UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
PROGRAMA DE RESIDÊNCIA EM TECNOLOGIA DA INFORMAÇÃO

Automação das configurações de switches usando o Netbox como elemento de orquestração.

Thiago de Oliveira Nunes Galeno

Natal-RN, Brasil
2020

Thiago de Oliveira Nunes Galeno

Automação das configurações de switches usando o Netbox como elemento de orquestração.

Trabalho de Conclusão de Curso apresentado ao Programa de Residência em Tecnologia da Informação do Instituto Metrópole Digital da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do título de Especialista em Tecnologia da Informação. Área de Concentração: Redes e Infraestrutura

Orientador: Marcos César Madruga Alves Pinheiro

Natal-RN, Brasil
2020

Thiago de Oliveira Nunes Galeno

Automação das configurações de switches usando o Netbox como elemento de orquestração.

Trabalho de Conclusão de Curso apresentado ao Programa de Residência em Tecnologia da Informação do Instituto Metrópole Digital da Universidade Federal do Rio Grande do Norte como requisito parcial para a obtenção do título de Especialista em Tecnologia da Informação. Área de Concentração: Redes e Infraestrutura

Trabalho aprovado. Natal-RN, Brasil, 28 de Outubro de 2020:

Prof. Dr. Marcos César Madruga A. Pinheiro
Orientador

Prof. Msc. Carlos Gustavo Araújo da Rocha
Examinador

Prof. Msc. Wellington Silva de Souza
Examinador

Natal-RN, Brasil
2020

Dedico este trabalho aos meus pais, minha fonte de sabedoria, a todos os meus amigos e a toda turma de residência em TI da JFRN, aos professores e funcionários da JFRN. Foi um grande prazer ter trabalhado e aprendido com todos vocês.

Agradecimentos

Agradeço a Deus pois, dEle e por Ele, e para Ele, são todas as coisas.

Aos meus pais Samuel e Eronildes pelo apoio e suporte, e pelos puxões de orelha que me ensinaram a não desistir e a seguir em frente mesmo quando as coisas parecerem difíceis, vocês são o meu maior exemplo na vida.

A minha família em especial a minha tia Luciene que é como uma segunda mãe e quem sempre acreditou no meu potencial e me incentivou a estudar. Ao meu primo Desnes que é uma das minhas inspirações de dedicação aos estudos, e me incentivou a seguir nesta área tão cheia de alegrias e desafios que é a TI.

Aos meus amigos que sempre acreditaram em mim e me fizeram rir, sempre que as coisas estavam difíceis.

Aos professores da residência da JFRN, em especial ao meu orientador deste Trabalho de conclusão de curso (TCC), o professor, Marcos César Madruga Alves Pinheiro, que foi essencial para o sucesso deste trabalho.

Aos meus colegas residentes, em especial, aos da equipe de redes e infraestrutura comigo, Ana e Deyvison pois compartilhavam das mesmas dores e alegrias e sempre dispostos a ajudar nas atividades. Também agradeço o apoio e ajuda da equipe do Núcleo de Tecnologia de Informação e Comunicação (NTIC) profissionais sempre proativos em resolver os problemas, em especial David, Bruno e Wellington, sempre solícitos em ajudar em qualquer adversidade que aparecia na residência.

“Nenhum esforço faz sentido, se você não acredita em si mesmo”
(Maito Gai, Naruto - Masashi Kishimoto)

Resumo

A configuração de dispositivos é uma tarefa crítica de gerenciamento, uma vez que modifica o estado da rede. Embora na atualidade ainda se configure os equipamentos de forma manual via CLI, muito se tem avançado nos estudos por uma melhor gerência de rede e de configuração dos equipamentos. Para tal faz-se necessário protocolos padronizados, que propõem padrões que sejam simples e genéricos o suficiente de forma a abranger todos os tipos de dispositivos. Este trabalho apresenta um novo modelo de gerência da rede, baseado no protocolo Network Configuration Protocol (NETCONF) e como estudo de caso realiza a gerência de configuração de criação de Virtual Local Area Network (VLAN) e atualização dos tipos de portas de ativos de rede (*switches*) de forma automatizada através de uma documentação da rede que esteja concisa com as configurações vigentes no dispositivo, e uma vez alterada, seja modificada a configuração no respectivo *switch*. Para tal foi utilizado o Netbox, ferramenta de documentação de rede que tem mecanismos que enviam via solicitação Hypertext Transfer Protocol (HTTP) o que foi modificado nas configurações dos dispositivos salvos. Além disso foi criado um webservice para coleta dessas informações, tratamento e envio para os ativos de rede, de forma a garantir que a configuração seja aplicada de forma automatizada sem intervenção humana no processo.
Palavras-chave: Automação. Netbox. Webservice. Switches. Orquestração de rede. Netconf.

Abstract

Device configuration is a critical management task as it changes the state of the network. Although equipment is still configured manually via CLI today, much progress has been made in studies for better network management and equipment configuration. For that, it is necessary to have standardized protocols, which propose standards that are simple and generic enough to cover all types of devices. This work presents a new model of network management, using NETCONF, a network protocol for this purpose. And as a case study of the use of NETCONF, in configuration management in the process of creating VLANs and update network asset (*switchs*) port types in an automated way through network documentation that is concise with the settings current in the device, and once changed, the configuration in the respective *switch* is modified. For this purpose, Netbox was used, a network documentation tool that has mechanisms that send via request HTTP which has been modified in the settings of the saved devices. In addition, a web service was created to collect this information, process it and send it to network assets, in order to ensure that the configuration is applied in an automated way without human intervention in the process.

Keywords:Automation. Netbox. Webservice. Switches. Network orchestration. Netconf.

Lista de ilustrações

Lista de Códigos

Lista de tabelas

Lista de abreviaturas e siglas

API Application Programming Interface

CLI Command-line Interface

HP Hewlett-Packard

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

IETF Internet Engineering Task Force

JFRN Justiça Federal no Rio Grande do Norte

JSON JavaScript Object Notation

NETCONF Network Configuration Protocol

NTIC Núcleo de Tecnologia Informação e Comunicação

REST Representational State Transfer

RESTful Arquitetura que atende o padrão REST

RFC Request for Comments

RPC Remote Procedure Call

SSH Secure Shell

TCC Trabalho de conclusão de curso

TI Tecnologia da Informação

TXT text-formatted data

URI Uniform Resource Identifier

URL Uniform Resource Locator

VLAN Virtual Local Area Network

XML Extensible Markup Language

YANG Yet Another Next Generation

Sumário

1 Introdução

No cenário atual das grandes instituições temos redes cada vez mais complexas e heterogêneas, por usarem equipamentos de diferentes fabricantes, fazendo-se necessário diversos tipos de programas de gerenciamento, além de precisar muitas vezes de um operador especialista em redes dedicado a gerência desses equipamentos, e que conheça muito bem os modelos e marcas.(LACERDA, 2007)

Mesmo em redes de médio e grande porte, a gerência e configuração destes equipamentos não é realizada de forma automática, necessitando sempre de intervenção humana. No entanto, para uma rede deste porte, formada por vários elementos de rede, configurar os elementos de rede manualmente via Command-line Interface (CLI), ou mesmo com *scripts*, significa estar sujeito a uma alta margem de erro. Além disso tem-se o problema na demora com deslocamento até os equipamentos, configurar cada dispositivo com suas peculiaridades, e testes para se assegurar que as mudanças foram satisfatórias. (LACERDA, 2007)

Nesse cenário, a configuração de dispositivos é uma das mais críticas tarefas de gerenciamento, a que envolve a alteração no estado da rede, da qual cada vez mais se exige um funcionamento estável e com um menor número possível de falhas ou interrupções. Com isso, faz-se necessário a utilização de protocolos padronizados e específicos para o gerenciamento de configuração. Dentre os requisitos de um protocolo para esse fim, pode-se citar, entre outros, suporte a transações, controle de conflitos, notificação de erros e capacidade de *rollback*. (IETF, 2003 apud TELOKEN, 2006)

Para se resolver os problemas de configuração e gerenciamento do ambiente de rede, é interessante se adotar um protocolo padronizado, para tal foi idealizado o NETCONF, que nasceu com o objetivo de unificar a maneira pela qual os dispositivos de rede são configurados, propondo padrões que sejam simples e genéricos o suficiente de forma a abranger todos os tipos de dispositivos, suprindo assim as deficiências das tecnologias atualmente disponíveis. (TELOKEN, 2006)

No cenário atual da Justiça Federal no Rio Grande do Norte (JFRN) no que diz respeito a documentação e configuração dos equipamentos, segue-se o modelo tradicional, onde essas informações são armazenadas em uma base de conhecimento - *wiki* e os equipamentos são configurados manualmente via CLI o que traz problemas como: falta de um modelo de dados consistente, CLIs diferem entre os fabricantes e tendência de ser voltada para a leitura humana, o que dificulta a configuração do dispositivo de forma programática. Este formato de trabalho como citado anteriormente traz consigo alguns problemas, tais quais: perda de tempo com deslocamento até os equipamentos, documentação na base de conhecimento encontra-se em desconformidade com as configurações vigentes nos *switches*, além da possibilidade de erro no processo de configuração.

1.1 Motivação

Este trabalho foi motivado a partir da experiência do processo diário de trabalho na JFRN, o qual é semelhante a diversas outras instituições publicas, onde há a necessidade de logar nos equipamentos via CLI, e por muitas vezes requerer o deslocamento físico do funcionário até as salas de rede para realizar as configurações necessárias. Baseado em como isso se torna oneroso e demanda mais tempo à equipe de suporte, fez-se necessário melhorar o processo de documentação e configuração dos switches.

1.2 Objetivos

1.2.1 Objetivo geral

Apresentar um novo modelo de gerência da rede, ao se utilizar um protocolo de rede com esta finalidade como é o NETCONF, e como estudo de caso de uso do NETCONF este trabalho propõe a gerência de configuração no processo de criação de VLANs e atualização dos tipos de portas de ativos de rede(*switches*) de forma automatizada, através de uma documentação da rede que esteja concisa com as configurações vigentes no dispositivo, e uma vez alterada, seja modificada a configuração no respectivo *switch*.

1.2.2 Objetivos específicos

a) Instalar e utilizar o Netbox para documentação dos *switches*, por se tratar de um sistema *open-source* de documentação de rede, contando ainda com mecanismos que garantem o processo de automação e integração com outros sistemas.

b) Criar um webservice para receber as informações do Netbox via *gateway* Application Programming Interface (API), tratá-las e aplicar nos equipamentos via código de forma automática utilizando os protocolos Secure Shell (SSH) e NETCONF com as informações necessárias, sem necessidade de intervenção humana no processo.

1.3 Estrutura do trabalho

Este texto está estruturado no seguinte formato, o Capítulo 2 será tratado o embasamento teórico necessário para a elaboração deste trabalho. No Capítulo 3 serão detalhados o Netbox, menus utilizados e seu mecanismo de integração. No Capítulo 4 sera detalhada a solução e seus componentes. Por fim, no Capítulo 5 será a conclusão deste trabalho.

2 Embasamento Teórico

Neste capítulo será apresentado o embasamento teórico necessário para elaboração deste trabalho.

2.1 Automação

> A automação de rede não se trata apenas de automatizar a configuração de dispositivos de rede. É verdade que é a percepção mais comum de automação de rede, mas usar APIs e interfaces programáticas pode automatizar e oferecer muito mais do que enviar parâmetros de configuração. Aproveitar uma API agiliza o acesso a todos os dados armazenados em dispositivos de rede. Pense em dados como dados de nível de fluxo, tabelas de roteamento, tabelas FIB, estatísticas de interface, tabelas MAC, tabelas VLAN, números de série e a lista pode continuar indefinidamente. O uso de técnicas de automação modernas que, por sua vez, alavancam uma API, pode ajudar rapidamente nas operações diárias de gerenciamento de redes para coleta de dados e diagnósticos automatizados. Além disso, como está sendo usada uma API que retorna dados estruturados, como administrador, você terá a capacidade de exibir e analisar o conjunto de dados exato que deseja e precisa, mesmo vindo de vários comandos show, reduzindo o tempo que leva para depurar e solucionar problemas na rede. Adaptado de (EDELMAN SCOTT S. LOWE, 2016)

Automação, como o nome sugere, é um *framework* para automatizar uma tarefa específica, compreendendo, interpretando e criando lógica. Isso inclui aprimorar os recursos atuais das tarefas executadas manualmente e reduzir a taxa de erro, concentrando-se em dimensionar a tarefa com menos esforço.Adaptado de (RATAN, 2017)

2.2 Netbox

Netbox (NETBOX, 2020) é um aplicação *Web open-source* projetado, para ajudar a gerenciar e documentar infraestrutura de redes. Foi projetado inicialmente pela equipe de engenharia de rede da DigitalOcean, e apesar de não ter sido criado com o intuito de gerenciar configuração, os dados armazenados no mesmo podem ser utilizados para apoiar a execução de funções que automatizarão as configurações nos *switches*.

O Netbox foi utilizado por sua robustez e por se tratar de uma aplicação que pode ser executada em container, o que é interessante para JFRN uma vez que estamos buscando cada vez mais migrar para esse tipo de ambiente, além disso tem uma API muito bem documentada e com uso simples, além de ter *webhooks* para chamadas de serviços Web. Com isso conseguimos garantir que a documentação descrita no Netbox, está em conformidade com a aplicada nos equipamentos.

2.3 Tipos de porta e vlans nos switches

Nos switches da JFRN existem três tipos de portas que podem ser aplicadas nos switches, que são:

- Access: Uma porta de acesso onde só trafegam dados referentes a uma única vlan, se está porta receber pacotes com o valor do campo ID da vlan diferente da que está associada a ela, o pacote é descartado.
- Hybrid: Uma porta que permite mais de uma VLANs com trafego marcado com a tag da VLAN ou não, podendo ter uma vlan ser marcação.
- Trunk: Uma porta por onde passam todas as VLANs do dispositivo, podendo ter uma VLAN sem marcação.

Uma vez que o tipo de porta foi definido, o próximo passo é aplicar as VLANs, que podem ser classificadas em duas categorias:

- Untagged: quadro é enviado sem o ID da VLAN
- Tagged: quadro é enviado contendo o ID da VLAN

2.4 Protocolos de comunicação com os switches

Para que as configurações sejam feitas de forma automática nos switches, faz-se necessário o uso de protocolos de comunicação para transmissão dos dados até os equipamentos de rede de maneira segura e eficiente. Neste trabalho usaremos os protocolos descritos a a seguir.

2.4.1 SSH

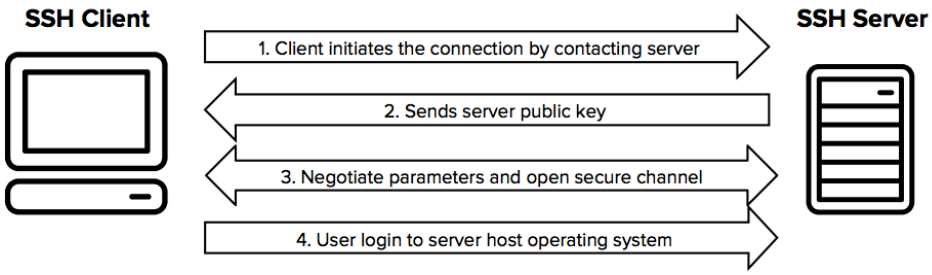
SSH, ou em português Shell seguro, é um protocolo de rede baseado nas Request for Comments (RFC)s 4251 (IETF, 2006c), 4252 (IETF, 2006a), 4253 (IETF, 2006d) e 4254 (IETF, 2006b) para operação de serviços de rede de forma segura devido a troca de mensagens entre o SSH client e o SSH server ser criptografada. É comumente usado para administração de servidores remotos e acesso a equipamentos de rede, tendo sido criado originalmente para substituir o telnet onde o tráfego não era criptografado.

Existem algumas maneiras de se utilizar SSH para atender às demandas, sendo que as formas mais comuns de uso são através de chaves pública-privada, para criptografado a conexão de forma que o que for trafegado está seguro e criptografado. Esta é a forma usada geralmente para configuração de dispositivos de rede por parte de uma equipe de Tecnologia da Informação (TI), sendo necessário ainda informar usuário e senha para login

no dispositivo, e após isso inserir os comandos de configuração. Outra forma é chaves público-privada geradas manualmente para autenticação automática.

A Figura 1 mostra como acontece a comunicação entre o cliente e o servidor SSH.

Figura 1 – Conexão ssh cliente - servidor



Fonte:(YLONEN, 2020)

2.4.2 NETCONF

O protocolo NETCONF define um mecanismo simples por meio do qual um dispositivo de rede pode ser gerenciado, as informações de dados de configuração podem ser recuperadas e novos dados de configuração podem ser carregados e manipulados. O protocolo permite que o dispositivo exponha uma interface de programação de aplicativo API completa e formal. Os aplicativos podem usar esta API simples para enviar e receber conjuntos de dados de configuração completos ou parciais.(Adaptado de (IETF, 2011))

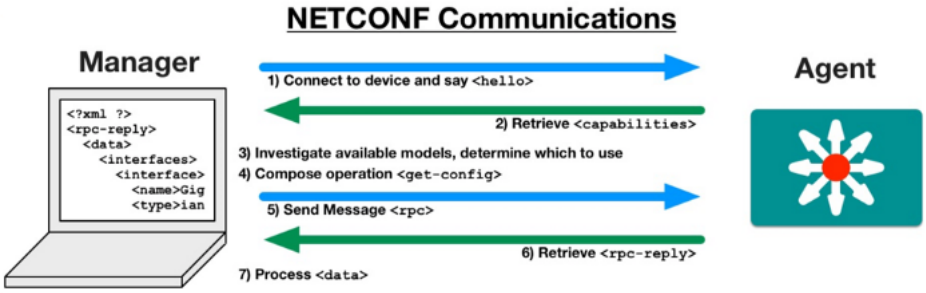
O NETCONF IETF (2011) surgiu da necessidade de um fluxo de trabalho mais ágil dentro de redes. Utiliza o SSH para transporte das mensagens, que por sua vez são encaminhadas até o servidor NETCONF do dispositivo (porta 830 por padrão).

O NETCONF se sobressai dos demais protocolos de gerência de redes por ele trabalhar com estado de configuração, Remote Procedure Call (RPC) e ser orientado a conexão, isso torna seu uso simples e intuitivo, pois uma vez que a conexão/sessão é estabelecida com a porta do servidor, pode-se enviar as mensagens NETCONF.

Segundo Claise Joe Clarke (2019) pode-se afirmar que a comunicação começa com o cliente envia uma mensagem de "olá" informando os recursos do mesmo. Já o servidor informa os modelos de dados com o qual trabalha e os recursos que aceita, e após isso o cliente envia as mensagens RPC (rpc-request) em formato Extensible Markup Language (XML) e o respectivo ID, uma vez que essa alteração na árvore de dados (que pode ser a árvore de configuração em execução, a do próxima inicialização ou uma configuração paralela que entra em execução após a confirmação por parte do cliente/usuário) forem executadas com sucesso, é enviada uma mensagem RPC (rpc-reply) com o ID solicitante informando "<ok/>" se bem sucedido ou uma mensagem de erro (rpc-error) e qual o tipo erro no caso de insucesso.

A Figura 2 ilustra de forma resumida como a comunicação entre cliente-servidor NETCONF funciona.

Figura 2 – Comunicação netconf cliente - servidor



Fonte: (OKASHA, 2017)

O trecho de Código 2.1 ilustra uma mensagem NETCONF no formato XML com uma solicitação para criação de uma vlan.

Código 2.1 – Mensagem Netconf em XML

```
<nc:config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
    <top xmlns="http://www.hp.com/netconf/config:1.0">
        <VLAN>
            <VLANs>
                <VLANID>
                    <ID>3000</ID>
                    <Description>msg netconf criacao vlan</Description>
                    <Name>Vlan 3000</Name>
                </VLANID>
            </VLANs>
        </VLAN>
    </top>
</nc:config>
```

2.4.2.1 Modelo de dados YANG

Um ponto importante sobre o NETCONF é que o mesmo não tem um modelo de dados implementado nele, por isso foi criado a linguagem de modelo de dados YANG que é regido pela RFC 7950 IETF (2016) para auxiliá-lo nesse propósito.

Uma vez que a mensagem RPC é enviada em XML para o servidor, o mesmo interpreta o XML e converte para um modelo de dados em Yet Another Next Generation (YANG) e só então salva no datastore, e aquele modelo passa a ser utilizável no servidor.

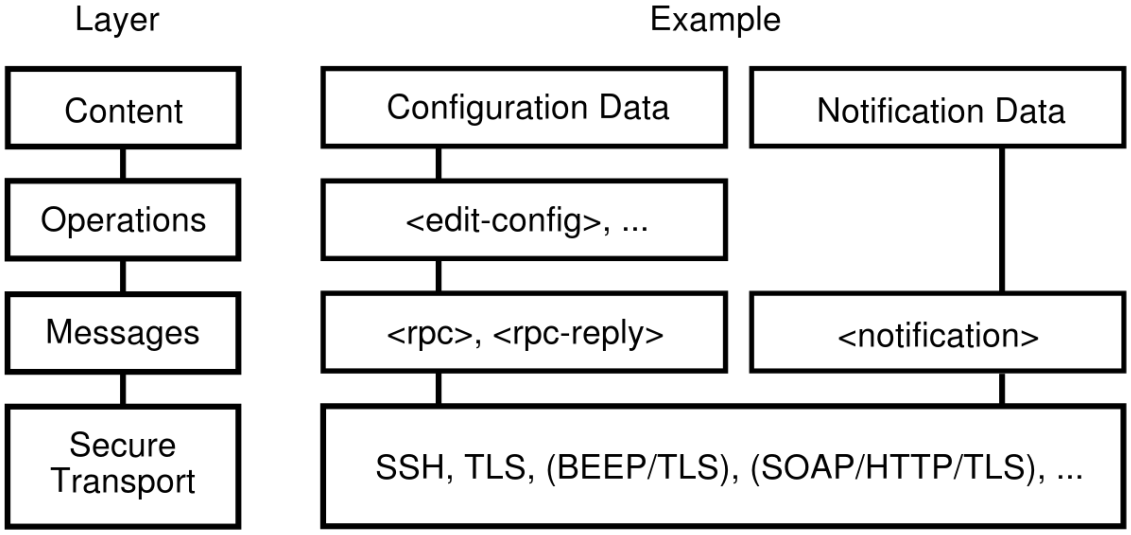
Vale ressaltar que a maioria dos dispositivos de rede, mais especificamente os switches que são o foco deste trabalho, já vem com os modelos de dados YANG suportados

definidos pelo fabricante, e o cliente NETCONF neste caso só altera os dados armazenados, e não os módulos YANG contidos no dispositivo.

2.4.2.2 Pilha NETCONF

A Figura 3 mostra as camadas da pilha NETCONF e a forma de transporte dos elementos da pilha.

Figura 3 – Pilha netconf



Fonte: (WIKIWAND, 2018)

A seguir serão detalhadas as camadas da pilha NETCONF:

1. Content

   Local onde é informado o modelo de dados que é seguido, pois como dito anteriormente o NETCONF não padroniza modelo de dados, de modo que, os modelos podem ser padrões abertos ou do próprio fabricante.

   O bloco de Código 2.2 mostra dois exemplos de content em XML representando o modelo YANG seguido, onde o primeiro utiliza um padrão aberto criado pelo grupo Internet Engineering Task Force (IETF), e o segundo modelo criado pelo fabricante a Hewlett-Packard (HP).

Código 2.2 – Content XML

```
1        <nc:config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
2            <top xmlns="http://www.hp.com/netconf/config:1.0">
```

2. Operations

   Campo onde informamos qual procedimento queremos fazer no modelo de dados do dispositivo.[1]

   A Tabela 1 mostra os tipos de operações e a descrição de cada uma delas.

Tabela 1 – Operações

Operação	Descrição
<get>	Retorna a configuração corrente no dispositivo
<get-config>	Retorna uma parte ou toda a configuração salva no datastore selecionado
<edit-config>	Carrega uma configuração toda ou uma parte específica no datastore selecionado
<copy-config>	Carrega e substitui toda a configuração salva no datastore selecionado
<delete-config>	Exclui configuração salva no datastore específico
<commit>	Copia uma configuração candidata para configuração corrente
<lock>/ <unlock>	Bloqueia/desbloqueia o datastore selecionado
<close-session>	Encerra com segurança a sessão netconf
<kill-session>	Força o encerramento da sessão netconf

Fonte: Produzido pelo autor

   As operações tem tipos e parâmetros que definem que datastore vai ser selecionado e o tipo de edição que se deseja fazer, conforme descrito a seguir:

   - Parâmetros
     - Source (get-config, copy-config): Nome do datastore que está sendo consultado.
     - Filter (get, get-config): Qual parte dos dados armazenados se quer ler
     - Target (get-config, copy-config, delete-config, lock/unlock): Datastore alvo que está sendo lido ou editado.
     - Default-operation (edit-config): Operação padrão a ser executada.
   - Tipos
     - Merge (edit-config): Mescla a configuração enviada com o estado atual que está no datastore.
     - Replace (edit-config): Substitui completamente os dados no datastore.
     - Create (edit-config): Cria o dado de configuração caso não exista
     - Delete (edit-config): Os dados com essa tag são excluídos do datastore
     - Remove (edit-config): Os dados com essa tag são excluídos do datastore, e caso não exista, o servidor ignora o pedido.

[1] Nem todas às operações e/ou atributos e parâmetros estarão disponíveis em todos os dispositivos, isso depende dos tipos de módulos YANG que estão carregados no dispositivo em questão.

– None (edit-config): Datastore não é afetado, é útil quando usa-se o delete, a fim de não excluir hierarquias pai de um determinado elemento do modelo que está sendo editado por acidente.

3. Message

   Essa camada engloba a comunicação entre as partes, fazendo com que a troca de mensagens seja codificado a parte do transporte. Cada tipo de mensagem representa o que está sendo executado.

   A tabela 2 mostra os tipos e a descrição.

Tabela 2 – Messagens xml

Mensagem	Descrição
<rpc>	Chamada de procedimento remoto
<rpc-request>	Solicitação por parte do cliente
<rpc-reply>	Resposta enviada do servidor para o cliente
<rpc-error>	Solicitação não foi executada e com o motivo do erro
<ok/>	Solicitação foi atendida

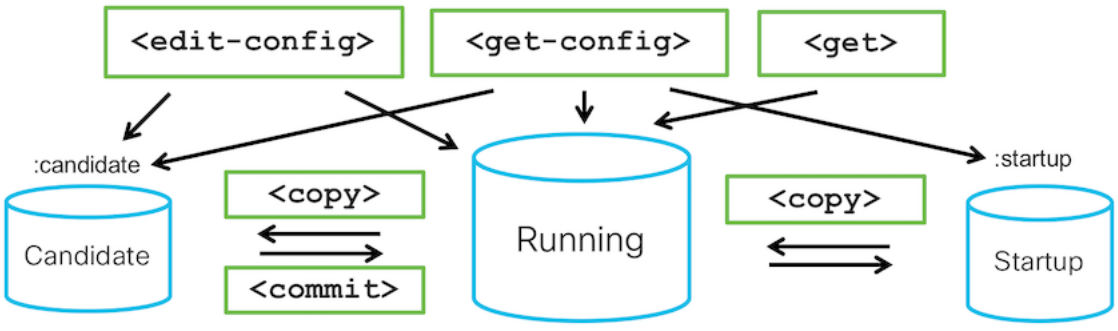
Fonte: Produzido pelo autor

4. Datastore

   Local onde ficam salvos os estados dos dados permitindo, por exemplo que configurações sejam realizadas, mas não sejam aplicadas imediatamente.

   A Figura 4 ilustra os três tipos de *datastores* usados na atualidade por parte do YANG. .

Figura 4 – Netconf datastore



Fonte: (CISCO, 2018)

- Tipos
  - Candidate: Estado de configuração paralela que após commit é carregado no running config.
  - Running: Estado de configuração corrente/em execução
  - Startup: Estado de configuração que é carregado quando o dispositivo é inicializado

5. Transport

   Fornece o meio seguro para que às mensagens netconf sejam trocadas. Por padrão faz-se uso do SSH pois o mesmo é configurado nativamente nos ativos de rede .

   O transporte via SSH é definido como obrigatório. Depois que o serviço de conexão SSH é estabelecido, o cliente abrirá um canal do tipo sessão. Uma vez que a sessão SSH tenha sido estabelecida, o usuário irá invocar o NETCONF como um subsistema chamado NETCONF.

   Caracteriza-se por indicar o fim da mensagem pela seguinte sequência de caracteres]]>]]>.(Lacerda (2007))

2.4.2.3 Cenário ideal protocolo padronizado de modelo de dados único

Para que às mensagens netconf fossem às mesmas em todos os dispositivos os fabricantes de equipamentos deveriam adotar modelos de dados YANG como os padronizados pelo OpenConfig OpenConfig (2016) em seus equipamentos. Infelizmente uma padronização entre a comunidade open-source e vendors leva tempo, o que faz com que as empresas necessitem desenvolver e usar seus modelos de dados privados, gerando mudanças no uso e formato das mensagens netconf de um equipamento para o outro e inviabilizando reaproveitamento de código entre equipamentos de vendors distintos.

2.5 API Rest

> O estilo de arquitetura REST é comumente aplicado ao design de APIs para serviços da web modernos. Uma API da Web em conformidade com o estilo arquitetônico REST é uma API REST.(Adaptado de (MASSE, 2011))

A sigla API em português significa interface de programação de aplicação, que se refere a um conjunto de rotinas e padrões estabelecidos por um software, aplicação ou plataforma para utilização de suas funcionalidades que podem ser uma informação contida em um banco de dados do sistema, resultado do processamento de alguma função onde se quer o valor retornado, por outros softwares, aplicações ou plataformas.

As principais vantagens de se utilizar APIs é a possibilidade de integração com outros sistemas e fazer isso de forma com que os outros sistemas não saibam como o

backend da aplicação servidora funciona, protegendo assim informações privilegiadas e tornando a aplicação mais segura, uma vez que a aplicação solicitante só recebe o que foi solicitado via API.

Já o Representational State Transfer (REST) em português significa "Transferência de Estado Representacional", se refere a um modelo com princípios e definições a ser utilizado para se projetar arquiteturas de software, baseadas em comunicação via rede. A utilização da arquitetura REST nos permite a comunicação entre aplicações distintas. Aplicações que se utilizam da arquitetura REST, são chamadas de Arquitetura que atende o padrão REST (RESTful).

Para que a arquitetura seja considerada REST e a aplicação RESTful elas tem que ter obrigatoriamente cinco características obrigatórias:

1. Cliente/servidor: Separar a arquitetura e responsabilidades em dois ambientes.

2. Stateless: Cada requisição é independente, toda requisição deve conter todas as informações necessárias para que o servidor consiga entender e processar de forma única e correta. O servidor não armazena o estado do cliente.

3. Cache: Respostas que possam ser armazenadas em cache evitando processamento desnecessário e aumentando significativamente a performance.

4. Interface uniforme: Regras para deixar os componentes o mais genéricos possíveis, para facilitar otimização ou refatoramento. Às regras são divididas entre:

    - identificação de recurso: cada recurso deve ter uma Uniform Resource Identifier (URI) específica para ser acessado.
    - representação do recurso: forma como o recurso vai ser retornado para o cliente. Geralmente essa resposta está em HyperText Markup Language (HTML), XML, JavaScript Object Notation (JSON), text-formatted data (TXT), entre outras.
    - resposta auto-explicativa: informações adicionais (metadados) na solicitação e na resposta sobre como processá-las.
    - hypermedia: retornar as informações de maneira completa de forma que com a resposta o cliente consiga navegar e ter acesso aos recursos de que precisa.

5. Sistema em camadas: A aplicação deve ter várias camadas e o cliente não deve se comunicar diretamente com servidor, e sim com um intermediário (que pode ser um load balance, proxy reverso, etc.), isso faz com que a aplicação fique modular e mais fácil de se inserir outras camadas ou alteração das existentes.

6. Código sob demanda (opcional): Execução de parte exclusiva de códigos do servidor no cliente como scripts.

Para podermos atender os requisitos do REST, garantindo uniformidade e identificação única por meio de sua URI para buscar e manipular dados, tem-se diversas maneiras de manipulação, sendo possível criá-los, atualizá-los, excluí-los, dentre outras operações. Para tal na web geralmente é usado o protocolo HTTP que fornece diversos métodos que são usados de acordo com a manipulação que se deseja fazer com o determinado recurso. A Figura 5 mostra os vários métodos (verbos) comumente usados.

Figura 5 – Métodos HTTP

GET	Obter os dados de um recurso.
POST	Criar um novo recurso.
PUT	Substituir os dados de um determinado recurso.
PATCH	Atualizar parcialmente um determinado recurso.
DELETE	Excluir um determinado recurso.
HEAD	Similar ao GET, mas utilizado apenas para se obter os cabeçalhos de resposta, sem os dados em si.
OPTIONS	Obter quais manipulações podem ser realizadas em um determinado recurso.

Fonte: (CAELUM, 2017)

2.6 Módulos Python

Para a criação da API e dos códigos que realizam às configurações foram utilizados os seguintes módulos.

2.6.1 ncclient

Ncclient NCCLIENT (2020) é uma biblioteca Python para clientes NETCONF. Seu objetivo é oferecer uma API intuitiva que mapeia de forma sensata a natureza codificada em XML de NETCONF para construções e expressões idiomáticas Python e torna mais fácil escrever scripts de gerenciamento de rede.

Outra vantagem é que está biblioteca suporta todas às operações e recursos definidos na RFC 6241, valendo ressaltar que mesmo que a biblioteca suporte todas às operações e recursos, a utilização das mesmas dependerá do que o dispositivo suporta.

Com o ncclient temos um função (manager) que nos permite informar a operação padrão e qual datastore queremos modificar, o modelo de dados, usuário e senha e passar os comandos via código python para os equipamentos de forma simples.

2.6.2 netmiko

Netmiko Byers (2020) é uma biblioteca multi vendor python desenvolvida e idealizada pelo Kirk Byers, para facilitar a conexão SSH do paramiko Paramiko (2020) a equipamentos de rede.

Com o netmiko temos um função (ConnectHandler) que nos permite informar o modelo do dispositivo, usuario e senha e passar os comando via código python para os equipamentos de forma simples e intuitiva.

2.6.3 connexion

Connexion CONNEXION (2020) é um módulo python pertencente ao micro framework Flask que lida automaticamente com solicitações HTTP definidas usando OpenAPI, suportando v2.0 e v3.0 da especificação.

O Connexion permite que você escreva essas especificações e, em seguida, mapeie os endpoints para suas funções Python. Isso é o que o torna único em relação a outras ferramentas que geram a especificação com base em seu código Python.

2.6.4 requests

É uma biblioteca python para trabalhar com requisições HTTP, informando o método que se deseja solicitar.

Neste trabalho foi usada para capturar informações adicionais diretamente da API do Netbox que não eram enviadas de maneira direta.

2.6.5 swagger

O Swagger Swagger (2020) é um conjunto de ferramentas de software de código aberto para projetar, criar, documentar e usar serviços Web RESTful, desenvolvido pela SmartBear Software que definiu a especificação OpenAPI implementada pelo swagger. Inclui documentação automatizada, geração de código e geração de casos de teste.

3 Ferramenta de documentação e orquestração

A documentação de uma infraestrutura, incluindo todos os seus componentes, serviços e respectivas configurações é indispensável para que uma empresa ou instituição consiga manter níveis elevados de disponibilidade de seus sistemas. E para isso se faz necessário uma documentação sólida, consistente e atualizada, pois localizar informações importantes e em momentos críticos, encurtar drasticamente o tempo e esforços gastos nas ações de diagnósticos e suporte de falhas na rede; planejamento de capacidades, planejamento estratégico, engenharia de redes, otimização da operação, ativação e suporte de assinantes e serviços, dentre tantas situações fundamentais para que uma empresa consiga fazer o certo e na hora certa. (Furtado (2019))

A documentação de rede e infraestrutura é tão ou mais importante do que manter o serviço em funcionamento, pois de nada adianta o ambiente estar funcionando em perfeito estado, mas apenas uma pessoa ter o conhecimento de como acessar o ambiente ou prestar manutenção, pois isso resultaria em atraso de soluções de problemas que no geral acontecem no dia a dia.(WOLFF (2017))

O cenário atual da JFRN conta com mais de quarenta ativos de rede só na capital, e esses equipamentos são documentados em uma *wiki*, onde apenas são guardados IPs dos *switches* e as conexões entre os mesmos, o que por sua vez não garante confiabilidade que os dados contidos ali estão realmente de acordo com o que está nos equipamentos, uma vez que qualquer alteração feita nos equipamentos e não cadastrada na *wiki* já faz com que elá esteja desatualizada.

Dado o cenário atual da JFRN o Netbox foi escolhido como ferramenta de documentação por se tratar de um sistema *open-source* que pode ser utilizado em *container*, pois é o modelo de novos sistemas que tem sido adotado como novo padrão pela equipe do Núcleo de Tecnologia Informação e Comunicação (NTIC) da JFRN, o que facilita sua implementação. Outro ponto relevante que o Netbox tem é a centralização da configuração de todos os equipamentos da TI e suas conexões entre si, o que facilita na procura de uma configuração especifica, caso algum sistema ou equipamento esteja apresentando falha. Podemos ainda criar VLANs, guardar o estado das interfaces dos *switches*, além disso possui uma API completa e mecanismos que possibilitam conexão com outros sistemas e com isso a automação dessas configurações, de forma a garantir que o que está salvo no Netbox é o que está realmente em uso nos *switches*.

3.1 Menus utilizados

Como o Netbox foi criado para documentar toda a rede de uma organização, ele possui vários menus e possibilidades de configuração. Para a documentação dos *switches* e automação da configuração deles na JFRN foram utilizados os seguintes menus:

1. Sites e Regiões

   - Sites: Na modelagem utilizada, os sites são os prédios da JFRN, neste primeiro momento foi criado a sede da Justiça.
   - Regions: Regiões comumente são utilizadas para definir as localidades geográficas de cada site, foi criada a região de Natal-RN.

2. Tenants

   Utilizado para representar departamentos internos da instituição que tenham algum dos dispositivos. Como no caso da JFRN todos os switches são gerenciados pelo NTIC, foi criado apenas o tenant NTIC - JFRN.

3. Devices

   É um dispositivo de rede que foi cadastrado no netbox, no nosso caso os devices são os switches da JFRN.

   - Devices types: Representa um modelo e fabricante de um equipamento cadastrado no netbox e uma vez cadastrado podem ser criados *devices* a partir de um modelo específico. Foram cadastrado todos os modelos de switches da instituição.
   - Interfaces dos devices: Representa às portas dos switches, que podem ser conectadas entre si, associadas a um IP (interface virtual) ou representar uma porta que será utilizada por um equipamento (PCs, APs). Neste ultimo caso, especificamos qual é o tipo de porta (access, tagged, tagged-all, null) e qual a vlan que está passando nessa porta, para que o dispositivo esteja com as configurações corretas.

   A Figura 6 ilustra a porta de um *switch* sendo modificada para *access* na VLAN 3000 onde um computador será conectado.

Figura 6 – Porta access na vlan 3000

Fonte: Autor

4. IPAM

   - Prefixes: Define as redes e sub-redes em uso na JFRN em notação CIDR.
   - IP Address: Identificação unica na rede, associamos os IPs a interfaces virtuais dos *switches* para serem acessados no IP daquela vlan ao qual ele faz parte.

5. Vlans

   Uma VLAN define um domínio isolado de camada 2 identificado por um nome e um ID numérico. Cada VLAN pode ser atribuído a um grupo.

3.2 Webhook

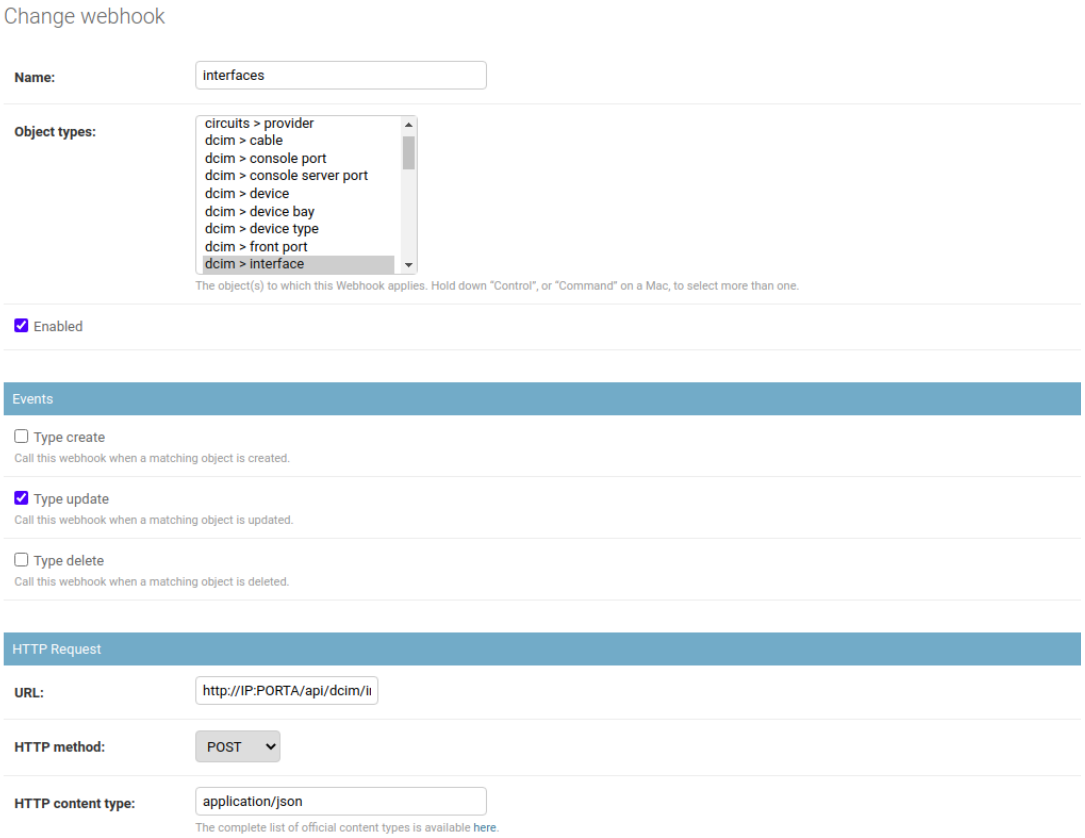
Os *webhooks* são mecanismos para para transmitir a algum sistema externo uma alteração que ocorreu no Netbox.(NETBOX, 2020)

Devido a este mecanismo podemos automatizar a configuração dos switches, uma vez que, ao acontecer um evento do tipo *update* em uma interface, será feita uma chamada na API REST informando que houve uma modificação naquela porta e passando os dados que foram modificados. Estes dados incluem a porta modificada, *switch*, IP do *switch*, VLANs e qual modo a porta está operando, de modo que é possível automatizar via código às configurações dos *switches*.

O *webhook* faz uma solicitação HTTP ao *webservice* ou API cadastrada nele passando os dados modificados para o sistema que recebe essas informações em formato de texto, para que o ambiente fique mais seguro, foi adotado o uso de *container* tanto para o Netbox quanto para o *webservice* de forma que os dois ficam isolados em uma rede interna.

A Figura 7 mostra a criação dos *webhooks* na interface de administração do netbox.

Figura 7 – Criação webhook



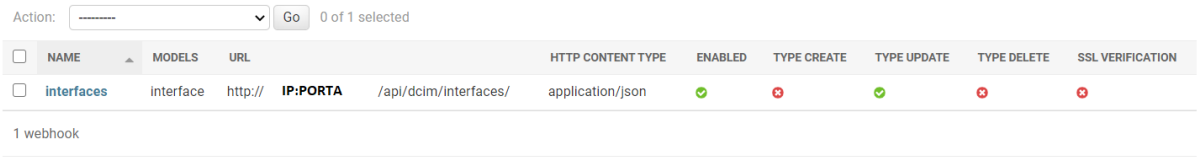
Fonte: Autor

O *webhook* utilizado neste trabalha tem a seguinte configuração:

- Object Type: dcim > interface
  Informa onde o evento ocorrerá
- Events: update
  Informa em qual tipo de evento o webhook deve ser chamado/disparado
- HTTP Request:
  - URL: url do webservice, com a URI específica a ser chamada
  - HTTP method: POST (atualização de uma interface)
  - HTTP content type: application/json (tipo de dado a ser enviado)

Após a criação o webhook aparece no menu de administração no formato segundo a Figura 8:

Figura 8 – Webhook criado



Fonte: Autor

4 Implementação da solução

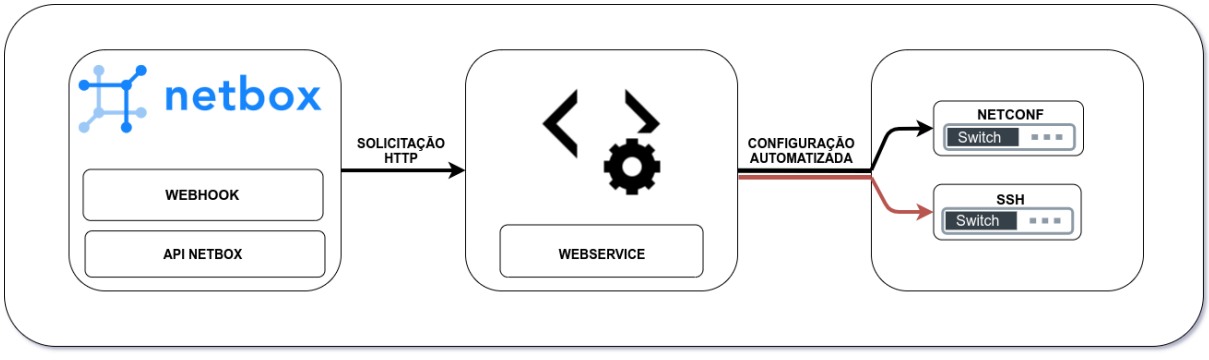
Dada a grande quantidade de dispositivos da JFRN faz-se necessário uma melhor gerência dessa rede, e para isso foi dado o primeiro passo ao se utilizar protocolos como o NETCONF que foi criado com o intuito de gerenciar a rede de forma simples, está solução visa sanar o problema da configuração manual dos ativos de rede (*switches*) ao usar a documentação do ambiente em uma aplicação que ao ser atualizada pode chamar outros sistemas que por sua vez se comunicam com os ativos de rede e aplicam as atualizações nos mesmos.

Com o Netbox sendo o responsável por documentar a rede, a solução será um *webservice*,que será chamado pelo Netbox, ou seja, uma vez que exista uma atualização em alguma das interfaces cadastradas, o *webhook* do Netbox faz uma solicitação HTTP ao *webservice* informando o nome do dispositivo que foi atualizado, a interface, quais VLANs foram cadastradas, protocolo de gerência que o dispositivo aceita e o tipo de porta que a interface tem.

O *webservice* por sua vez, recebe essas informações na função interfaces a qual é mapeada no *endpoint* do *webservice* e a mesma chama funções auxiliares que tem o papel de filtrar os dados e armazena-los em memória e/ou realizar chamadas na API do Netbox caso seja necessário dados extras que não foram passados na solicitação HTTP. Uma vez que os dados foram filtrados são chamadas funções que são responsáveis por se conectar ao *switch* onde será passada a configuração de forma automatizada. Caso o switch aceite o protocolo de gerência do tipo SSH as configurações são aplicadas em formato de comandos diretos no dispositivo, caso o *switch* aceitar o protocolo de gerência NETCONF então será feita uma solicitação via RPC do tipo *edit-config* onde serão feitas alterações no datastore com os valores que foram armazenados em memória pela aplicação.

A Figura 9 retrata de forma geral o fluxo desde o Netbox até os *switches*.

Figura 9 – Fluxograma automação



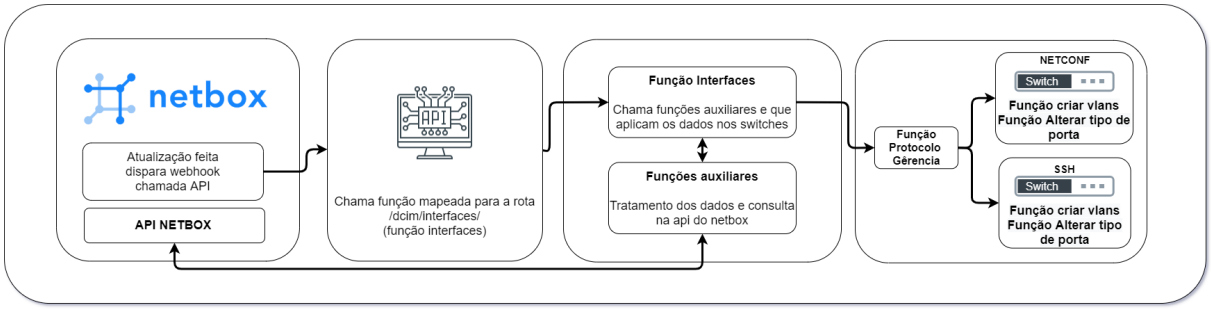
Fonte: Autor

4.1 API para recebimento dos dados do Netbox

Com a visão geral do funcionamento da solução, a partir dessa sessão e nas seguintes vamos abordar cada uma das partes integrantes do *webservice* e suas relações com o Netbox.

A Figura 10 retrata as partes integrantes do webservice de forma detalhada e o fluxo desde o Netbox até os *switches*.

Figura 10 – Fluxograma automação *webservice* detalhado



Fonte: Autor

Foi definida uma API para ser chamada pelo webhook do netbox de forma a receber os dados do que foi alterado em uma determinada interface do switch selecionado.

Para a criação da API foram utilizadas as bibliotecas do python swagger e connexion. Antes do connexion começar a tratar às requisições HTTP, o mesmo lê um arquivo descritivo do swagger com às especificações da API, para saber quais endpoints chamar.

4.1.1 Definição API

A API utiliza o método *POST*, e uma vez que o *webhook* chamar o endpoint da API a mesma receberá os dados e encaminhará para a função no endpoint que tratará esses dados e enviará às configurações para o *switch*. Após isso o *POST* é confirmado com sucesso se a configuração for aplicada, caso contrário retornará o erro encontrado ao atualizar a porta do *switch*.

A URI de acesso à API foi especificada de forma estática uma vez que a mesma só precisa receber os dados vindos do *webhooks*.

4.1.2 Endpoint API

Foi criado para ser acessada/chamada pelo HTTP *Request* no *webhook* do Netbox e receber os dados atualizados e enviar para a função interfaces, que está no arquivo func_int na pasta API que está indicada no *operationId*.

O bloco de Código 4.1 mostra a criação do *endpoint* e a URI a ser chamada, que relaciona o *webhook* com a API. Uma vez que aconteceu uma atualização e o *webhook* chame

o *webservice* o mesmo procura qual path (URI) está cadastra no *webhook* ao encontrar o "/dcim/interfaces" procura o metódo que foi executado(*POST*) e a qual função (interfaces) deve ser chamada para a execução do metódo.

Código 4.1 – Criação Endpoint

```
openapi: 3.0.0
info:
  description: Swagger file
  version: "1.0.0"
  title: Automacao Switches

servers:
    - url: http://ip:porta/api

paths:
  /dcim/interfaces/:
    post:
      operationId: api.func_int.interfaces
      tags:
        - DCIM
      summary:  Captura Interface Editada
      description:  Captura Interface Editada

      responses:
        204:
          description: 204 OK
          content:
            text/plain:
              schema:
                type: string
```

4.2 Implementação da aplicação flask

A aplicação feita usando o connexion e o flask instância a aplicação e define o host e porta que receberão às conexões. O ip pode ser estático, ou qualquer ip usado pelo host, ressaltando que o ip neste caso independente de qual for deve ser o mesmo utilizado dentro do HTTP Request que o webhook chamará.

O bloco de Código 4.2 mostra a criação do servidor, onde podemos observar na linha 6 que a inserção do arquivo contendo os *endpoints* do *webservice*, responsável por ser chamado pelo *webhook*.

Código 4.2 – Criação servidor aplicação

```
from flask import render_template
import connexion

app = connexion.App(__name__, specification_dir='./')

app.add_api('swagger.yml')

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000, debug=True)
```

4.3 Implementação da função mapeada no endpoint

A seguir será descrita a função que recebe os dados do netbox (função mapeada no *endpoint* pelo *operationId*).

- interfaces

  Está função recebe os dados vindos do Netbox em uma estrutura de dados do tipo dicionario e chama funções auxiliares para tratar e armazenar as informações importantes sobre a configuração da porta que foi atualizada, estas informações serão utilizadas por outras funções que são responsáveis por se conectar ao dispositivo de rede e aplicar esses dados no equipamento.

  O Código 4.3 mostra a função interfaces e que por sua vez recebe as informações na variavel dict_interfaces, filtra os dados e armazena na variável dados_int e por fim chama a função add_dell_vlan_port que recebe os dados filtrados e chama outras funções internamente para coleta de informações adicionais e aplica as configurações no dispositivo específico.

Código 4.3 – Função Interfaces

```
def interfaces():

    dict_interfaces = request.get_data(as_text=True)
    dados_int = func_aux.dados_interface(dict_interfaces)
    func_aux.add_del_vlan_port(dados_int[2], dados_int[1], func_aux.
        get_ip_sw(dados_int[2]), dados_int[0], func_aux.get_vlans(
        dict_interfaces), dados_int[3])
```

4.4 Implementação das funções auxiliares

A Tabela 3 mostra as funções auxiliares e seus papeis

Tabela 3 – Funções Auxiliares

Função	Descrição
dados_interface	Retorna os dados da interface modificada
get_ip_sw	Retorna IP *switch*
get_vlans	Retorna VLANs usadas em um interface

Fonte: Produzido pelo autor

A seguir as funções auxiliares responsáveis por tratar os dados, bem como armazená-los em memória para uso posterior, serão detalhadas.

- dados_interface

  A função "dados_interface" tem o papel de tratar o dicionário python e armazenar os dados da interface que foi alterada, armazenando e retornando os valores para serem usados para a automação da configuração dos *switches*, como mostra o boco de Código 4.4.

Código 4.4 – Função Dados Interface

```
1  def dados_interface(dict_interfaces):
2
3      dados_interface = json.loads(dict_interfaces)
4      data_interface = dados_interface['data']
5      interface = data_interface['name']
6      url_device = data_interface['device']['url']
7      desc_interface = data_interface['description']
8
9      try:
10         port_type = data_interface['mode']['value']
11     except TypeError:
12         port_type = 'null'
13
14     return [interface, port_type, url_device, desc_interface]
```

Um ponto importante a destacar é que se a porta for limpa, ou seja seus dados excluídos, na hora de armazenar daria um erro pois não podemos ficar com uma variável vazia para ser aplicada no *switch*. Para resolver essa situação, foi adicionado ao código às linhas de 9 a 12 como mostra o bloco de Código 4.4, a fim de garantir que se a porta está sem configuração cadastrada, deve-se armazenar o valor *"null"* no tipo da porta garantindo com que na hora que for aplicada ao *switch* aquela porta fique sem configurações específicas, utilizando-se a configuração padrão do equipamento.

- get_ip_sw

  Essa função tem o papel de buscar o IP do *switch* no qual a porta foi alterada, a mesma é importante devido ao fato de que o IP será usado na hora da automação da configuração, e precisou ser criada pois os dados que vem do *webhook*, traz apenas o nome do dispositivo e não seu IP.

Código 4.5 – Função Get IP switch

```
1 def get_ip_sw(url_device):
2     get_dic_device = requests.get('http://{}{}'.format(IP_NB,
          url_device),headers=HEADERS,timeout=5)
3     dados_device = get_dic_device.json()
4     ip_sw = (dados_device['primary_ip']['address'].split('/')[0])
5     return ip_sw
```

  Podemos observar que pelo trecho de Código 4.5 que para obtermos o IP foi utilizada a função *requests* para fazer uma chamada na API do Netbox, informando o IP do Netbox e a Uniform Resource Locator (URL)/URI do dispositivo contendo seu *ID*.

- get_vlans

  Está função tem o papel de tratar o dicionário python e separar as VLANs que estão em uso em uma determinada porta. Às portas dos switches podem assumir três tipos de funções (access, hybrid e trunk), então para cada tipo de porta às VLANs vão ser salvas de forma diferente, uma vez que dependendo do tipo de porta pode ter uma vlan ou várias VLANs.

  Como essas vlans vão ser criadas nos switches, é importante ter todos os dados como nome, id e descrição da vlan. Como o dicionário não traz a descrição então foi necessário fazer uma chamada na API com a URI e id da vlan específica para coletar esse dado.

  O bloco de Código 4.6 mostra o trecho de código para captura das *VLAN tagged* e a chamada na API para coleta da descrição.

Código 4.6 – Trecho da função Get Vlans - *Tagged*

```
elif (port_type) == 'tagged':
    untagged_vlan = data_interface['untagged_vlan']
    url_vlan_untagged = untagged_vlan['url']
    name_vlan_untagged = untagged_vlan['name']
    vid_vlan_untagged = untagged_vlan['vid']
    dict_untagged = requests.get('http://{}{}'.format(IP_NB,
        url_vlan_untagged),headers=HEADERS,timeout=5)
    description_untagged = dict_untagged.json()['description']

    vlan_untagged = [name_vlan_untagged, vid_vlan_untagged,
        description_untagged]

    tagged_vlans = data_interface['tagged_vlans']
    port_type = data_interface['mode']['value']
    vlans_tagged = []
    dados_tagged = [{'name': tag_vlans['name'], 'vid': tag_vlans['vid'
        ]} for tag_vlans in tagged_vlans]

    for  desc in tagged_vlans:
        dict_tagged = requests.get('http://{}{}'.format(IP_NB, desc['
            url']),headers=HEADERS,timeout=5)
        vlans_tagged.append({'vid': desc['vid'], 'description':
            dict_tagged.json()['description']})

    for  i in range(len(tagged_vlans)):
        if vlans_tagged[i]['vid'] == dados_tagged[i]['vid']:
            vlans_tagged[i].update(dados_tagged[i])

    return [vlan_untagged, vlans_tagged]
```

4.5 Implementação das funções de envio de configurações

Atualmente a JFRN conta com diversos modelos de *switches*, alguns antigos e outros mais modernos. Devido a isso, para que pudéssemos fazer uso da automação em todos os modelos a solução encontrada foi a utilização de dois protocolos para comunicação com os ativos de rede, sendo o SSH para os modelos mais antigo, onde abre-se uma conexão SSH com o switch e aplica-se às configurações vindas do netbox diretamente no terminal do dispositivo, utilizando-se dos comandos nativos do dispositivos dentro das funções; e para os switches mais modernos o NETCONF, usando também da conexão SSH para trafegar as informações, mas por sua vez aplicando às configurações diretamente no *datastore* do servidor NETCONF rodando no switch via mensagem NETCONF em XML

do tipo *edit-config*.

Essas funções são responsáveis pela automação de fato, pois com os dados do netbox tratados e salvos às funções a seguir se encarregam de estabelecer conexão com os switches e aplicar as devidas configurações na porta do *switch* de forma automatizada sem que exista intervenção humana no processo.

A Tabela 4 mostra as funções responsáveis por receber os dados que foram tratados pelas funções auxiliares, estabelecer conexão com o *switch*, criar as VLANs necessárias e alterar o tipo de porta, de acordo com o que foi atualizado na documentação vigente no Netbox.

Tabela 4 – Funções de envio de configurações

Função	Descrição
get_proto_gerencia	Captura protocolo aceito pelo *switch*
create_vlan	Cria vlan no dispositivo
port_type	Modifica o tipo de porta no *switch*

Fonte: Produzido pelo autor

- Protocolo de gerência

  Como se adotou dois protocolos para utilização devido aos switches mais antigos não terem suporte ao NETCONF, fez-se necessário a criação de uma função para capturar qual protocolo de gerência é suportado pelo switch ao qual a porta foi alterada.

  Como podemos observar no bloco de Código 4.7, utilizamos novamente a biblioteca *requests* para ir até a API do Netbox e nos retornar qual protocolo foi utilizado; primeiramente para sabermos o modelo do dispositivo (*Device Type*) do dispositivo e depois outra chamada para identificar qual o protocolo associado ao respectivo modelo. Com o protocolo reconhecido pode-se chamar as funções corretas para criação das VLANs e alteração do tipo de porta.

Código 4.7 – Função Protocolo de Gerência

```
def get_proto_gerencia(url_device):
    get_dic_device = requests.get('http://{}{}'.format(IP_NB,
        url_device),headers=HEADERS,timeout=5)
    dados_device = get_dic_device.json()
    url_device_type = (dados_device['device_type']['url'])
    get_dic_dt = requests.get(url_device_type, headers=HEADERS,
        timeout=5)
    dados_dt =  get_dic_dt.json()
    protocolo_gerencia = (dados_dt['custom_fields']['protocol']['label
        '])
    return protocolo_gerencia
```

- Criação das vlans

  Uma vez que o protocolo de gerência foi estabelecido, o próximo passo é garantir que as VLANs que serão utilizadas na porta existam no *switch*. Uma das limitações encontradas no netbox é que ele trata as VLANs como itens separados dos switches, isso impede por sua vez que quando uma vlan é criada no Netbox já seja criada nos switches também, para garantir que a vlan que será usada sempre exista no *switch*. A solução foi criar funções que criam a VLAN ou VLANs no *switch* antes de utilizar a(s) mesma(as) em uma das portas.

  Os trechos de Código 4.8 e 4.9 mostram a criação de uma vlan no dispositivo para garantir que a mesma exista antes de ser utilizada.

  O Código 4.8 é chamado para criar a VLAN em um *switch* que utiliza o protocolo SSH.

Código 4.8 – Função criação vlan com SSH

```
def create_vlan(ip, user, password, port_link_type, dados_vlans):

    device_kargs = {'device_type': DEVICE_TYPE, 'use_keys': False, '
        allow_agent': False, 'conn_timeout': 30}
    device_kargs['ip'] = ip
    device_kargs['username'] = user
    device_kargs['password'] = password

    if port_link_type == 'access':

        a = 'vlan {number_of_vlan}'.format(number_of_vlan=dados_vlans
            [1])
        b = 'name {name_of_vlan}'.format(name_of_vlan=dados_vlans[0])
        c = 'description {description_of_vlan}'.format(
            description_of_vlan=dados_vlans[2])
        config_commands = [a, b, c]

        with ConnectHandler(**device_kargs) as net_connect:
            net_connect.send_config_set(config_commands)
            net_connect.save_config()
```

O Código 4.9 é chamado para criar a VLAN em um *switch* que utiliza o protocolo NETCONF

Código 4.9 – Função criação vlan com Netconf

```
def create_vlan(ip, user, password, port_link_type, dados_vlan):

    device_kargs = {''port': 830, 'hostkey_verify': False, '
       allow_agent': False,
                    'look_for_keys': False, 'device_params':
                       DEVICE_PARAMS}
    device_kargs['host'] = ip
    device_kargs['username'] = user
    device_kargs['password'] = password

    if (port_link_type) == '1':  # Access
      xml_create_vlan = '''
        <nc:config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
          <top xmlns="http://www.hp.com/netconf/config:1.0">
            <VLAN>
              <VLANs>
                <VLANID>
                  <ID>{number_of_vlan}</ID>
                  <Description>{description_of_vlan}</Description>
                  <Name>{name_of_vlan}</Name>
                </VLANID>
              </VLANs>
            </VLAN>
          </top>
        </nc:config>'''.format(number_of_vlan=dados_vlan[1],
                    description_of_vlan=dados_vlan[2], name_of_vlan=
                       dados_vlan[0])

        with manager.connect(**device_kargs) as netconf_manager:
            netconf_manager.edit_config(
                target='running', config=xml_create_vlan,
                   default_operation='merge')
```

- Modificação do tipo de porta

  Após garantir que as VLANs que serão utilizadas existam o próximo passo feito pelo *webservice* é aplicar o tipo de porta (access, hybrid, trunk) que se quer no switch e as VLANs que serão usadas. Para isso foram criadas funções para aplicar o tipo de porta documentada no Netbox, na porta do switch.

  Para que se consiga modificar o tipo de porta, a porta deve estar sem configuração ou no tipo de porta *access* (padrão dos *switches* da JFRN), e por isso às funções

também garantem que a porta está sem configurações antes de aplicar uma nova configuração.

O Código 4.10 é chamado para alterar o tipo de porta para *access* em um *switch* que utiliza o protocolo SSH.

Código 4.10 – Função modifica tipo de porta com SSH

```
1  def access(ip, user, password, interface, port_link_type, dados_vlans,
       description_of_port):
2
3      device_kargs = {'device_type': DEVICE_TYPE, 'use_keys': False, '
           allow_agent': False}
4      device_kargs['ip'] = ip
5      device_kargs['username'] = user
6      device_kargs['password'] = password
7
8      a = 'interface {interface}'.format(interface=interface)
9      b = 'undo port link-type'
10     c = 'port link-type {port_link_type}'.format(port_link_type=
           port_link_type)
11     d = 'port {port_link_type} vlan {number_of_vlan}'.format(
           port_link_type=port_link_type, number_of_vlan=dados_vlans[1])
12     e = 'description {description_of_port}'.format(description_of_port
           =description_of_port)
13     config_commands = [a, b, c, d, e]
14
15     with ConnectHandler(**device_kargs) as net_connect:
16         net_connect.send_config_set(config_commands)
17         net_connect.save_config()
```

O Código 4.11 é chamado para alterar o tipo de porta para *access* em um *switch* que utiliza o protocolo NETCONF

Código 4.11 – Função modifica tipo de porta com Netconf

```
def access(ip, user, password, interface, port_link_type, dados_vlan,
    desc_of_port):

    xml_port_type_acc = '''
        <nc:config xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0">
            <top xmlns="http://www.hp.com/netconf/config:1.0">
                <Ifmgr>
                    <Interfaces>
                        <Interface>
                            <IfIndex>{interface}</IfIndex>
                            <LinkType>{port_link_type}</LinkType>
                            <PVID>{number_of_vlan}</PVID>
                            <Description>{desc_of_port}</Description>
                        </Interface>
                    </Interfaces>
                </Ifmgr>
                <VLAN>
                    <AccessInterfaces>
                        <Interface>
                            <IfIndex>{interface}</IfIndex>
                            <PVID>{number_of_vlan}</PVID>
                        </Interface>
                    </AccessInterfaces>
                </VLAN>
            </top>
        </nc:config>
    '''.format(interface=interface, port_link_type=port_link_type,
            number_of_vlan=dados_vlan[1], desc_of_port=desc_of_port)

    device_kargs = {'port': 830,
        'hostkey_verify': False,
        'allow_agent': False,
        'look_for_keys': False,
        'device_params': DEVICE_PARAMS}

    device_kargs['host'] = ip
    device_kargs['username'] = user
    device_kargs['password'] = password

    with manager.connect(**device_kargs) as netconf_manager:
        netconf_manager.edit_config(target='running', config=
            xml_port_type_acc, default_operation='merge')
```

- Registro das configurações aplicadas

  Uma vez que a configuração é enviada para o dispositivo, a aplicação recebe um *status code* HTTP 204 informando que a configuração foi aplicada com êxito. Em contra partida caso a configuração não seja executada de forma correta o webservice recebe como resposta um *status code* HTTP 500 informando que a ação não foi executada com sucesso. Nos *switches* SSH é retornado em texto puro os comandos aplicados no terminal do *switch* e qual comando deu erro ou se estourou o tempo de configuração; já nos *switches* NETCONF é retornada uma mensagem NETCONF em formato XML informando o erro ocorrido ao atualizar o datastore.

A partir dos Códigos 4.8, 4.9, 4.10 e 4.11, podemos notar que com o protocolo NETCONF apesar dos códigos ficarem maiores, é bem mais simples de se configurar os *switches* pois é uma mensagem em XML e não precisamos saber os comandos que o *switch* aceita, e apenas modificamos os dados armazenados no datastore do mesmo, em contra partida nos switches que só aceitam SSH temos que escrever os comandos de forma correta dentro do código para que ele aplique as devidas configurações. Isso faz toda diferença levando em consideração que a instituição tem vários tipos de dispositivos, pois para os que aceitam apenas SSH precisaríamos criar funções distintas para cada *switch* com os comandos corretos que os mesmos aceitam, já se os *switches* aceitam o NETCONF e utilizam os mesmos modelos de dados, a mesma função servirá para todos eles.

5 Conclusão

Neste trabalho foi proposto um novo modelo de gerência para a rede da JFRN uma vez que o ambiente de redes é grande e complexo e o trabalho de gerência dessa rede é feito de forma manual e sem processos de automação, faz-se necessário automatizar os processos de configuração desse ambiente, a fim de garantir maior agilidade em novas configurações ou no caso da necessidade de reconstrução da rede devido a alguma falha inesperada, o que consumiria muito mais tempo em um processo todo manual, tanto para novas configurações ou reconstrução do ambiente.

Com o processo automatizado temos vários benefícios tais quais: redução de erros, ganho de eficiência por parte da equipe e redução do conhecimento necessário para configuração, uma vez que com os processos automatizados não se faz necessário saber comandos específicos dos diferentes tipos de equipamentos ou mesmo se deslocar até o ativo.

Para tal foi utilizado um protocolo padronizado de gerência, o NETCONF que se mostrou simples e eficiente por ter uma linguagem mais próxima da humana, utilizando o XML para aplicar suas configurações e não comandos que podem e muitas vezes são diferentes dentro das opções de equipamentos de um mesmo fabricante.

Utilizando-se do NETCONF, o escopo deste trabalho focou na criação de VLANs e atualização de portas dos *switches* da JFRN por ser um trabalho recorrente na instituição. Para tal foi utilizado o Netbox como ferramenta de documentação dos *switches* uma vez que seu uso possibilita integração com outras ferramentas através da sua API e *webhook*, e com isso podemos automatizar as configurações desses equipamentos.

Além disso foi criado um webservice para haver integração com o Netbox e receber as atualizações da documentação de forma automática via solicitação HTTP contendo os dados que foram alterados. Dentro do webservice foram implementadas diversas funções que tinham o papel de receber os dados vindos do Netbox, tratar esses dados, e uma vez os dados tratado chamar outras funções responsáveis por se comunicarem com os switches via SSH e NETCONF de forma automática, não se fazendo necessária intervenção humana, trazendo não só a automação dessas configurações mas também agilidade, redução de erros, eficiência e por fim segurança pois não haverá entradas e saídas continuas nas salas dos equipamentos de rede.

Espera-se que com o uso do Netbox e do webservice de forma integrada o trabalho diário de atualização de portas dos equipamentos seja facilitado e que a documentação esteja sempre de acordo com o que está em uso nos equipamentos, uma vez que essas informações estão centralizadas no Netbox e é a partir dele que os equipamentos serão atualizados. Além disso a utilização do sistema e do webservice é o primeiro passo no caminho a uma melhor gerência da rede através de protocolos como o NETCONF que

facilitem o trabalho diário de configuração.

5.1 Trabalhos Futuros

Durante a execução deste trabalho foram identificadas as seguintes sugestões para trabalhos futuros:

- Cadastro dos ativos no netbox de forma automatizada.
- Substituir o mecanismo de login. Atualmente feito passando usuário e senha, pela utilização de um par de chaves.
- Criar mecanismo de atualização de varias portas, para configuração de um equipamento novo, por exemplo.

Referências

BYERS, K. *Netmiko*. 2020. Disponível em: <https://github.com/ktbyers/netmiko/blob/develop/README.md>. Citado na página 26.

CAELUM. *REST: Princípios e boas práticas*. 2017. Disponível em: <https://blog.caelum.com.br/rest-principios-e-boas-praticas/>. Citado na página 25.

CISCO. *Introduction to NETCONF*. 2018. Disponível em: <http://yang.ciscolive.com/pod0/labs/lab2/lab2-m2>. Citado na página 22.

CLAISE JOE CLARKE, J. L. B. *Network Programmability with YANG: The Structure of Network Automation with YANG, NETCONF, RESTCONF, and gNMI*. 1. ed. Addison-Wesley Professional, 2019. ISBN 978-0135180396. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=a6609e43bc91a3b7d65213eeb0d91394>. Citado na página 18.

CONNEXION. *ncclient: Python library for NETCONF clients*. 2020. Disponível em: <https://connexion.readthedocs.io/en/latest/>. Citado na página 26.

EDELMAN SCOTT S. LOWE, M. O. J. *Network Programmability and Automation: Skills for the Next-Generation Network Engineer*. 1. ed. O'Reilly Media, 2016. ISBN 1491931256,9781491931257. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=47ed47e2a29211455fd26ba56d80f93c>. Citado na página 16.

FURTADO, L. *Boas Praticas para a Documentaçao de Infraestruturas de Redes e Servicos do Provedor*. 2019. Disponível em: <https://wiki.brasilpeeringforum.org/w/Boas_Praticas_para_a_Documentacao_de_Infraestruturas_de_Redes_e_Servicos_do_Provedor>. Citado na página 28.

IETF. *Configuring Networks and Devices with Simple Network Management Protocol (SNMP)*. [S.l.], 2003. 1-83 p. Disponível em: <https://tools.ietf.org/html/rfc3512>. Citado na página 14.

IETF. *The Secure Shell (SSH) Authentication Protocol*. [S.l.], 2006. 1-17 p. Disponível em: <https://tools.ietf.org/html/rfc4252>. Citado na página 17.

IETF. *The Secure Shell (SSH) Connection Protocol*. [S.l.], 2006. 1-24 p. Disponível em: <https://tools.ietf.org/html/rfc4254>. Citado na página 17.

IETF. *The Secure Shell (SSH) Protocol Architecture*. [S.l.], 2006. 1-30 p. Disponível em: <https://tools.ietf.org/html/rfc4251>. Citado na página 17.

IETF. *The Secure Shell (SSH) Transport Layer Protocol*. [S.l.], 2006. 1-32 p. Disponível em: <https://tools.ietf.org/html/rfc4253>. Citado na página 17.

IETF. *Network Configuration Protocol (NETCONF)*. [S.l.], 2011. 1-113 p. Disponível em: <https://tools.ietf.org/html/rfc6241>. Citado na página 18.

IETF. *Network Configuration Protocol (NETCONF)*. [S.l.], 2016. 1-217 p. Disponível em: <https://tools.ietf.org/html/rfc7950>. Citado na página 19.

LACERDA, F. C. de. *Uma proposta de arquitetura para o protocolo NETCONF sobre SOAP*. Dissertação (Mestrado) — Universidade Estadual de Campinas, 2007. Citado 2 vezes nas páginas 14 e 23.

MASSE, M. *REST API Design Rulebook*. O'Reilly Media, 2011. ISBN 1449310508,9781449310509. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=4bc6e7afe80cc98a5e0b112ef3f349f8>. Citado na página 23.

NCCLIENT. *ncclient: Python library for NETCONF clients*. 2020. Disponível em: <https://ncclient.readthedocs.io/en/latest/index.html>. Citado na página 26.

NETBOX. *Netbox*. 2020. Disponível em: <https://netbox.readthedocs.io/en/stable/>. Citado 2 vezes nas páginas 16 e 30.

OKASHA karim. *Network Automation and the Rise of NETCONF*. 2017. Disponível em: <https://medium.com/@k.okasha/network-automation-and-the-rise-of-netconf-e96cc33fe28>. Citado na página 19.

OPENCONFIG. *OpenConfig*. 2016. Disponível em: <https://www.openconfig.net/>. Citado na página 23.

PARAMIKO. *Paramiko*. 2020. Disponível em: <http://www.paramiko.org/>. Citado na página 26.

RATAN, A. *Practical Network Automation: Leverage the power of Python and Ansible to optimize your network*. 1. ed. Packt Publishing, 2017. ISBN 1788299469, 978-1788299466. Disponível em: <http://gen.lib.rus.ec/book/index.php?md5=05cfd45fd7d7ff11e2c52294bd68d2e3>. Citado na página 16.

SWAGGER. *swagger definition*. 2020. Disponível em: <https://swagger.io/>. Citado na página 27.

TELOKEN, R. *Gerenciamento de Configuração de Dispositivos de Rede através de NETCONF e Web Services*. Dissertação (Mestrado) — UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2006. Citado na página 14.

WIKIWAND. *NETCONF*. 2018. Disponível em: <https://www.wikiwand.com/en/NETCONF>. Citado na página 20.

WOLFF, M. *A importância da Documentação de Rede*. 2017. Disponível em: <https://penseemti.com.br/artigos/a-importancia-da-documentacao-de-rede/>. Citado na página 28.

YLONEN, T. *SSH (Secure Shell)*. 2020. Disponível em: <https://www.ssh.com/ssh/>. Citado na página 18.