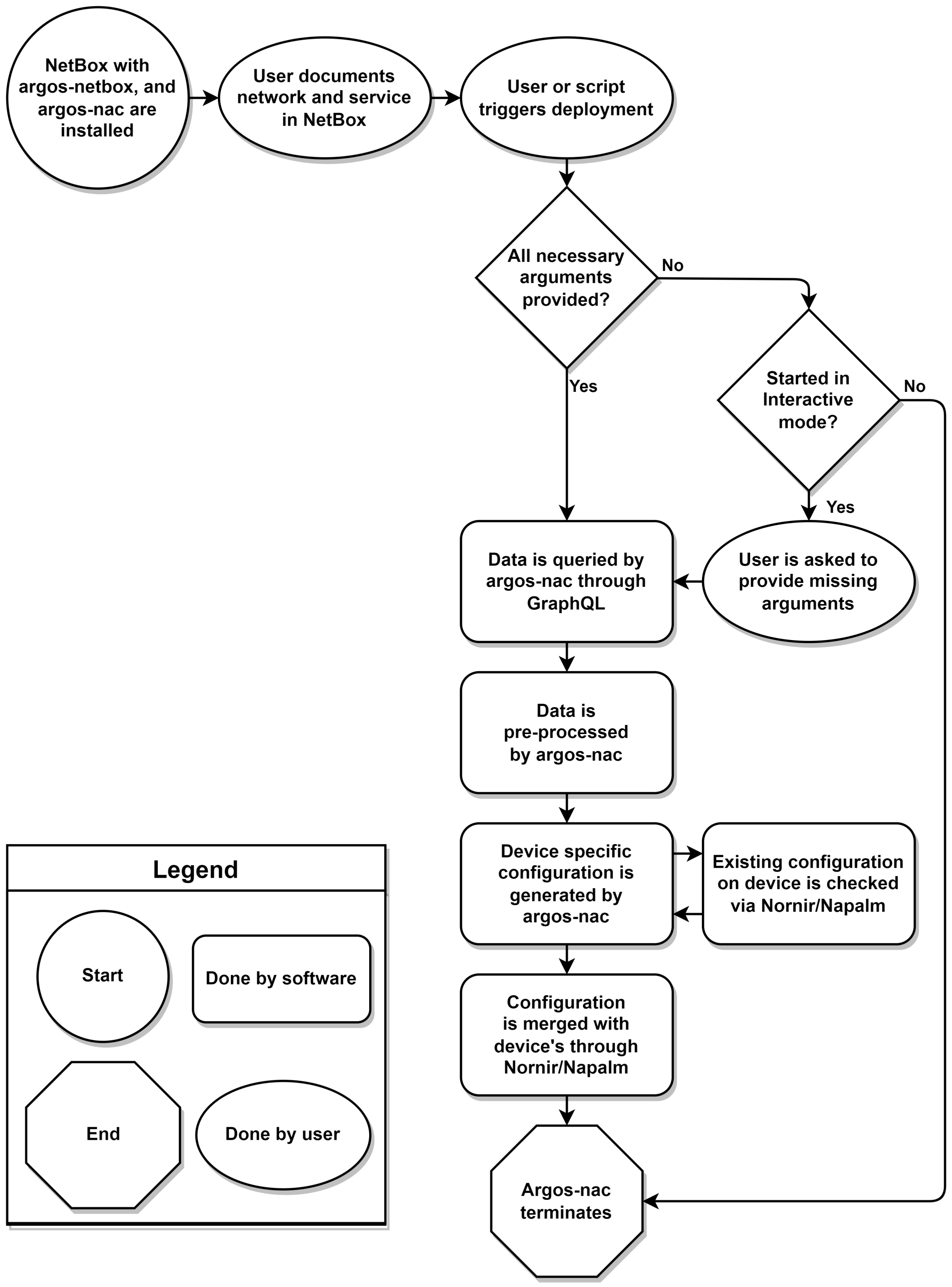

NetBox with argos-netbox, and argos-nac are installed
User documents network and service in NetBox
User or script triggers deployment
All necessary arguments provided?
No
Yes
Started in Interactive mode?
No
Yes
Data is queried by argos-nac through GraphQL
User is asked to provide missing arguments
Data is pre-processed by argos-nac
Device specific configuration is generated by argos-nac
Existing configuration on device is checked via Nornir/Napalm
Configuration is merged with device's through Nornir/Napalm
Argos-nac terminates
Legend
Start
Done by software
End
Done by user

Figure 3.1: Expected Workflow of the End-User

3.2 Functional Requirements

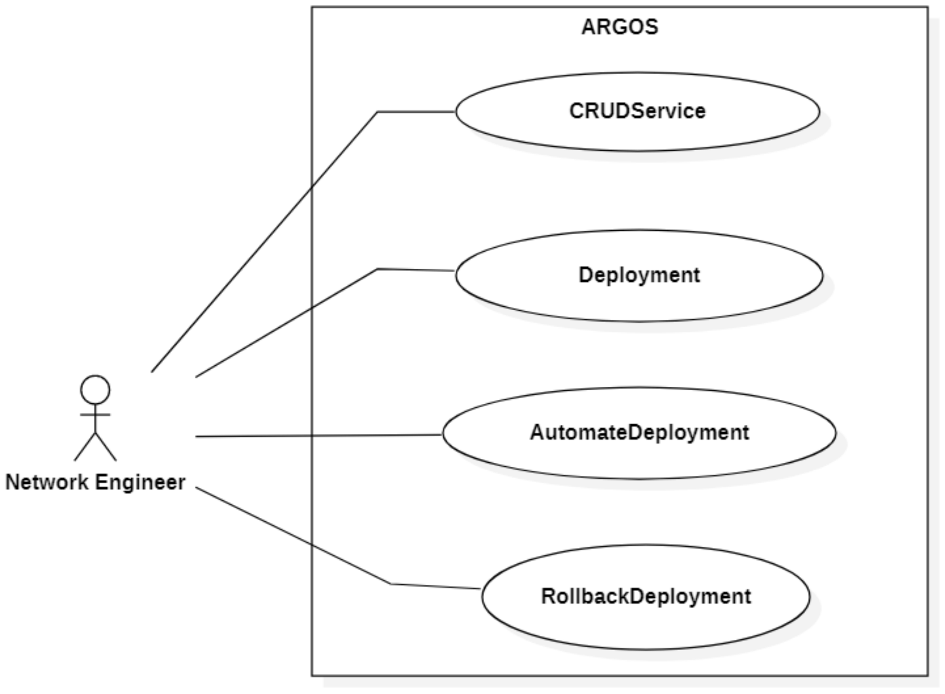


Figure 3.2: Use Cases

3.2.1 Actors

The sole actor in our system is the network engineer, who will document services in NetBox and set triggers for the automatic deployment of these services. In cases where a deployment fails, he can revert a configuration to the previous state by initiating a rollback manually.

3.2.2 Validation

The validation of the functional requirements was done in terms of the System Tests.

3.2.3 Actions

CRUDService

Actor	Network Engineer
Success Scenario	To document the target state of a service in NetBox, a network engineer can add, update, view and delete service-specific settings within the platform. The engineer can do so by leveraging Argos' interface to manage and maintain documentation of the service.
Validation Result	A MPLS L3 VPN service was documented in Argos-NetBox. It is possible to add, update, view and delete objects via the user interface of NetBox.

Table 3.1: Functional Requirement - CRUDService

Deployment

Actor	Network Engineer
Success Scenario	A network engineer wants to deploy the target state as documented in NetBox. He can initiate the deployment process, by issuing a CLI command without requiring any additional interactions. The configuration data stored in NetBox will be used for the deployment.
Validation Result	After documenting the target state in NetBox, it was possible to initiate the deployment procces by issuing a CLI command with Argos-NAC. Argos-NAC generates the configurations out of the saved data in Argos-NetBox and deploys it on the target devices.

Table 3.2: Functional Requirement - Deployment

AutomateDeployment

Actor	Network Engineer
Success Scenario	A network engineer wants to automate the deployment of the target state as documented in NetBox. He can do so by scheduling the deployment based on pre-defined triggers, therefore the deployment process will be initiated automatically.
Validation Result	Automating the deployment can be done, by starting the deployment process in the lstinline—–no-interact— mode, Argos-NAC will not halt at warnings and will proceed without requiring further interactions.

Table 3.3: Functional Requirement - AutomateDeployment

DeprovisionService

Actor	Network Engineer
Success Scenario	A network engineer wants to deprovision a previously configured service. He can do so by deleting the relevant settings from the NetBox interface. The service will be deprovisioned by the next deployment.
Validation Result	To deprovision a previously configured service, a network engineer can first remove or change an object in Argos-NetBox, then start the deployment process with Argos-NAC. The generated configuration will then include commands which will remove or alter service configurations as necessary.

Table 3.4: Functional Requirement - DeprovisionService

3.3 MVP

The Minimum Viable Product (MVP) is a development strategy where the developers create a basic version of the product with the minimum set of features included. In our case the MVP will include the Use Cases **CRUDService** and **Deployment**. In the MVP, the network engineer should be able to add, update, view and delete a specific service in NetBox with Argos. E.g. he should be able to document BGP on multiple devices and interfaces. The engineer should then be able to deploy the target state as documented in NetBox via the network automation controller. The network automation controller will have a pre-defined configuration template, which will be filled with data taken out of NetBox with the help of GraphQL queries. The network automation controller can then be used to deploy these configurations on the network devices.

3.4 Non-Functional Requirements

Interoperability

- ***Argos-NetBox MUST NOT interfere with the normal functioning of NetBox.***
  - **Validation process:**

    Ensure that the NetBox unit tests still pass with Argos-NetBox installed.
  - **Validation results:**

    Since we opted to create our own NetBox plugin instead of submitting a pull request and our code makes no changes to the NetBox internals, there are no interactions between NetBox's unit tests and our code.

Portability

- ***Argos-NetBox SHOULD be installable via a single command, provided the appropriate Python version is pre-installed. In addition, entering the plugin name in the list of installed plugins in the NetBox configuration file enables the plugin upon restart.***
  - **Validation process:**

    The validator will try to install Argos-NetBox with the command `pip install argos-netbox`.

    The validator will try to enable Argos-NetBox by adding `argons_netbox` to the `PLUGINS` list of the netbox configuration.

  - **Validation results:**

    The installation of Argos-NetBox with the command `pip install argos-netbox` was successful, as seen in Figure B.1 "NFR Portability Argos-NetBox Pip Command".

    As seen in Figure B.2 "NFR Portability Argos-NetBox Plugin Configuration", by adding Argos-NetBox to the NetBox configuration file, the plugin is then activated and visible in the user interface. This was also tested as part of the usability test.

- ***Argos-NAC SHOULD be installable via a single command, provided the appropriate Python version is pre-installed.***

  - **Validation process:**

    The validator will try to install Argos-NAC with the command `pip install argos-nac`.

  - **Validation results:**

    The installation of Argos-NAC with the command `pip install argos-nac` is possible, as successfully tested in subsection 10.2.3 "Usability Test Result".

Usability

- ***A network engineer with knowledge of NetBox SHOULD be able to document a simple service within a reasonable time frame.***

  - **Validation process:**

    At the end of the project, the Validator will do a usability test with Urs Baumann, who is our advisor and network engineer, his findings will determine if this NFR is passed or not.

  - **Validation results:**

    The results of the Usability Test can be seen in subsection 10.2.3 "Usability Test Result".

Performance

- ***Within Argos-NetBox, saving a service MUST take the same time approximately, as saving other services outside of our plugin.***

  - **Validation process:**

    The validator will compare the time it takes to save services outside of our plugin with the time measured while saving a service using Argos-NetBox. A maximum deviation of 1 second is deemed as the criteria for passing this NFR.

  - **Validation results:**

    The time required for creating a new device in NetBox is approximately 750 milliseconds, as evidenced by the Figure B.3 "NFR Performance Add Device Time". Similarly, the creation of a new MPLS instance in Argos-NetBox takes approximately 450 milliseconds, as indicated by the Figure B.4 "NFR Performance Add MPLS Instance Time". Given that the deviation is less than one second, this NFR can be considered successfully met.

Testability

- ***Argos-NAC MUST contain tests covering the generation of device configurations.***
  - **Validation process:**

    With the exception of the deployment code, the tests should cover most of Argos-NAC.
  - **Validation results:**

    This requirement has been successfully fulfilled, as the code coverage currently stands at 71 percent during the testing of this NFR, as depicted in the Figure B.5 "NFR Testability Code Coverage".

Reliability

- ***Argos-NetBox SHOULD display and log an error message if it could not save the documented service properly.***
  - **Validation process:**

    The validator will try to input invalid or incomplete data while documenting a service and attempt to save it.
  - **Validation results:**

    Deliberately attempting to enter invalid or incomplete data while documenting a service caused Argos-NetBox to generate errors in the user interface and prompt the user to fill out the necessary fields. The input validation, such as when documenting the LDP range, functioned as expected.

Maintainability

- ***Argos-NetBox SHOULD be documented. With the documentation, the NetBox community should be able to understand the plugin and extend/improve the plugin without the initial project stakeholders.***
  - **Validation process:**

    This requirement is considered passed if all major software components are covered in the technical documentation written in MkDocs. This includes:
    - Argos-NetBox installation instructions
    - Argos-NetBox architecture
    - Argos-NAC installation instructions
    - Argos-NAC broad component overview
    - Argos-NAC architecture

- **Validation results:**

  The technical documentation written in MkDocs covers all significant software components, thus fulfilling this non-functional requirement.

Chapter 4

Architecture

4.1 SSoT Priciples vs. Real World

In a perfect SSoT world, all relevant devices could be configured from the ground up solely based on the contents of its documentation, with complete disregard for the previous state of the devices. One would simply document the ideal state of a device and have that state manifest itself automatically.

This however presupposes that the database acting as the source of truth is capable of documenting every aspect of the configuration of any device, with any software version, of any vendor. The sheer variety of options makes this scenario neigh intractable. Therefore a compromise will likely have to be made.

Additionally, as we will see, the state of the device can itself affect the configuration process when it comes to the IP address with which the device is managed.

This section will cover some of the considerations which lead to the final architecture of Argos-NAC, as well as its limitations.

4.1.1 VRF Deprovisioning

Although deprovisioning was ruled "out of scope" in **??**, its necessity was later rediscovered. The reason for it can be summarized as "consistency without collateral damage".

Suppose a VRF named "Cust_A" with route-distinguisher "65000:1" is configured on a router and documented accordingly in NetBox with ID "1". What happens when the VRF is renamed to "Cust_B" without a deprovisioning mechanism built into Argos-NAC? In NetBox, the name of the VRF entry is simply altered in-place. The ID stays consistent and so does the route-distinguisher. However, attempting to deploy this configuration change to the relevant network devices leads to unexpected behaviour as the existing "Cust_A" VRF and the pending "Cust_B" VRF share the same route-distinguisher, yet neither the network device, nor the deployment tool has any indication that a name change has occurred. From their point of view, the operation is indistinguishable from a deprovisioning of "Cust_A", followed by a provisioning of "Cust_B". Without a deprovisioning mechanism in place, the resulting config would contain two distinct VRFs (both "Cust_A" and "Cust_B") with the same "65000:1" route distinguisher, resulting in undefined behaviour.

While the goal of this project is not to protect a network engineer from his own mistakes, in such a scenario they would be hardly at fault. Without knowledge of the inner workings of Argos-NAC, they would be unaware that this perfectly valid NetBox operation can lead to issues. A deprovisioning mechanism is therefore necessary.

Challenges

The main difficulties of deprovisioning arise from the following thought: "How can I delete what I don't know, without deleting the things I can't re-create?".

In a system with complete SSoT automation coverage, the question is answered by simply being able to re-create everything. One can simply replace the existing configuration with a freshly generated one, without regard for the contents of the previous configuration.

In a system without any SSoT automation coverage, the question is answered by delegating the task to the network engineer. This may however lead to errors in configuration or lack of up-to-date documentation, as configuring and documenting the network are distinct steps.

In a system with partial SSoT automation coverage, things get more complicated. For instance, Cisco routers have no built-in mechanisms for removing all VRF definitions at once, instead requiring each one to be removed individually. This necessitates consulting the current router configuration to determine which VRFs are no longer required. These have to then be removed from the router's configuration without affecting any of the settings not yet covered by Argos-NAC.

Options

There are several options for this procedure, however none are vendor independent.

Using CLI commands to alter the configuration is both vendor specific and difficult to implement safely, since if any error occurs in the chain of commands to execute the state of both the CLI context (Eg. is the terminal still in "(config)" mode?) and of the router as a whole are hard to determine as the possible states vary from command to command.

YANG would be a less vendor dependent technology in principle, however every vendor appears to have come up with their own flavour of it, partly negating this advantage. Additionally, generating verbose XML payloads through tools like Jinja2 is neither pleasant nor easy to debug.

Another option is editing the device's configuration directly. While this is still vendor specific, it allows for all the configuration steps to be joined together into a single, all-or-nothing transaction. Furthermore, Cisco routers allow for features to be disabled by prefixing them with "no" in the configuration. This allows for both the provisioning and deprovisioning of services to happen simultaneously.

Conclusion

To summarize, **deprovisioning of VRFs is an indispensible part of Argos-NAC** as it is necessary to avoid unexpected behaviours and thereby undefined device states. For this purpose, and for the scope of this thesis, we will resort to editing the configurations of the devices, as this appears to be the path with the fewest hurdles, at least when deploying exclusively on Cisco devices.

4.1.2 Management Interface

Some settings are downright impossible to manage well via SSoT. The IP address of a device's management interface, for example, cannot be modified via deployment from an SSoT source due to the following possible scenarios:

- If only the desired new address is documented:
    - The device is unreachable as it is still operating with an old, now unknown IP address.[1]
- If both the old and new address are documented:
    - Issues with temporary double-allocation of IP addresses appear in scenarios where IP addresses are swapped between devices.
    - Should any issues occur in the deployment process, it is neigh impossible to determine whether the IP address change has been applied successfully.
        - Premature discarding of the old IP address leads to the device becoming unreachable, as per the first scenario.
        - Belated discarding of the old IP address may lead to a different device being mistakenly configured in the next deployment, as it may have been re-assigned to a different device in the mean time.
        - Changing the IP address of the management interface terminates the current connection to the device, resulting in the confirmation message not being received by the deploying software. This is the only indication that *something* has changed on the device. However, as this is not the only possible cause of a connection termination, it gives little insight on the success of the deployment, resulting in ambiguity between the states mentioned above.

Due to this, one might come to the reasonable conclusion that the IP address of the management interface is off-limits and is to be left as-is. Cisco's implementation of VRFs however complicates this matter.

As explained in subsection 4.1.1 "VRF Deprovisioning", in the event that a user renames a VRF in NetBox, the previous VRF definition is removed and a new one is added. This requires the "`vrf forwarding`" setting to be updated on all relevant interfaces as well (including the management interface). Cisco routers, however, automatically add "`no ip address`" to the configuration of an interface if its VRF forwarding setting is modified, overwriting any previous settings. This means that, despite being unable to modify it in any meaningful way, **setting the IP address of the management interface is a**

[1]This is the case in the current iteration of NetBox.

must in order to avoid rendering the device unreachable, however **without the ability to change it**.

4.1.3 Interface Configuration Comparison

During development, we considered the option to compare the existing configuration of interfaces against the desired state in order to disable any prior states which may be problematic. In the process, however, we hit two major obstacles: interface aliasing and the lack of terminal symbols in the interface configuration.

Interface aliasing is a feature present in Cisco routers which helps to reduce the amount of typing needed when manually configuring the device by allowing abbreviated interface names. With it, the keywords "g4", "Gi4" and "GigabitEthernet4" all reference the same interface. While this is great for convenience, it's problematic for automation, as the interface name provided by the user in NetBox may or may not be an abbreviated variant of what the router returns when queried. This leads to the issue that comparing the existing configuration of a device against the desired state would require figuring out which *documented* interface name corresponds to which *configured* interface name.

As for terminal symbols, Cisco configurations are very inconsistent in their employment. For example, the `address-family` scope is terminated via `exit-address-family`, whereas the `interface` scope has no terminal symbol and is instead closed either by reaching the end of the file or whenever a command is used which would be invalid in the scope of an interface, which may vary between device models and software versions. The complexity of this task is reflected in the projects attempting to parse these configuration files, such as ciscoconfparse, which has thousands of lines of code dedicated to cisco configurations alone.

Both of these issues could be resolved by individually querying each interface documented in NetBox, as the reply would use the full interface name and only contain the properties of that particular interface, which also resolves the terminal symbol issue. However after some prototyping, we instead decided to forego the comparison of interface configurations entirely in favour of setting "safe defaults" for all supported protocols. For example, if a particular interface has no VRF assigned to it in the documentation, "`no vrf forwarding`" is added to the configuration, overwriting any previous VRF setting, if present.

4.2 Plugin vs. Pull Request

As per the assignment, we had the choice between implementing a standalone plugin or extending NetBox directly through a pull request. The latter, however, comes with some caveats.

Firstly, the NetBox project requires all pull requests to have an associated issue which is accepted and assigned to the author of the pull request. Should these conditions be unmet, the pull request is automatically closed.[9] This would require us negotiating our vision with the project's maintainers ahead of time or risk sinking considerable time into a proof of concept, without any guarantees of approval.

Second, working on an already established open-source project comes with the additional requirement that it needs to be interoperable with other plugins in the ecosystem. This would be especially challenging with plugins such as netbox-bgp, which already covers many of the models our contribution would need to implement, but without some of the strict checks necessary to reliably generate a valid configuration.

In our opinion, these complications are not easily resolved and may lead to considerable delays in the project. For this reason, we have opted to develop a plugin instead.

4.3 Data Structure

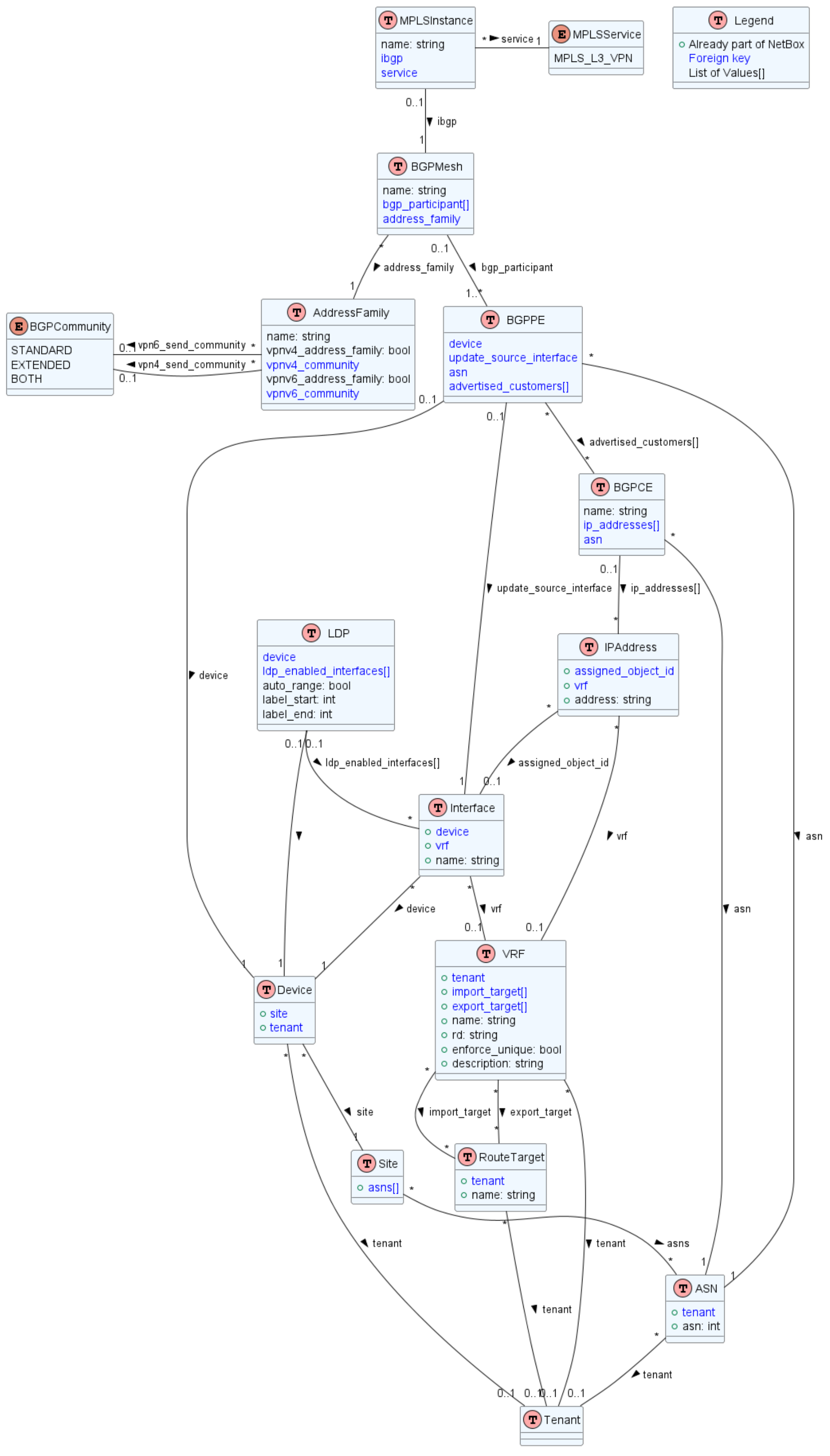


Figure 4.1: Data Structure Overview

Figure 4.1 shows how the various pieces of information required for configuring an MPLS L3 VPN are stored and how they relate to each other. This structure was arrived at while considering the following requirements:

- Protocols which can function standalone or in conjunction with protocols other than the ones covered by Argos should be stored independently, thereby avoiding interdependencies which would only apply in narrow cases.
- Bottom-up instantiation is preferable. For example, this allows participants of a network to exist on their own, without the network needing to be documented ahead of time.
- Future expansion should be facilitated, where possible. For instance, we opted to only implement BGP-mesh configurations, where every member is connected to all others. This mesh network is kept as a standalone table in order to allow drop-in replacements if, for example, route reflectors are used instead.
- Relationships should be considered redundant if the same information can be extrapolated through data-structure traversal.

4.3.1 Device - ASN Relationship

During the early stages of development, we discovered that NetBox's "Device" model cannot be directly associated with an "ASN" instance. Instead, they can only be indirectly related via the "Site" model. Unfortunately, a Site instance can hold a list of ASNs, thereby creating ambiguity as to which device is associated with which ASN. For the purposes of BGP, this ambiguity was not an option.

This problem has been raised as part of issue #8782[5], but was largely dismissed by the maintainers of the project, who instead suggest using a plugin or a "custom object field" for this relationship - neither of which we find particularly enticing options. The underlying cause of the maintainers' reluctance is that NetBox attempts to not make assumptions about users' infrastructure, thereby favouring not implementing something over implementing something which may preclude a particular use case. This, combined with real-world configurations and topologies which sometimes verge on the esoteric, can lead to more common use cases needing to be neglected.

To circumvent NetBox's limitation and avoid possible conflicts with other plugins, the decision was made to add a relation from the BGPCE and BGPPE models to ASN. This is in no way an ideal solution, but it allows us to implement BGP without ambiguities.

4.4 Model Class Diagram

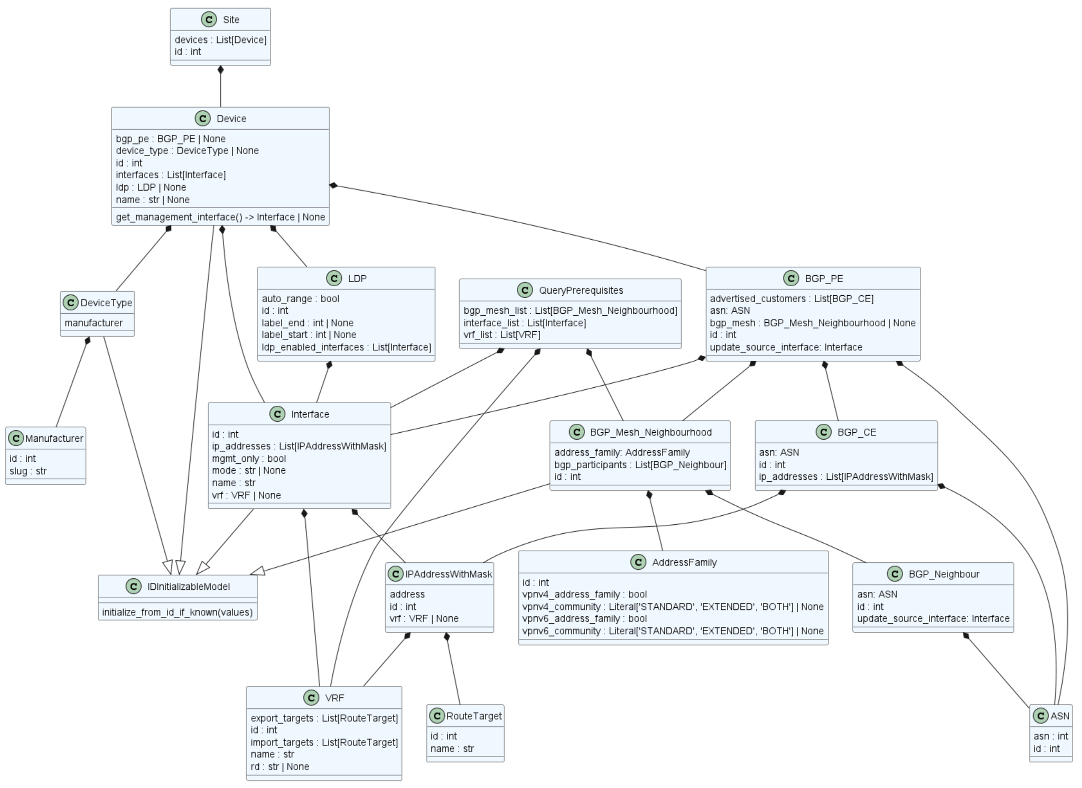

Site
devices : List[Device]
id : int
Device
bgp_pe : BGP_PE | None
device_type : DeviceType | None
id : int
interfaces : List[Interface]
ldp : LDP | None
name : str | None
get_management_interface() -> Interface | None
DeviceType
manufacturer
LDP
auto_range : bool
id : int
label_end : int | None
label_start : int | None
ldp_enabled_interfaces : List[Interface]
QueryPrerequisites
bgp_mesh_list : List[BGP_Mesh_Neighbourhood]
interface_list : List[Interface]
vrf_list : List[VRF]
BGP_PE
advertised_customers : List[BGP_CE]
asn: ASN
bgp_mesh : BGP_Mesh_Neighbourhood | None
id : int
update_source_interface: Interface
Manufacturer
id : int
slug : str
Interface
id : int
ip_addresses : List[IPAddressWithMask]
mgmt_only : bool
mode : str | None
name : str
vrf : VRF | None
BGP_Mesh_Neighbourhood
address_family: AddressFamily
bgp_participants : List[BGP_Neighbour]
id : int
BGP_CE
asn: ASN
id : int
ip_addresses : List[IPAddressWithMask]
IDInitializableModel
initialize_from_id_if_known(values)
IPAddressWithMask
address
id : int
vrf : VRF | None
AddressFamily
id : int
vpnv4_address_family : bool
vpnv4_community : Literal['STANDARD', 'EXTENDED', 'BOTH'] | None
vpnv6_address_family : bool
vpnv6_community : Literal['STANDARD', 'EXTENDED', 'BOTH'] | None
BGP_Neighbour
asn: ASN
id : int
update_source_interface: Interface
VRF
export_targets : List[RouteTarget]
id : int
import_targets : List[RouteTarget]
name : str
rd : str | None
RouteTarget
id : int
name : str
ASN
asn : int
id : int

Figure 4.2: Representation of Data Structure in Python

While the data structure shown in Figure 4.2 is similar to the one shown in Figure 4.1, there are some variations in the relationships between the classes. The reason for this is simplicity.

While a relationship in a database can be queried in either direction, this becomes more challenging when translated to a Python class structure which natively only supports parent-to-child traversal. Turning this strict parent-child relationship into a less hierarchical one would also require great care to be taken during initialization, as initialization would also need to be supported bi-directionally, but without causing recursive loops. While this is certainly achievable and may eventually be required if the scope and complexity of the model increases drastically, this is not currently necessary.

For these reasons, the data structures of the database and its Python counterpart intentionally deviate from each other, with the translation between them being done through the GraphQL query itself.

4.5 Component Overview

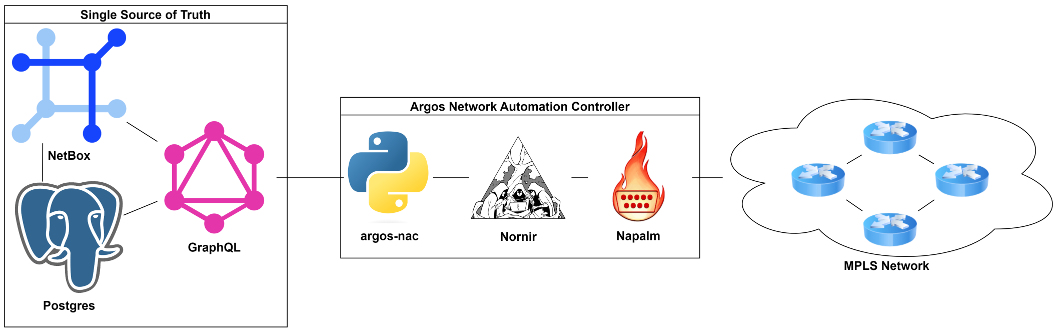


Figure 4.3: Component Overview

Chapter 5

Results

In this thesis, we managed to create Argos-NetBox, a standalone NetBox plugin capable of documenting an MPLS L3 VPN which, while limited in scope, is made to be extendable. The included protocol models are kept independent wherever possible and attempt .

Argos-NAC, on the other hand, can deploy and deprovision all the documentable properties Argos-NetBox provides. It aims to treat the NetBox database as a Single Source of Truth wherever possible and can easily be extended to support additional vendors. All this while attempting to keep the user experience as simple and straight-forward as possible.

Future developers can use this project as a starting point, adding different topologies, new protocols and re-combine them into new services. The MkDocs documentation provided alongside should help promote such endeavors.

Chapter 6

Limitations

6.1 DHCP Incompatibility

As it stands, this project is fundamentally incompatible with DHCP. This is due to the following reasons:

- at least philosophically, DHCP breaks the SSoT principle unless the DHCP server and the network documentation are part of the same process. Since NetBox does not check this box natively, support for DHCP is not conducive to the goals of this project.
- this project relies on every device to configure having a management interface with a known IP address. Without this, it would be impossible to target a machine during the configuration process. Therefore, at least the network used for management cannot use DHCP.
- the way BGP is implemented in the Cisco configuration, the IP address of neighbouring provider edges have to bee known when generating the configuration. While this could be determined by querying the device itself for DHCP-sourced IP addresses on non-management interfaces, this breaks the SSoT ethos by treating the device state as an additional source of truth.

While these issues may be resolved by loosening the adherence to the SSoT ethos, this runs counter to the goal of this thesis. As such, we believe it is up to the future maintainers to weigh the advantages and disadvantages of relaxing the SSoT principle.

6.2 Single Source of Truth

While we tried to follow the SSoT principle to the letter where possible, developing Argos-NAC to work with Cisco devices demonstrated the difficulty therein. Without the ability to generate the entire device configuration from scratch and the limited ability to deprovision the entire configuration related to a particular protocol, we believe compromises have to be made. Our approach was to query the existing device configuration in order to find the names and IDs of the components which need deprovisioning. This does break the SSoT ethos, however it was the only approach we could find which worked in all cases and did not add significant complexity to the project.

Chapter 7

Further Work

7.1 Work to be Done Within Argos

The bulk of the work to be done within Argos-NetBox concerns expanding on the features it provides. This may include new protocols and services and new BGP topologies (potentially including route reflectors). In addition, we believe adding a way to visualise the network service and topology through graphs would help in keeping an overview in an otherwise fractured set of user interfaces.

For Argos-NAC, many of the same points apply, with the addition of more vendors, which would allow for mixed-vendor networks to be deployed. Additionally, a better mechanism for determine which aspects of the device configuration require deprovisioning would be recommended, as the current iteration breaks the SSoT ethos.

7.2 Proposed Work in the Ecosystem

7.2.1 Addition of OneToMany to Django

Django (and by extension NetBox) has several different relationship classes which provide more fine-grained control over the multiplicity between table entries, for instance `ManyToMany` can be used where many individual objects may relate to many other objects in a non-exclusive manner. To our great surprise however, a `OneToMany` type does not exist as an option. This, despite Django clearly having carve-outs intended for it, namely a `one_to_many` flag set to `False` inside each of the different relationship types.

The suggestions typically floated whenever this apparently missing feature is brought up are twofold: either "use `ManyToMany`", which does not play nicely with GraphQL out of the box, as we will see shortly, or "use `ForeignKey`", which requires the relation to be established in a particular direction.

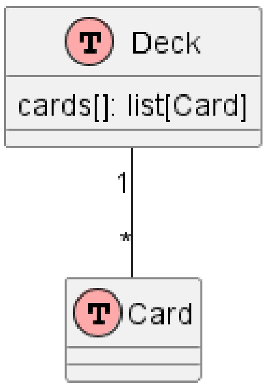


Figure 7.1: OneToMany Example

Suppose there are many cards, each of which can only be part of a single deck, as seen in Figure 7.1. The reasons why someone would want to establish the relationship from the "deck" side side may vary, for example it could be done to facilitate a better overview of which cards are part of the same deck in the UI. This can be implemented fairly well with a `ManyToMany` reference type in addition to filters to ensure the exclusivity of the relationship.

When Querying such a relationship from the "Many" side (ergo starting from one of the individual cards being referenced and looking up which deck it belongs to), the resulting JSON however defaults to a list of objects, even through in a proper "one to many" relation there can only be at most a single object at the other end. This requires messing with custom GraphQL serializers to avoid these implementation details affecting the structure of the output data.

Proposals for a `OneToMany` relation have been made for over a decade[6], so the chances of it being added anytime soon are slim. Still, we believe it would be a worthwhile addition.

7.2.2 NetBox Permissions Rework

While testing Argos-NetBox after the construction phase, we noticed that, while the plugin worked for the super-admin account, it would only partly work when used by a regular user with all permissions unlocked. The reason for this is the way NetBox parses and stores permissions, namely strings taking the form `<app_label>.<action>_<model>`, which is then parsed using the following code:

```python
def resolve_permission(name):
    try:
        app_label, codename = name.split('.')
        action, model_name = codename.rsplit('_', 1)
    except ValueError:
        ...
```

This however is rather problematic when considering that models (which are effectively regular python classes) may contain underscores in their names. This would result in wrongful splitting of the permission string. For example, `my_plugin.add_bgp_pe` would be erroneously split into `app_label="my_plugin"`, `action="add_bgp"` and `model_name="pe"`.

This specific scenario could be fixed by using `.split()` instead of `.rsplit()`, making the split at the first instead of last underscore, however this is not an option as compound actions such as `bulk_edit` and `bulk_delete`. And while these may yet be accounted for by using regular expressions, NetBox also support custom actions to be added by plugins, which completely rules out this workaround.

These circumstances currently prevent any model containing underscores in its name from properly functioning for non super-admins. While this may not be an issue for NetBox itself since the project follows strict PEP8 guidelines which recommends against underscores in class names, it does restrict third-party plugins unnecessarily and, at least currently, in a problematic way, as the permission checks fail silently.

This issue was reported in issue #12809 on the NetBox repository.[2]

While nothing is more divisive than people's naming convention preferences, we are of the opinion that PEP8 is not appropriate for long acronyms due to its ALL-CAPS, no-demarcations approach to acronyms. In cases like "MPLS_L3_VPN", removing the underscores does not promote readability and expanding the acronym leads to an unnecessarily long model name. We would therefore like to see the permission system to be reworked.

Part III

Project Documentation

Chapter 8

Project Plan

8.1 Organization

In typical projects, the costs, timeline and scope of a project have to be considered and balanced out, this is typically referred to as the "project management triangle". For our purposes however, the final deadline is non-negotiable and the costs are non-existent. This means our scope is limited to whatever can be achieved within the given time-frame, making artifacts like Gantt charts and milestone-trend analysis rather pointless for planning purposes. We instead opted to subdivide our allotted time into phases, akin to RUP, and use a simple kanban board to keep track of outstanding tasks and assign the associated responsibilities. This keeps the organizational overhead to a minimum, leaving more time for productive work.

8.2 Project Time Plan

8.2.1 Phases

Inception

The inception phase starts on 20.02.2023 and will end on 10.03.2023. In this phase, we make ourselves familiar with NetBox and the technologies/protocols surrounding the Layer 3 and Layer 2 services. Reaching a consensus on the project scope and requirements with the stakeholder is another goal of this phase.

Elaboration

The elaboration phase starts on 11.03.2023 and lasts till 31.03.2023. In this phase, we define what our risks, our technical requirements and NFRs are. We take a decision on whether the project will entail a NetBox plugin or a contribution to the NetBox project itself, and evaluate the feasibility of a data-driven approach. We also set up our developing environment in preparation for the construction phase.

Construction

The construction phase starts on 01.04.2023 and ends on 02.06.2023, in this time span the bulk of the implementation takes place.

Transition

This phase lasts the last week before the final hand-in of the project from 03.06.2023 to 16.06.2023 17:00. In this phase we finalize the documentation and prepare it for the final hand-in.

8.2.2 Milestones

M1: End of Inception - 10.03.2023

The goal of this milestone is to understand the assignment and get a grasp of NetBox and the corresponding technologies/protocols for the Layer 3 and Layer 2 services. A project time plan is created.

Deliverables:

- Research NetBox and Layer 2/3 Services
- Project time plan

M2: End of Elaboration - 31.03.2023

At the end of the elaboration the decision should have been made, if our end product is gonna be a plugin or a PR and the feasibility of a data-driven plugin. The requirements, non-functional requirements and a risk analysis are defined by the end of this phase.

Deliverables:

- Risk analysis
- Functional and non-functional requirements
- Use cases
- Feasibility of data-driven approach

M3: Prototype - 14.04.2023

This milestone represents an important stage in the project where the goal is to validate the concept by prototyping.

Deliverables:

- NetBox plugin/integration prototype
- Network automation tool prototype

M4: MVP - 12.05.2023

The goal of the MVP milestone is to create a basic but functional cycle that meets all minimum requirements of the project. This MVP will serve as a foundation for further development.

Deliverables:

- MVP prototype (minimum viable product)
- MVP presentation to stakeholders

M5: End of Construction - 02.06.2023

The project is functional and complete, only small adjustments and parts of the documentation are left to be finished.

Deliverables:

- Test protocol
- Usability Test
- Working plugin
- Working Network Automation Controller
- Abstract (due 12.06.2023)

M6: End of Transition - 16.06.2023 17:00

Everything should be finished and the project should be handed-in with the complete documentation. The poster is created according to the guidelines.

Deliverables:

- Finished poster
- Finished project documentation
- Finished product

M7: Presentation - 21.07.2023

The presentation for the bachelor thesis is prepared and the team is ready to present it.

Deliverables:

- Presentation

8.3 Roles

Advisor and Product Owner

Urs Baumann is partly responsible for the evaluation of the project and is simultaneously the product owner. He is available for any help of the project if the team is facing any sort of issues.

Developers

Dejan Jovicic and Dominic Walther will be the developers and responsible for the success of the project.

Additional Stakeholders

The additional stakeholders for our project are Mitra Purandare as the internal co-examiner and Patrick Mosimann, who works as a Technical Solutions Architect for Cisco, as the external co-examiner.

8.4 Meetings

Every friday a meeting will be held with the advisor and stakeholder Urs Baumann and the two developers. The meeting minutes can be found in ?? "??".

8.5 Tools

- clockify.me (Time tracker)
- Gitlab (Issue tracker, file storage & versioning)
- Microsoft Teams (Communication channel & meeting)
- Visual Studio Code (Used to write the documentation in LaTeX)
- Docker (Development)
- NetBox (Plugin)
- LTB Lab from the Institute for Networked Solutions (Virtual network used for development/testing)

Chapter 9

Risk Management

9.1 Risks

This chapter deals with the risks that might occur during the project. The properties of the risks are:

- ID: Identifier of the risk.
- Description: Short text explaining the risk.
- Probability: The probability of the risk measured in percentage. 100% meaning that it will occur at least once during the project.
- Maximum Time Loss: The maximum time loss we will have because of this risk, measured in hours
- Mitigation: Description of the precautions we can take to mitigate the risk.
- Behaviour: Describes the behaviour of what is done if the risk happens.
- Severity: The severity shows the extent of the damage the risk will do to the goals and objectives to the project. It is measured qualitatively with "low", "mid" and "high" severity.

ID	Description	Probability	Max. time loss	Mitigation	Behaviour	Severity
R1	A team member gets sick	low	23h	No mitigation possible	Communication between team members, prioritize tasks which need to be done	low
R2	Irreparable corruption of the git server	low	46h	Weekly off-site backups on the devices of the team members	Restore data with your backup	mid
R3	Irreparable corruption of the clockify data	low	0.5h	Weekly off-site backups on the devices of the team members	Restore data with your backup	low
R4	Unrealistic timeline	low	46h	Detailed project plan	Reduction of the project scope	mid
R5	Specifications cannot be implemented as intended	mid	69h	Weekly meetings with the advisor	Getting help from the advisory & experts	mid
R6	Familiarization with NetBox, network services and network automation tools takes longer than expected	mid	46h	Watch tutorials to gain knowledge of technology	Reduction of the project scope	mid
R7	An external dependency does not meet our requirements	high	46h	Isolate and compartmentalize dependencies within the project to minimize affected surface area	Conduct assessments to ensure that the external dependencies meet the desired functionality and scalability requirements, before integrating them into the project	mid

Table 9.1: Risk Analysis

9.2 Realized Risks

ID	Description	Behaviour	Impact	Date	RID
RR1	Dejan was ill for several days	Dominic was instructed which pending and important tasks need to be done till the next sprint	23h	29.03.23	R1
RR2	The NetBox API does not fit our requirements for requesting data	Use GraphQL instead of the NetBox API	46h	31.03.23	R7

Table 9.2: Realized Risks Analysis

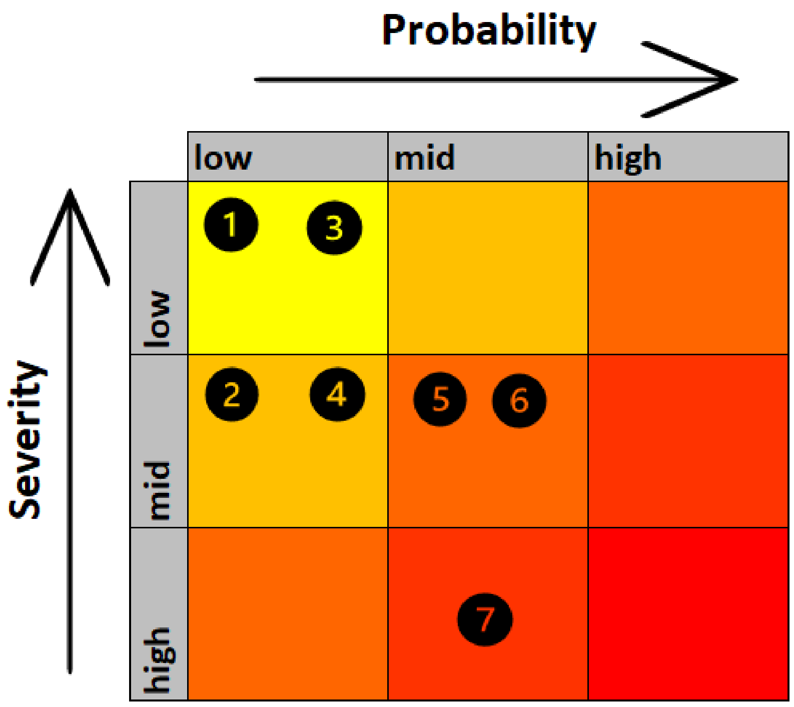


Figure 9.1: Risk Matrix

Chapter 10

Quality Measures

10.1 Code Guidelines

The principles described in this section should always be adhered to to ensure a consistent code-style. Change proposals are always welcome.

Editor: The editor to be used is VSCode. Files associated with said editor can be committed to the repository provided the contained settings are not user-dependent. The recommended plugins should be installed.

Linter: The linter specified in the settings file should be installed and enabled at all times with the provided settings. Exceptions to the linting rules should be avoided. Linting rules may be changed if an agreement is found.

Classes: Class names are written in *CamelCase*. Member fields are annotated with their expected type. Should annotation not be possible the regular way (e.g. due to circular imports), annotations are done via so-called "forward references".

Function Names: Function names are written in *snake_case*.

Variable Names: Variable names are written in *snake_case*. Too explicit is better than too short. Abbreviations are only allowed if they are immediately obvious in their context.

Comments: Comments should primarily explain why something was done, not how it works. Comments explaining how something works are to be treated as a code-smell.

Comprehensions: Comprehensions should only be used for trivially understandable operations. Use explicit for-loops otherwise.

Exception handling: Exceptions are to be treated as errors and should only be raised if the program cannot proceed safely. Exceptions should not be caught unless absolutely necessary. Exceptions may be caught and re-thrown to add additional context to the exception.

10.2 Testing

10.2.1 Code Coverage

Due to the number of involved technologies and components, as well as the difficulty of testing the automated deployment aspect of the project, reaching a high degree of code coverage through automated tests is a challenge in and of itself. Attempts will be made to facilitate testability through strategies like dependency injection where possible, however automated testing will remain a best-effort goal.

10.2.2 Usability Tests

The usability testing will be conducted by Urs Baumann, who serves as both our advisor and the product owner. With his expertise in NetBox and configuring network devices, he is the ideal candidate to test Argos-NetBox. Urs Bauman will receive a zip file containing the project files, along with a Word document outlining the tasks to be done and fields for providing feedback.

The usability test is scheduled to take place after the construction phase, allowing us ample time to address any minor imperfections and enhance the provided documentation, if necessary. The Usability Test Protocol and the filled out Word document can be found in the section B.2 ”Usability Test Protocol Result” for review.

Goal

The objective of this usability test is to assess the usability and user-friendliness of Argos-NetBox, ensuring its alignment with the needs and expectations of our target audience. Additionally, we seek valuable feedback on Argos-NAC, which will guide us in enhancing its usability if necessary.

10.2.3 Usability Test Result

As described in subsection 10.2.2 ”Usability Tests”, our usability tester for this project was Urs Baumann, a senior network engineer working for the INS. The completed Word document can be accessed at section B.2 ”Usability Test Protocol Result”.

Result

During the usability test, Urs Baumann encountered some difficulties, primarily related to his computer setup, as stated in his feedback. In step 2 of the NetBox tasks, he expressed confusion regarding the concept of ”Address Families”, he elaboreated on why he believed the term could be misunderstood in the Open-Ended Questions. However, he also mentioned that he could not propose a better alternative name and acknowledged that renaming might not be necessary. In the Network Automation Controller task 1, he noted that he had received an incorrect command from us, as no output folder was created. However, he was able to find the correct option by using the help command.

In his general feedback, Urs suggested that including a brief description of the object in the ”Add” interface in Argos-NetBox would be helpful, or alternatively, linking the ”Help” button to our documentation on these objects.

Conclusion

We agree with our tester's recommendation to provide additional information about the objects in Argos-NetBox. As a result, we have updated the "Add" user interface to include a direct link to our MkDocs documentation.

Based on the overall feedback from the usability test, we consider the NFR Usability to be successfully met, as the tester rated the overall user experience as "very well."

10.2.4 CI/CD Pipeline

To avoid "it works on my machine" scenarios, we added a CI/CD Pipeline for the Argos-NAC source code, as well as the documentation.

The documentation CI pipeline consists of two stages: "build_doc", which generates a PDF out of the latest LaTeX-files and "build_doc_diff", which builds a PDF in which the changes that were made since the last meeting are highlighted. This is especially helpful during meetings, as it makes visualizing progress trivial. The latest versions of these documents are downloadable from the GitLab repository by clicking on the relevant badges.

The Argos-NAC source code CI/CD contains the "run_pytest" stage, which runs Pytest and generates a code coverage report. If all tests pass, a second "deploy" stage can be triggered which takes said coverage report, merges it into the MkDocs wiki and host both on "GitLab Pages", allowing users to interactively navigate the source code with added coverage information, in addition to reviewing the project's documentation. This comes with the added benefit that the documentation and the source code can easily be synchronized to each other.

10.2.5 System Tests

The system testing was done with the LTB Lab, which was set up for us by Urs Baumann and hosted by the Institute for Networked Solutions. Argos-NetBox is visible in NetBox as seen in Figure 10.1 and Figure 10.2 where we can document a Multiprotocol Label Switching Layer 3 Virtual Private Network (MPLS L3 VPN) network.

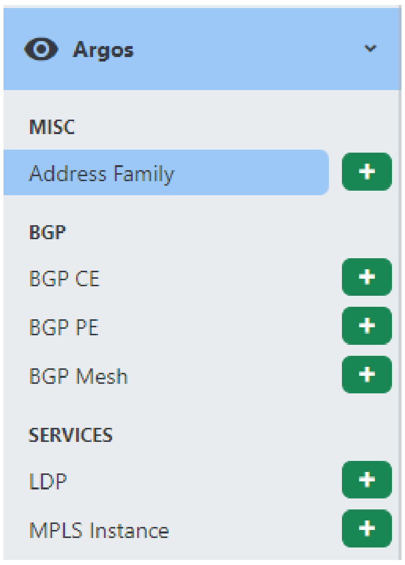
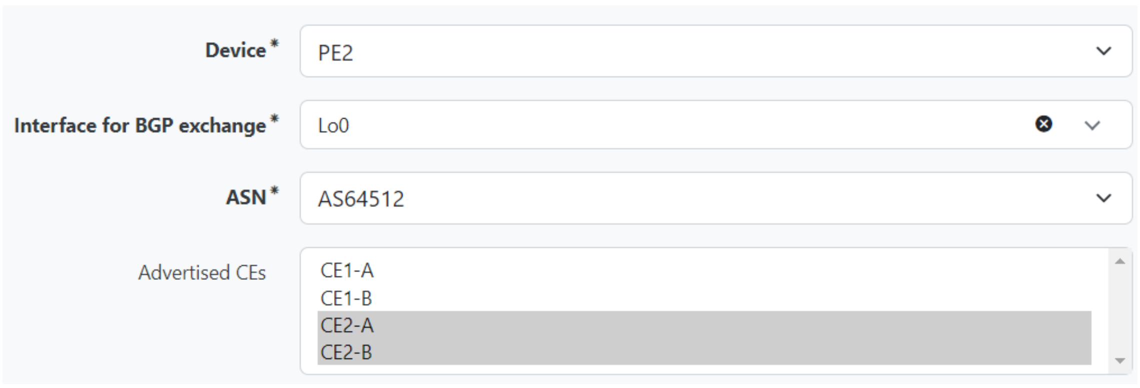


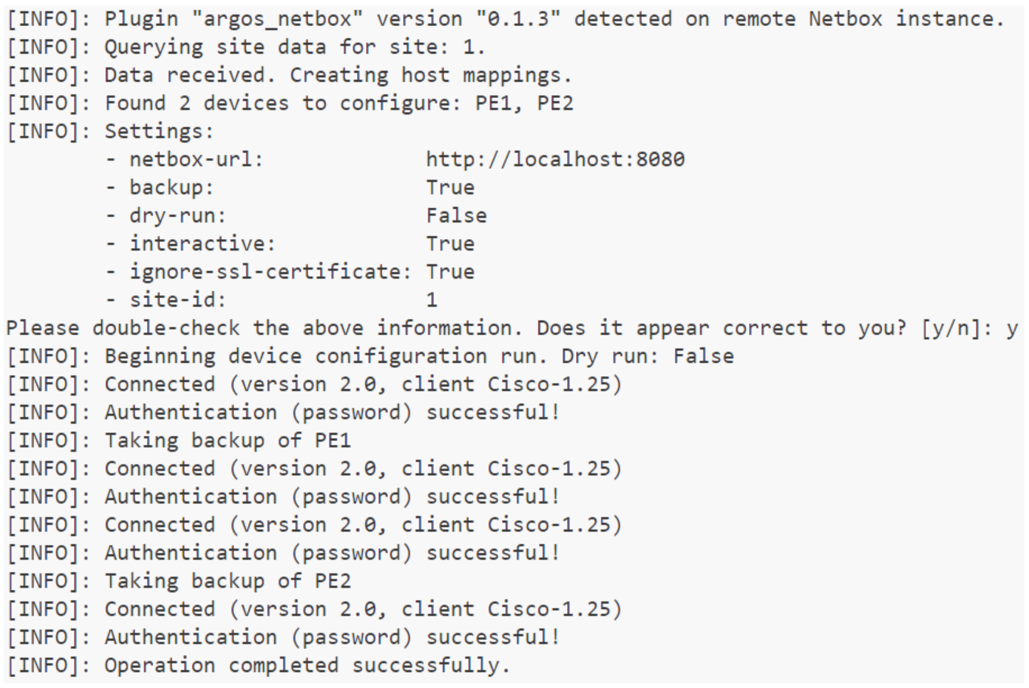
Figure 10.1: Argos-NetBox Overview



Device *	PE2
Interface for BGP exchange *	Lo0
ASN *	AS64512
Advertised CEs	CE1-A CE1-B CE2-A CE2-B

Figure 10.2: Argos-NetBox BGP PE Add

After documenting a MPLS L3 VPN network in Argos-NetBox, we then can deploy this state to the network devices by using Argos-NAC. This can be done via the CLI using the command `argos-deploy --ignore-ssl --file-output --backup --verbose`, the output of which can be seen in Figure 10.3.

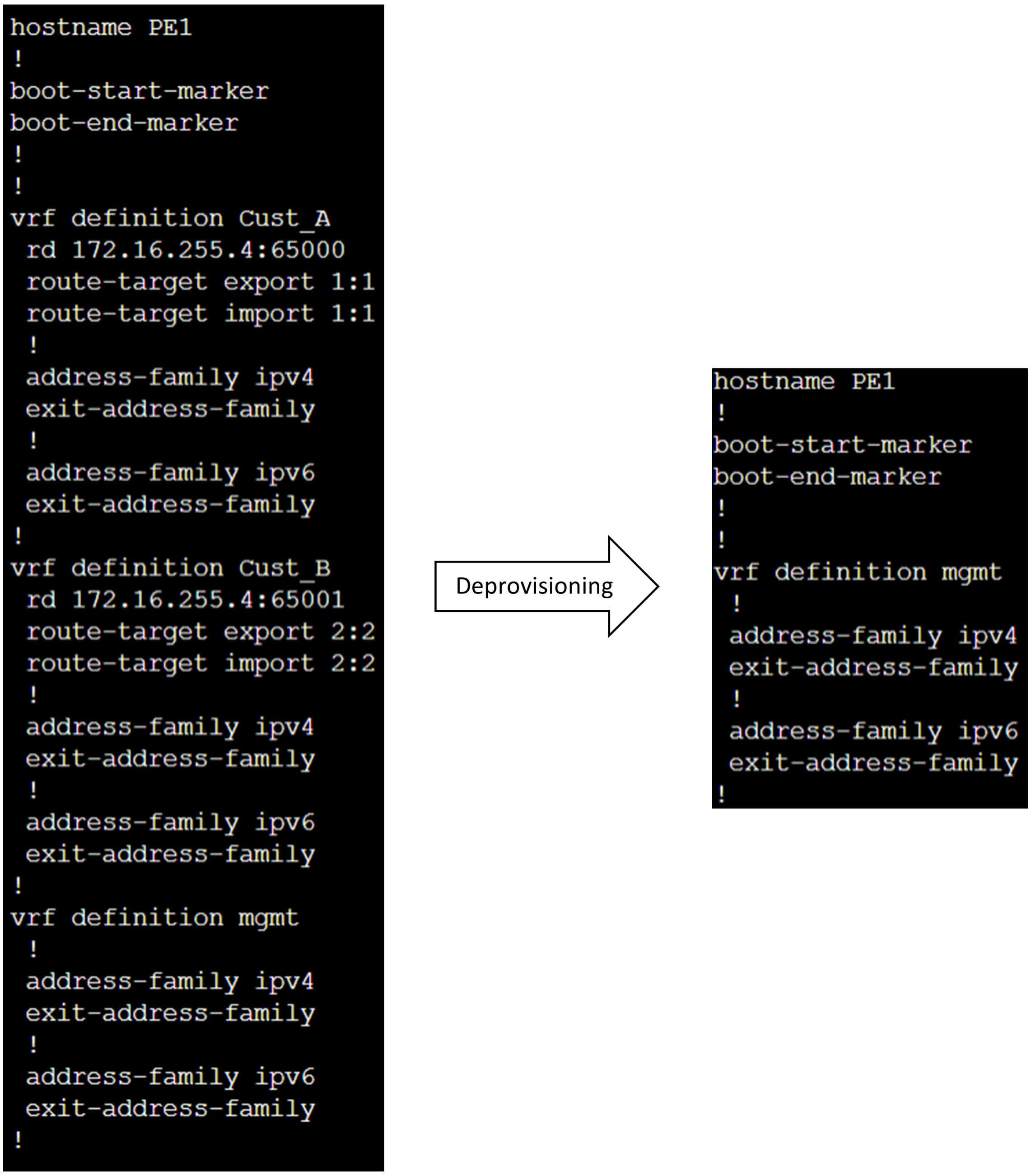


```
[INFO]: Plugin "argos_netbox" version "0.1.3" detected on remote Netbox instance.
[INFO]: Querying site data for site: 1.
[INFO]: Data received. Creating host mappings.
[INFO]: Found 2 devices to configure: PE1, PE2
[INFO]: Settings:
        - netbox-url:             http://localhost:8080
        - backup:                 True
        - dry-run:                False
        - interactive:            True
        - ignore-ssl-certificate: True
        - site-id:                1
Please double-check the above information. Does it appear correct to you? [y/n]: y
[INFO]: Beginning device conifiguration run. Dry run: False
[INFO]: Connected (version 2.0, client Cisco-1.25)
[INFO]: Authentication (password) successful!
[INFO]: Taking backup of PE1
[INFO]: Connected (version 2.0, client Cisco-1.25)
[INFO]: Authentication (password) successful!
[INFO]: Connected (version 2.0, client Cisco-1.25)
[INFO]: Authentication (password) successful!
[INFO]: Taking backup of PE2
[INFO]: Connected (version 2.0, client Cisco-1.25)
[INFO]: Authentication (password) successful!
[INFO]: Operation completed successfully.
```

Figure 10.3: Argos-NAC Deploy

To automate this process, we can start Argos-NAC in the `--no-interact` mode, allowing it to proceed without any additional interactions including ignoring warnings, if enabled. The scheduling aspect can be managed using established tools like cron or similar scheduling mechanisms.

Likewise, the deprovisioning of services can be accomplished by removing objects within Argos-NetBox, before running Argos-NAC. Argos-NAC will then compare the current device configurations with the documented state in NetBox and automatically generate the necessary deprovisioning commands. The end result can be seen Figure 10.4



```
hostname PE1
!
boot-start-marker
boot-end-marker
!
!
vrf definition Cust_A
 rd 172.16.255.4:65000
 route-target export 1:1
 route-target import 1:1
 !
 address-family ipv4
 exit-address-family
 !
 address-family ipv6
 exit-address-family
!
vrf definition Cust_B
 rd 172.16.255.4:65001
 route-target export 2:2
 route-target import 2:2
 !
 address-family ipv4
 exit-address-family
 !
 address-family ipv6
 exit-address-family
!
vrf definition mgmt
 !
 address-family ipv4
 exit-address-family
 !
 address-family ipv6
 exit-address-family
!
```

```
hostname PE1
!
boot-start-marker
boot-end-marker
!
!
vrf definition mgmt
 !
 address-family ipv4
 exit-address-family
 !
 address-family ipv6
 exit-address-family
!
```

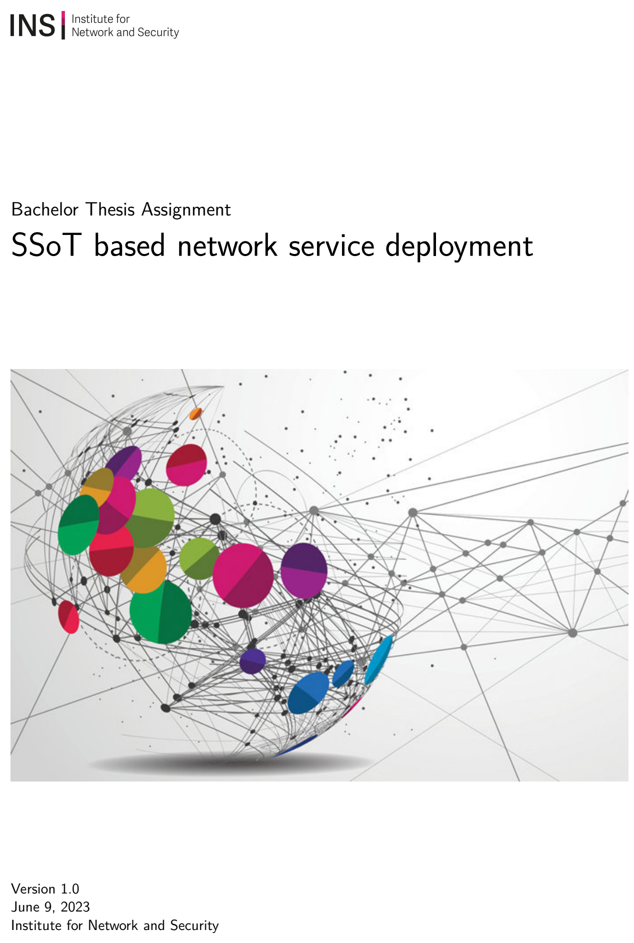
Figure 10.4: Argos-NAC VRFs

Part IV

Appendix

Appendix A

Assignment



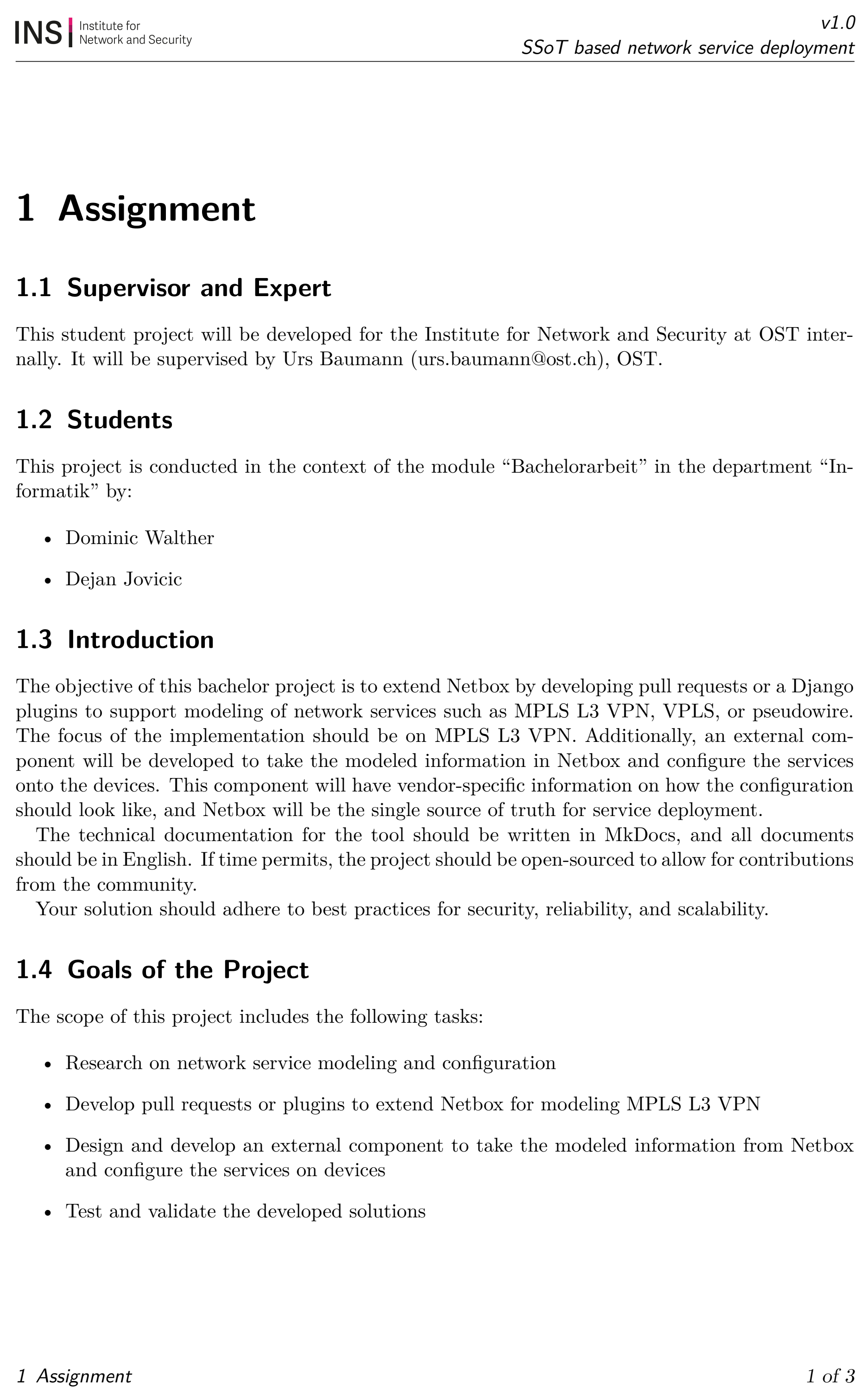
INS | Institute for Network and Security

Bachelor Thesis Assignment

SSoT based network service deployment

Version 1.0
June 9, 2023
Institute for Network and Security

Figure A.1: Bachelor Thesis Assignment Page 1



INS | Institute for Network and Security

v1.0
SSoT based network service deployment

1 Assignment

1.1 Supervisor and Expert

This student project will be developed for the Institute for Network and Security at OST internally. It will be supervised by Urs Baumann (urs.baumann@ost.ch), OST.

1.2 Students

This project is conducted in the context of the module “Bachelorarbeit” in the department “Informatik” by:

- Dominic Walther
- Dejan Jovicic

1.3 Introduction

The objective of this bachelor project is to extend Netbox by developing pull requests or a Django plugins to support modeling of network services such as MPLS L3 VPN, VPLS, or pseudowire. The focus of the implementation should be on MPLS L3 VPN. Additionally, an external component will be developed to take the modeled information in Netbox and configure the services onto the devices. This component will have vendor-specific information on how the configuration should look like, and Netbox will be the single source of truth for service deployment.

The technical documentation for the tool should be written in MkDocs, and all documents should be in English. If time permits, the project should be open-sourced to allow for contributions from the community.

Your solution should adhere to best practices for security, reliability, and scalability.

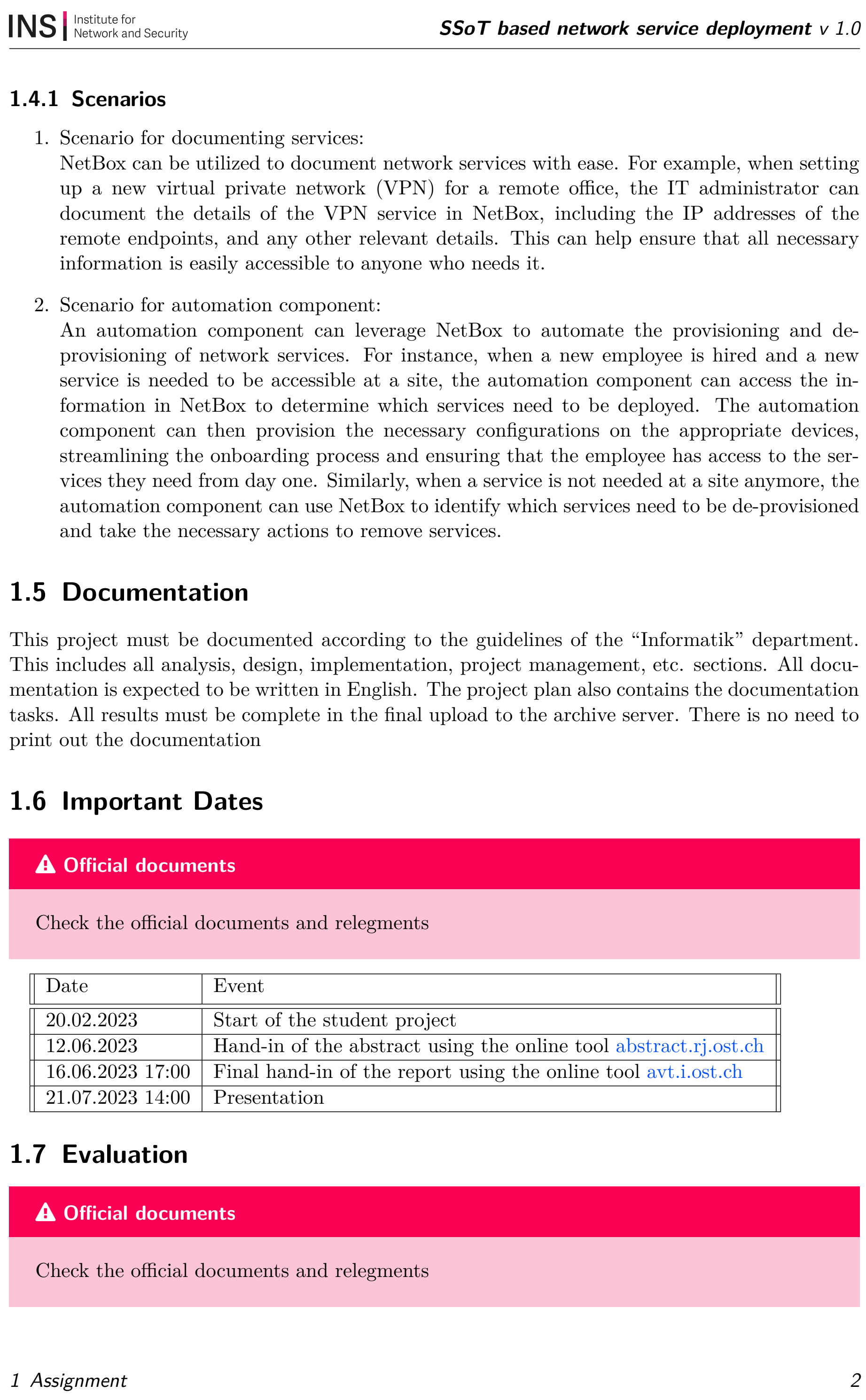
1.4 Goals of the Project

The scope of this project includes the following tasks:

- Research on network service modeling and configuration
- Develop pull requests or plugins to extend Netbox for modeling MPLS L3 VPN
- Design and develop an external component to take the modeled information from Netbox and configure the services on devices
- Test and validate the developed solutions

1 Assignment *1 of 3*

Figure A.2: Bachelor Thesis Assignment Page 2



1.4.1 Scenarios

1. Scenario for documenting services:
   NetBox can be utilized to document network services with ease. For example, when setting up a new virtual private network (VPN) for a remote office, the IT administrator can document the details of the VPN service in NetBox, including the IP addresses of the remote endpoints, and any other relevant details. This can help ensure that all necessary information is easily accessible to anyone who needs it.
2. Scenario for automation component:
   An automation component can leverage NetBox to automate the provisioning and deprovisioning of network services. For instance, when a new employee is hired and a new service is needed to be accessible at a site, the automation component can access the information in NetBox to determine which services need to be deployed. The automation component can then provision the necessary configurations on the appropriate devices, streamlining the onboarding process and ensuring that the employee has access to the services they need from day one. Similarly, when a service is not needed at a site anymore, the automation component can use NetBox to identify which services need to be de-provisioned and take the necessary actions to remove services.

1.5 Documentation

This project must be documented according to the guidelines of the "Informatik" department. This includes all analysis, design, implementation, project management, etc. sections. All documentation is expected to be written in English. The project plan also contains the documentation tasks. All results must be complete in the final upload to the archive server. There is no need to print out the documentation

1.6 Important Dates

⚠ Official documents

Check the official documents and relegments

Date	Event
20.02.2023	Start of the student project
12.06.2023	Hand-in of the abstract using the online tool abstract.rj.ost.ch
16.06.2023 17:00	Final hand-in of the report using the online tool avt.i.ost.ch
21.07.2023 14:00	Presentation

1.7 Evaluation

⚠ Official documents

Check the official documents and relegments

Figure A.3: Bachelor Thesis Assignment Page 3

INS | Institute for Network and Security

***SSoT based network service deployment** v 1.0*

Criterion	Weight
Organization and implementation	10%
Formal quality of the report	10 %
Analysis, design and evaluation	20 %
Technical implementation	40 %
Presentation	20 %

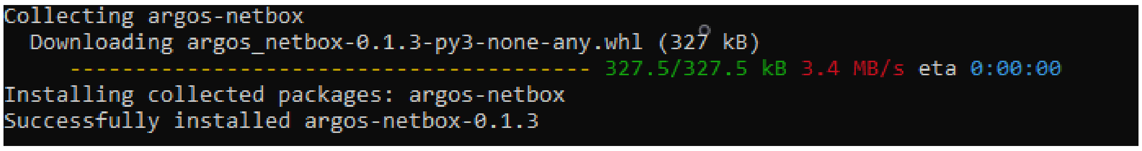
1 Assignment 3

Figure A.4: Bachelor Thesis Assignment Page 4

Appendix B

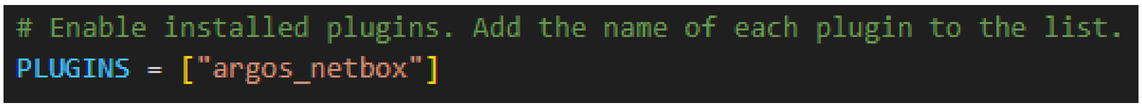
Screenshots

B.1 NFR Validation Screenshots



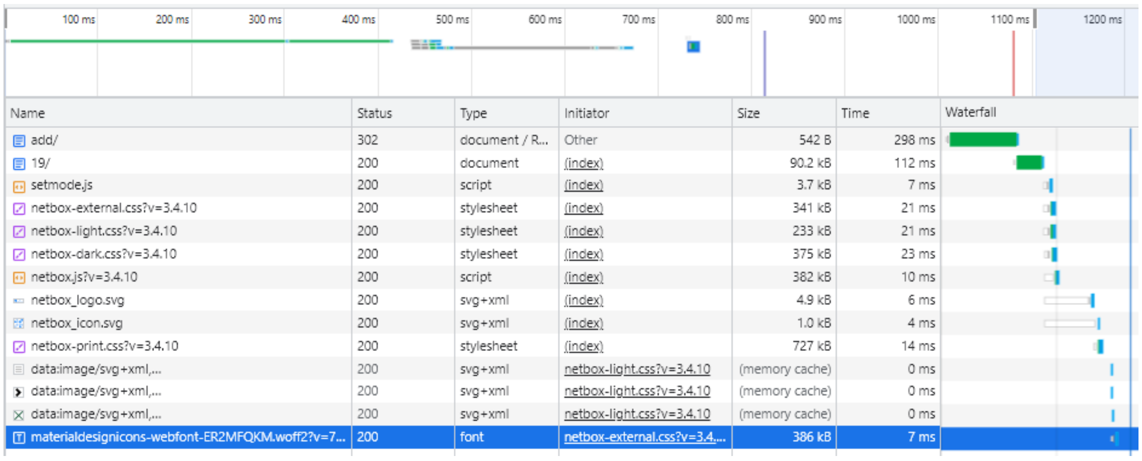
```
Collecting argos-netbox
  Downloading argos_netbox-0.1.3-py3-none-any.whl (327 kB)
     ---------------------------------------- 327.5/327.5 kB 3.4 MB/s eta 0:00:00
Installing collected packages: argos-netbox
Successfully installed argos-netbox-0.1.3
```

Figure B.1: NFR Portability Argos-NetBox Pip Command



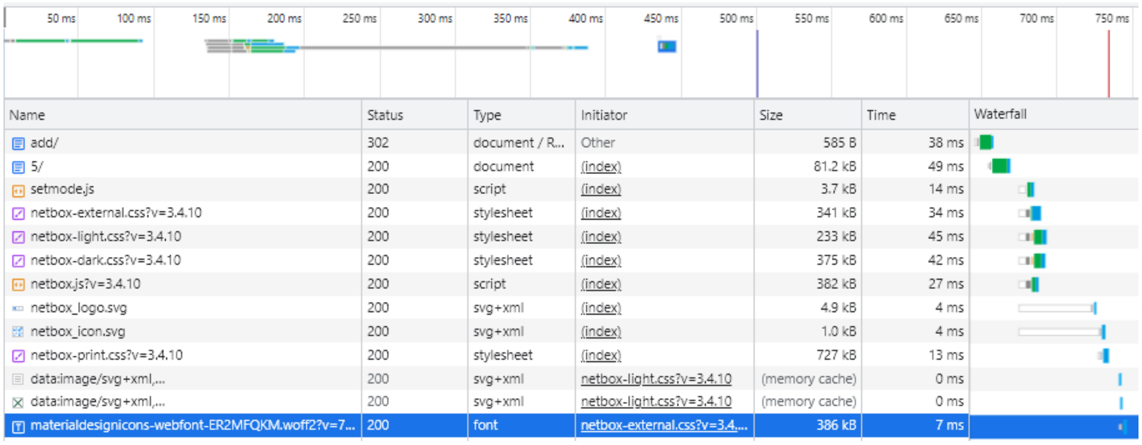
```
# Enable installed plugins. Add the name of each plugin to the list.
PLUGINS = ["argos_netbox"]
```

Figure B.2: NFR Portability Argos-NetBox Plugin Configuration



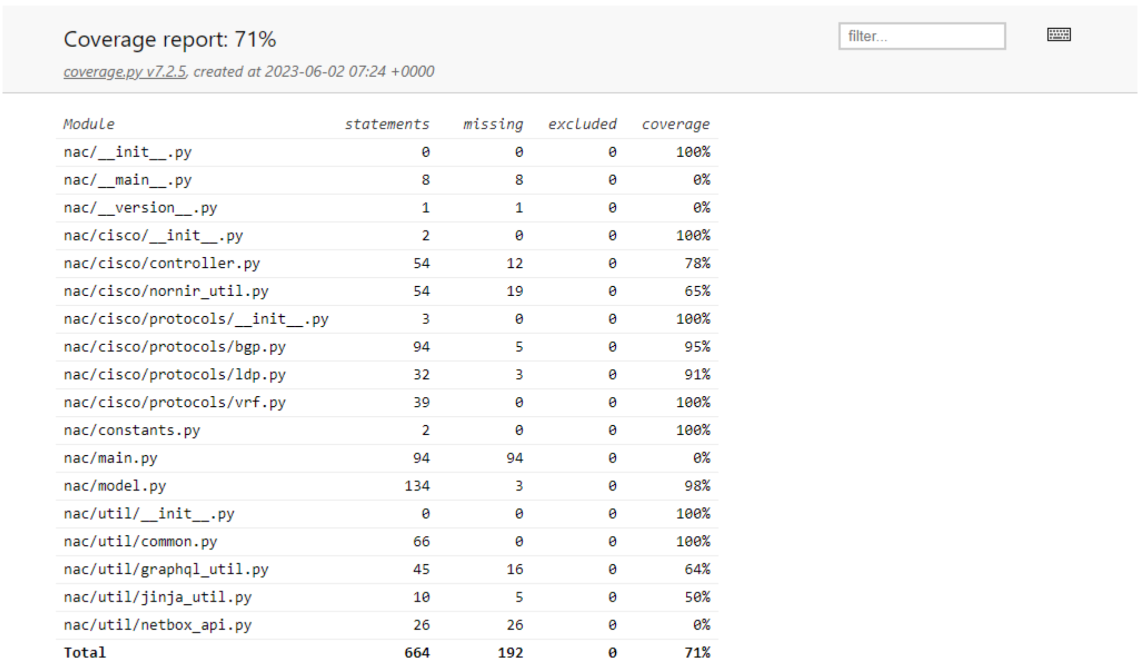
Name	Status	Type	Initiator	Size	Time	Waterfall
add/	302	document / R...	Other	542 B	298 ms	
19/	200	document	(index)	90.2 kB	112 ms	
setmode.js	200	script	(index)	3.7 kB	7 ms	
netbox-external.css?v=3.4.10	200	stylesheet	(index)	341 kB	21 ms	
netbox-light.css?v=3.4.10	200	stylesheet	(index)	233 kB	21 ms	
netbox-dark.css?v=3.4.10	200	stylesheet	(index)	375 kB	23 ms	
netbox.js?v=3.4.10	200	script	(index)	382 kB	10 ms	
netbox_logo.svg	200	svg+xml	(index)	4.9 kB	6 ms	
netbox_icon.svg	200	svg+xml	(index)	1.0 kB	4 ms	
netbox-print.css?v=3.4.10	200	stylesheet	(index)	727 kB	14 ms	
data:image/svg+xml,...	200	svg+xml	netbox-light.css?v=3.4.10	(memory cache)	0 ms	
data:image/svg+xml,...	200	svg+xml	netbox-light.css?v=3.4.10	(memory cache)	0 ms	
data:image/svg+xml,...	200	svg+xml	netbox-light.css?v=3.4.10	(memory cache)	0 ms	
materialdesignicons-webfont-ER2MFQKM.woff2?v=7...	200	font	netbox-external.css?v=3.4....	386 kB	7 ms	

Figure B.3: NFR Performance Add Device Time



Name	Status	Type	Initiator	Size	Time	Waterfall
add/	302	document / R...	Other	585 B	38 ms	
5/	200	document	(index)	81.2 kB	49 ms	
setmode.js	200	script	(index)	3.7 kB	14 ms	
netbox-external.css?v=3.4.10	200	stylesheet	(index)	341 kB	34 ms	
netbox-light.css?v=3.4.10	200	stylesheet	(index)	233 kB	45 ms	
netbox-dark.css?v=3.4.10	200	stylesheet	(index)	375 kB	42 ms	
netbox.js?v=3.4.10	200	script	(index)	382 kB	27 ms	
netbox_logo.svg	200	svg+xml	(index)	4.9 kB	4 ms	
netbox_icon.svg	200	svg+xml	(index)	1.0 kB	4 ms	
netbox-print.css?v=3.4.10	200	stylesheet	(index)	727 kB	13 ms	
data:image/svg+xml,...	200	svg+xml	netbox-light.css?v=3.4.10	(memory cache)	0 ms	
data:image/svg+xml,...	200	svg+xml	netbox-light.css?v=3.4.10	(memory cache)	0 ms	
materialdesignicons-webfont-ER2MFQKM.woff2?v=7...	200	font	netbox-external.css?v=3.4....	386 kB	7 ms	

Figure B.4: NFR Performance Add MPLS Instance Time

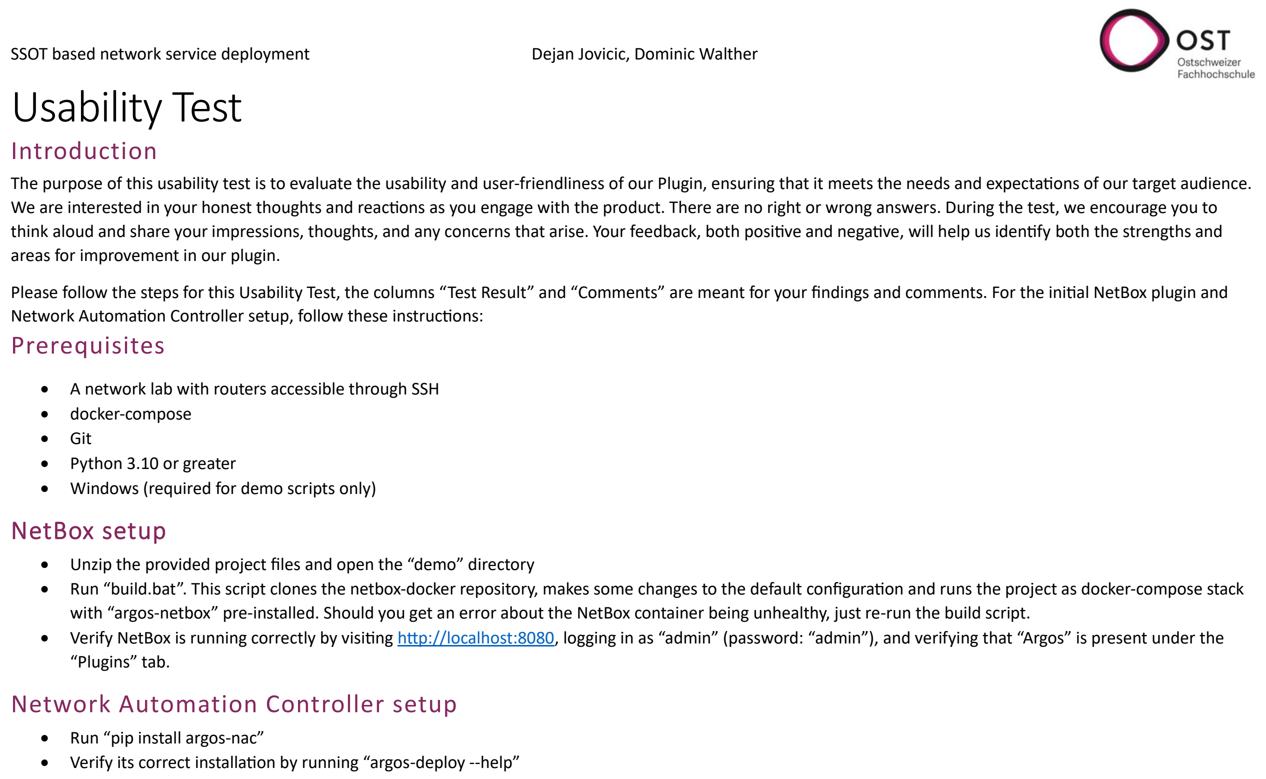

Coverage report: 71%

coverage.py v7.2.5, created at 2023-06-02 07:24 +0000

Module	statements	missing	excluded	coverage
nac/__init__.py	0	0	0	100%
nac/__main__.py	8	8	0	0%
nac/__version__.py	1	1	0	0%
nac/cisco/__init__.py	2	0	0	100%
nac/cisco/controller.py	54	12	0	78%
nac/cisco/nornir_util.py	54	19	0	65%
nac/cisco/protocols/__init__.py	3	0	0	100%
nac/cisco/protocols/bgp.py	94	5	0	95%
nac/cisco/protocols/ldp.py	32	3	0	91%
nac/cisco/protocols/vrf.py	39	0	0	100%
nac/constants.py	2	0	0	100%
nac/main.py	94	94	0	0%
nac/model.py	134	3	0	98%
nac/util/__init__.py	0	0	0	100%
nac/util/common.py	66	0	0	100%
nac/util/graphql_util.py	45	16	0	64%
nac/util/jinja_util.py	10	5	0	50%
nac/util/netbox_api.py	26	26	0	0%
Total	**664**	**192**	**0**	**71%**

Figure B.5: NFR Testability Code Coverage

B.2 Usability Test Protocol Result



SSOT based network service deployment Dejan Jovicic, Dominic Walther

Usability Test

Introduction

The purpose of this usability test is to evaluate the usability and user-friendliness of our Plugin, ensuring that it meets the needs and expectations of our target audience. We are interested in your honest thoughts and reactions as you engage with the product. There are no right or wrong answers. During the test, we encourage you to think aloud and share your impressions, thoughts, and any concerns that arise. Your feedback, both positive and negative, will help us identify both the strengths and areas for improvement in our plugin.

Please follow the steps for this Usability Test, the columns “Test Result” and “Comments” are meant for your findings and comments. For the initial NetBox plugin and Network Automation Controller setup, follow these instructions:

Prerequisites

- A network lab with routers accessible through SSH
- docker-compose
- Git
- Python 3.10 or greater
- Windows (required for demo scripts only)

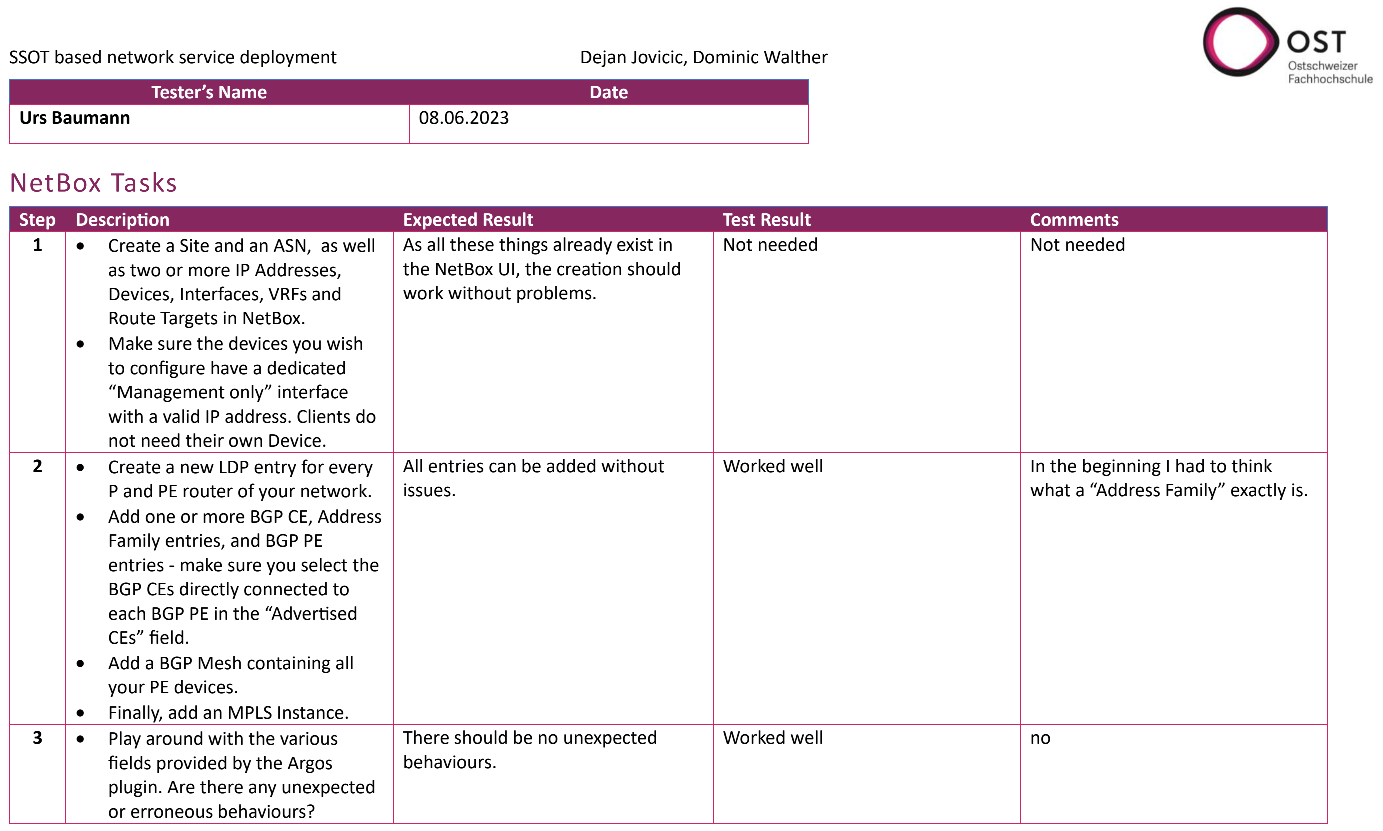
NetBox setup

- Unzip the provided project files and open the “demo” directory
- Run “build.bat”. This script clones the netbox-docker repository, makes some changes to the default configuration and runs the project as docker-compose stack with “argos-netbox” pre-installed. Should you get an error about the NetBox container being unhealthy, just re-run the build script.
- Verify NetBox is running correctly by visiting http://localhost:8080, logging in as “admin” (password: “admin”), and verifying that “Argos” is present under the “Plugins” tab.

Network Automation Controller setup

- Run “pip install argos-nac”
- Verify its correct installation by running “argos-deploy --help”

Figure B.6: Usability Test Urs Baumann Page 1



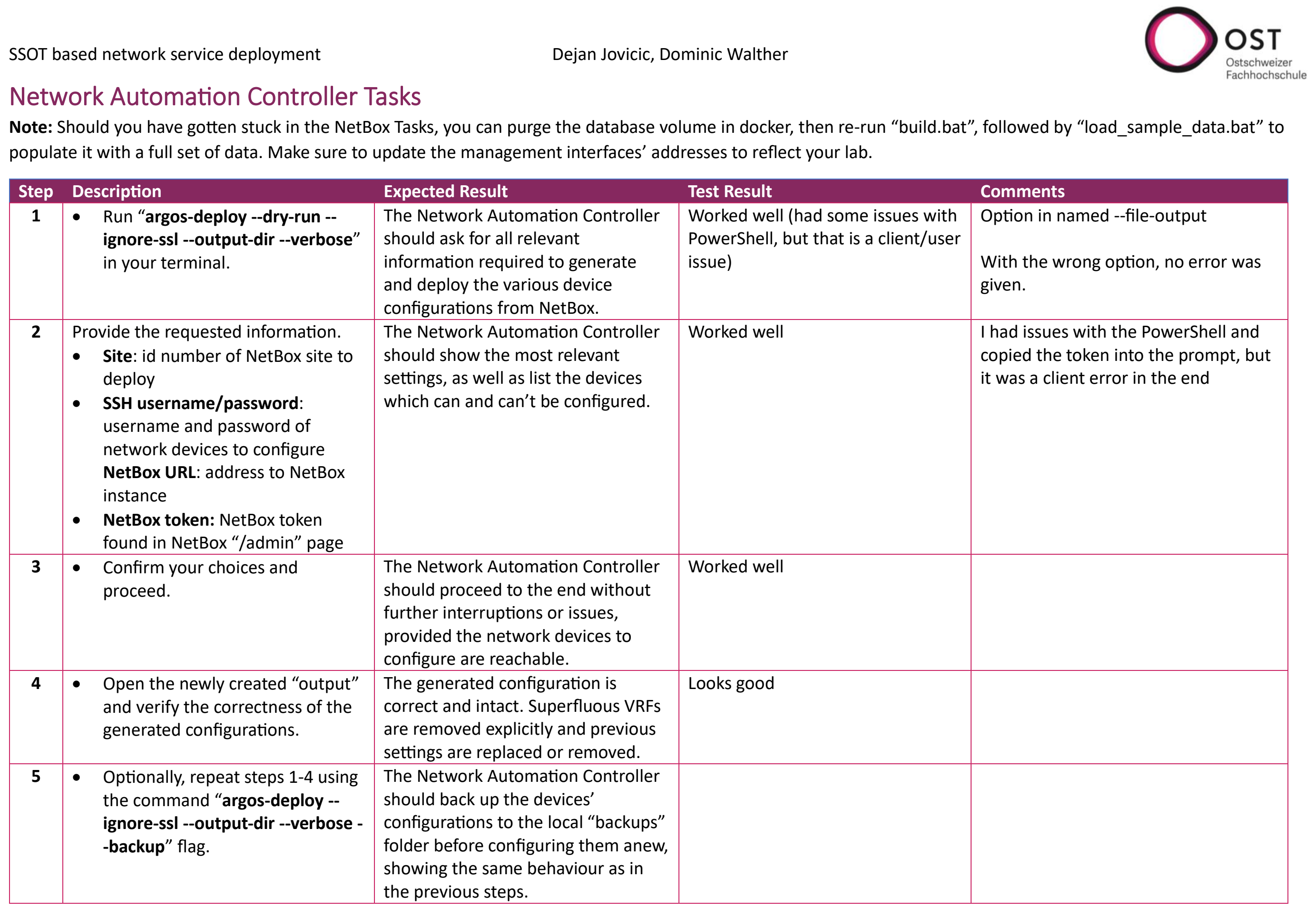
SSOT based network service deployment Dejan Jovicic, Dominic Walther

Tester's Name	Date
Urs Baumann	08.06.2023

NetBox Tasks

Step	Description	Expected Result	Test Result	Comments
1	• Create a Site and an ASN, as well as two or more IP Addresses, Devices, Interfaces, VRFs and Route Targets in NetBox. • Make sure the devices you wish to configure have a dedicated "Management only" interface with a valid IP address. Clients do not need their own Device.	As all these things already exist in the NetBox UI, the creation should work without problems.	Not needed	Not needed
2	• Create a new LDP entry for every P and PE router of your network. • Add one or more BGP CE, Address Family entries, and BGP PE entries - make sure you select the BGP CEs directly connected to each BGP PE in the "Advertised CEs" field. • Add a BGP Mesh containing all your PE devices. • Finally, add an MPLS Instance.	All entries can be added without issues.	Worked well	In the beginning I had to think what a "Address Family" exactly is.
3	• Play around with the various fields provided by the Argos plugin. Are there any unexpected or erroneous behaviours?	There should be no unexpected behaviours.	Worked well	no

Figure B.7: Usability Test Urs Baumann Page 2



SSOT based network service deployment Dejan Jovicic, Dominic Walther

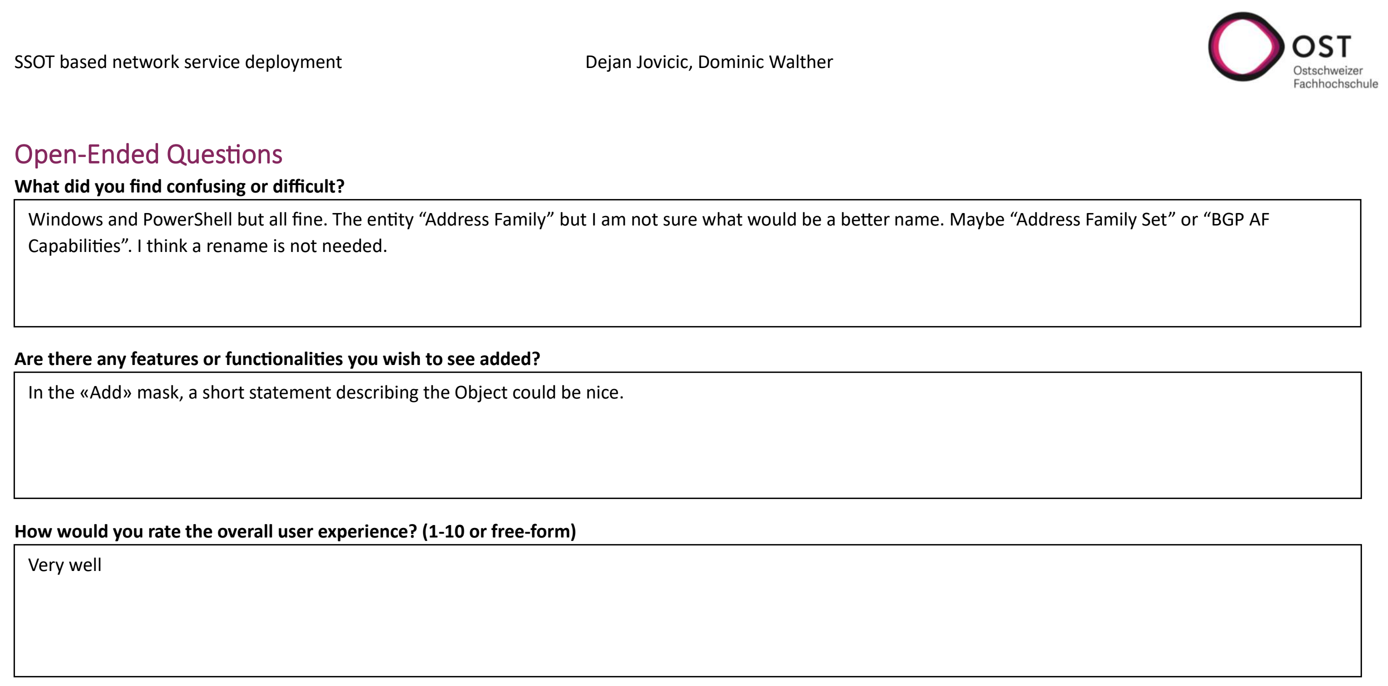
OST Ostschweizer Fachhochschule

Network Automation Controller Tasks

Note: Should you have gotten stuck in the NetBox Tasks, you can purge the database volume in docker, then re-run "build.bat", followed by "load_sample_data.bat" to populate it with a full set of data. Make sure to update the management interfaces' addresses to reflect your lab.

Step	Description	Expected Result	Test Result	Comments
1	• Run "**argos-deploy --dry-run --ignore-ssl --output-dir --verbose**" in your terminal.	The Network Automation Controller should ask for all relevant information required to generate and deploy the various device configurations from NetBox.	Worked well (had some issues with PowerShell, but that is a client/user issue)	Option in named --file-output With the wrong option, no error was given.
2	Provide the requested information. • **Site**: id number of NetBox site to deploy • **SSH username/password**: username and password of network devices to configure **NetBox URL**: address to NetBox instance • **NetBox token:** NetBox token found in NetBox "/admin" page	The Network Automation Controller should show the most relevant settings, as well as list the devices which can and can't be configured.	Worked well	I had issues with the PowerShell and copied the token into the prompt, but it was a client error in the end
3	• Confirm your choices and proceed.	The Network Automation Controller should proceed to the end without further interruptions or issues, provided the network devices to configure are reachable.	Worked well	
4	• Open the newly created "output" and verify the correctness of the generated configurations.	The generated configuration is correct and intact. Superfluous VRFs are removed explicitly and previous settings are replaced or removed.	Looks good	
5	• Optionally, repeat steps 1-4 using the command "**argos-deploy --ignore-ssl --output-dir --verbose --backup**" flag.	The Network Automation Controller should back up the devices' configurations to the local "backups" folder before configuring them anew, showing the same behaviour as in the previous steps.		

Figure B.8: Usability Test Urs Baumann Page 3



SSOT based network service deployment Dejan Jovicic, Dominic Walther

Open-Ended Questions

What did you find confusing or difficult?

Windows and PowerShell but all fine. The entity “Address Family” but I am not sure what would be a better name. Maybe “Address Family Set” or “BGP AF Capabilities”. I think a rename is not needed.

Are there any features or functionalities you wish to see added?

In the «Add» mask, a short statement describing the Object could be nice.

How would you rate the overall user experience? (1-10 or free-form)

Very well

Figure B.9: Usability Test Urs Baumann Page 4

Bibliography

[1] Ansible, “How ansible works,” last time accessed: 26/03/2023. [Online]. Available: https://www.ansible.com/overview/how-ansible-works

[2] Dominic-Walther, “Underscores in model names results in broken permissions · issue #12809,” June 2023, last time accessed: 14/06/2023. [Online]. Available: https://github.com/netbox-community/netbox/issues/12809

[3] M. Leiniö, “ci: test against the currently supported python versions - pull request #446,” Feb 2022, last time accessed: 11/03/2023. [Online]. Available: https://github.com/netbox-community/pynetbox/pull/446#issuecomment-1031803433

[4] Nornir, “Welcome to nornir’s documentation!” Mar 2023, last time accessed: 27/03/2023. [Online]. Available: https://nornir.readthedocs.io/en/latest/index.html

[5] PieterL75, “Assign asn to prefix and device,” Mar 2022, last time accessed: 11/06/2023. [Online]. Available: https://github.com/netbox-community/netbox/issues/8782

[6] A. Rachum, “A case for a onetomany relationship in django,” last time accessed: 12/05/2023. [Online]. Available: https://amir.rachum.com/a-case-for-a-onetomany-relationship-in-django/

[7] SALTSTACK, “Salt project,” Mar 2023, last time accessed: 26/03/2023. [Online]. Available: https://docs.saltproject.io/en/latest/topics/about_salt_project.html#about-salt

[8] StackStorm, “Stackstorm overview,” last time accessed: 26/03/2023. [Online]. Available: https://docs.stackstorm.com/overview.html#how-it-works

[9] J. Stretch, “Contributing.md,” Jan 2023, last time accessed: 14/06/2023. [Online]. Available: https://github.com/netbox-community/netbox/blob/36e0bf0490e084c835098d6d865d8e7a3f3caa6a/CONTRIBUTING.md?plain=1#L42