UNIVERSIDADE FEDERAL DE SANTA MARIA
COLÉGIO TÉCNICO INDUSTRIAL DE SANTA MARIA
CURSO SUPERIOR DE TECNOLOGIA EM REDES DE COMPUTADORES

ESTUDO E IMPLEMENTAÇÃO DE SOLUÇÕES PARA AUTOMAÇÃO DE DISPOSITIVOS DE REDE

TRABALHO DE CONCLUSÃO DE CURSO

DANIEL CRISTIANO MENZEN

Santa Maria, RS, Brasil
2015

ESTUDO E IMPLEMENTAÇÃO DE SOLUÇÕES PARA AUTOMAÇÃO DE DISPOSITIVOS DE REDE

Daniel Cristiano Menzen

Trabalho apresentado ao Curso de Graduação em Tecnologia em Redes de Computadores, Área de concentração em Infraestrutura de Redes, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Tecnólogo em Redes de Computadores.**

Orientador: Prof. Me. Tiago Antônio Rizzetti

Santa Maria, RS, Brasil

2015

Universidade Federal de Santa Maria
Colégio Técnico Industrial de Santa Maria
Curso Superior de Tecnologia em Redes de Computadores

A Comissão Examinadora, abaixo assinada, aprova a Monografia

ESTUDO E IMPLEMENTAÇÃO DE SOLUÇÕES PARA AUTOMAÇÃO DE DISPOSITIVOS DE REDE

Elaborada por,
Daniel Cristiano Menzen

Como requisito parcial para obtenção do grau de
Tecnólogo em Redes de Computadores

COMISSÃO EXAMINADORA
Tiago Antônio Rizzetti, Me. (Presidente/Orientador)

Renato Preigschadt de Azevedo, Me. (UFSM)

Tarciso Ceolin Junior, Me. (UFSM)

Santa Maria, 03 de dezembro de 2014

RESUMO

Monografia
Curso Superior de Tecnologia em Redes de Computadores
Universidade Federal de Santa Maria

ESTUDO E IMPLEMENTAÇÃO DE SOLUÇÕES PARA AUTOMAÇÃO DE DISPOSITIVOS DE REDE

AUTOR: DANIEL CRISTIANO MENZEN
ORIENTADOR: TIAGO ANTÔNIO RIZZETTI
Santa Maria, 03 de dezembro de 2015.

O gerenciamento das redes de computadores está cada vez mais complexo devido à quantidade de equipamentos de diversos fabricantes e seus respectivos softwares proprietários. Equipamentos estes que normalmente possuem apenas uma forma de serem administrados, que é através da *interface* de linha de comando. Isso faz com que a administração da rede seja complexa, pois os administradores possuirão redes heterogêneas e necessitarão de um vasto conhecimento sobre diferentes equipamentos. Com isso, mesmo as mais básicas das tarefas de administração podem se tornar dispendiosas, fazendo com que o trabalho seja repetitivo e utilize todo o tempo, impedindo a pesquisa e criação de melhores soluções que possam evoluir o gerenciamento da rede. Devido a isso, é necessário que se utilizem softwares capazes de administrar diversos dispositivos e automatizar tarefas repetitivas. Este trabalho apresenta inicialmente o estudo de algumas soluções de automação utilizando equipamentos disponíveis no laboratório do Curso de Redes de Computadores. Além disso, é apresentada a criação de uma interface gráfica de usuário, responsiva para utilização tanto em computadores quanto *smartphones*, capaz de criar *VLANs* em dispositivos *CISCO* utilizando uma biblioteca *Python* e *SSH*. Esta mesma interface também coleta informações dos dispositivos e as apresenta de forma clara ao administrador.

Palavras-chave: automação de redes

ABSTRACT

Monograph
Technology in Computer Network's Degree
Universidade Federal de Santa Maria

STUDY AND IMPLEMENTATION OF SOLUTIONS FOR NETWORK DEVICES AUTOMATION

AUTHOR: DANIEL CRISTIANO MENZEN
SUPERVISOR: TIAGO ANTÔNIO RIZZETTI
Santa Maria, 3th December, 2015.

The management of computer networks has become more complex due the amount of different equipments from different manufacturers and their own proprietary software. These equipments normally have just one management interface, which is the command line interface. Because of this the management of networks is complex. Managers will face a heterogeneous network and will need a wide knowledge about all different equipments. The most basic management tasks could be repetitive and time consuming, so managers would spend less time with research and development of better ways to improve their network management. Regarding this, is necessary to have softwares capable of manage various devices and automate repetitive tasks. This work presents a study of some of these automation tools using the equipments available at the testing lab of the course of Technology in Computer Networks from the Universidade Federal de Santa Maria. In addition, it presents the creation of a responsive web graphical user interface for use on computers and smartphones. This interface is capable to create VLANs on CISCO devices using a Python library and the SSH protocol. Furthermore, the interface collects data from the devices and presents it to the administrator in a web page.

Keywords: network automation

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

ANSIBLE	*Network automation tool*
API	*Application Program Interface*
ARP	*Address Resolution Protocol*
CSS	*Cascading Style Sheets*
HTTPS	*Hyper Text Transfer Protocol Secure*
IETF	*Internet Engineering Task Force*
IOS	*Internetwork Operating System*
IPV4	*Internet Protocol version 4*
IPV6	*Internet Protocol version 6*
NETCONF	*Network Configuration Protocol*
NETMIKO	*Python library for network management*
NTC	*Network to Code*
OPENFLOW	*Protocol for network management*
PUPPET	*Network automation tool*
PySNMP	*SNMP library for Python*
RFC	*Request for Comments*
RPC	*Remote Procedure Call*
SDN	*Software Defined Networking*
SSH	*Secure Shell*
SNMP	*Simple Network Management Protocol*
SQL	*Structured Query Language*
VLAN	*Virtual Local Area Network*
XML	*Extensible Markup Language*
YAML	*Ain't Markup Language*

SUMÁRIO

INTRODUÇÃO

Atualmente as redes de computadores tem se tornado cada vez mais complexas pois sua infraestrutura é cada vez mais heterogênea, sendo formada por diferentes equipamentos como roteadores e *switches* que possuem hardware e software desenvolvidos por diferentes empresas. Além disso, as redes têm se tornado cada vez mais dinâmicas para atender as necessidades atuais de disponibilidade e segurança, já que existe a tendência de que o número de dispositivos conectados às redes corporativas continue aumentando. Esse dinamismo e complexidade traz desafios aos administradores de rede já que é necessário que as redes se mantenham em constante evolução (KIM; FEAMSTER, 2013).

Os administradores de uma rede são responsáveis por definir e aplicar políticas para uma série de eventos que determinarão o comportamento da rede. Essas políticas precisam ser aplicadas em uma variedade de equipamentos e nem sempre estarão de acordo com as constantes mudanças de necessidades das redes, que dependerão de alterações ou adaptações posteriores. Porém, normalmente tanto a implementação quanto à manutenção dessas políticas é feita em baixo nível, ou seja, os administradores de rede precisam acessar os diferentes dispositivos via linha de comando, tornando a tarefa dispendiosa (KIM; FEAMSTER, 2013).

Esses diferentes dispositivos que formam as redes corporativas atuais possuem softwares proprietários com código fonte fechado. Como por exemplo, um dispositivo *CISCO* com sistema operacional *IOS* e um dispositivo *Juniper* com *JunOS*. Apesar de se comunicarem entre si através de protocolos padrão e proverem interoperabilidade para a rede, sua administração acaba por ser diferente. Isso dificulta que softwares de gerenciamento possam administrar os diferentes dispositivos e também dificulta a inovação, pois os dispositivos dependem de diferentes mecanismos e protocolos para serem administrados. Ou seja, a grande gama de dispositivos acaba por ser uma barreira administrativa enquanto as redes

se tornam cada vez mais complexas e necessitam mais funcionalidades (FEAMSTER; REXFORD; ZEGURA, 2013).

Uma das soluções propostas para resolver este problema são as redes definidas por software, mais conhecidas como *Software Defined Networking* - SDN, em inglês. A ideia principal é tornar as redes programáveis e com um controle lógico central, ou seja, os dispositivos físicos seriam apenas distribuidores de pacotes enquanto o controle seria centralizado via software (FEAMSTER; REXFORD; ZEGURA, 2013). Com isso surgiu o *OpenFlow*, um protocolo específico para as SDN que começou a ser estudado e incentivado pela indústria, já que através da associação de inúmeras empresas foi criada a O*pen Networking Foundation* que prevê a colaboração mútua dessas empresas para a adoção do protocolo (OPEN NETWORKING FOUNDATION, 2015). Porém nem todos os dispositivos atuais estão preparados para utilizar o *OpenFlow*, além disso, como os dispositivos de rede são proprietários, o suporte a uma SDN propriamente dita pode depender da contratação de licenças extras ou a adoção de softwares específicos.

Com isso uma outra solução surgiu, baseada em um novo conceito cultural que cresce cada vez mais na indústria mundial. Este conceito se chama *DevOps*, em inglês, e se baseia na ideia de unir desenvolvimento (*Dev*) e operações (*Ops*). Um *DevOps* normalmente é conhecido por utilizar softwares de automação de infraestrutura, que pode ir de serviços até redes (GREENE, 2015). Ou seja, o objetivo principal deste movimento é acabar com o conflito entre desenvolvimento e operações, fazendo com que a criação de serviços seja mais ágil (WILSON, 2015). Devido ao nascimento do movimento *DevOps*, *frameworks* de automação como *Puppet*, *Chef* e *Ansible* surgiram para ajudar na automação da infraestrutura (LERNER, 2014). Com o surgimento desse movimento, uma das maiores empresas de dispositivos de rede, a *CISCO*, criou um programa chamado *DevNet* que promove o desenvolvimento de softwares para o gerenciamento de redes (HIGGINBOTHAM, 2014; BABCOCK, 2014). A *Juniper* também fez o mesmo facilitando que diversas ferramentas se integrem a seus dispositivos e os tornem programáveis (BURKE, 2013).

Esses softwares satisfazem as necessidades básicas de automação das redes atuais. Necessidades essas que iniciam com o provisionamento da

infraestrutura, que envolve a configuração inicial para tornar a rede funcional. A segunda necessidade básica é o seu gerenciamento constante, e é aqui que a automação pode fazer a maior diferença já que esta tarefa é que a consome mais tempo dos administradores de rede. Além disso, em alguns casos, existe a necessidade de orquestração, que é basicamente quando o gerenciamento e o provisionamento de diversos dispositivos da rede é feito de forma conjunta (JUNIPER NETWORKS, 2015).

Devido à existência desse problema este trabalho apresenta um estudo de conceitos e softwares para resolver o problema de automação e provisionamento de equipamentos de rede. Este estudo foi realizado utilizando os dispositivos de rede atualmente disponíveis nos laboratórios do curso de Tecnologia em Redes de Computadores da Universidade Federal de Santa Maria – UFSM. Durante o trabalho descobriu-se que apesar da variedade de ferramentas existentes, poucas poderiam ser utilizadas no ambiente de testes. Além disso, apesar das ferramentas proverem uma eficaz solução para a automação dos dispositivos, estas também possuíam a necessidade de se utilizar interface de linha de comando já que a realização de tarefas é programável. Visando facilitar a administração da rede este trabalho também propõe a criação de uma interface gráfica simplificada para o gerenciamento dos equipamentos e utilização em conjunto com as ferramentas capazes de serem testadas com os equipamentos disponíveis no laboratório.

1.1 Automação

Para a *IBM*, automação de infraestrutura é o processo de codificar ambientes de TI, ou, infraestrutura como código. Ou seja, a automação é a forma mais prática de se evitar tarefas manuais e repetitivas (DUVALL, 2012). *Terry Slatterry*, da consultoria de redes *Netcraftsmen*, cita que o principal fator para se adotar automação de redes é evitar erro humano. Segundo ele, configurar centenas de roteadores e *switches* não deveria ser uma tarefa manual (SLATTERY, 2010). A *PuppetLabs* também elegeu as suas razões para a automatizar a configuração de dispositivos e entre elas estão evitar quedas nos serviços, melhorar a escalabilidade e ter rápida resposta a vulnerabilidades (PUPPET LABS, 2014 B). Com isso,

percebe-se que a automação além de uma necessidade está sendo amplamente defendida pela indústria de tecnologia. Outros benefícios como diminuir as tarefas tediosas e repetitivas permitindo que os administradores possam realizar tarefas mais importantes também são citados (CONDE, 2015). Ou seja, a automação da infraestrutura é um passo importante que gera dinamismo, flexibilidade e qualidade na entrega de serviços de TI.

1.2 Pesquisas de mercado

Em 2011 a *UBM TechWeb* realizou uma pesquisa patrocinada pela *CA Technologies* onde 460 participantes entre profissionais de TI e responsáveis por negócios em TI de empresas com mais de mil funcionários nos Estados Unidos e Canadá responderam perguntas sobre computação na nuvem, virtualização e automação. A pesquisa foi organizada de forma a considerar itens como muito importantes ou não importantes, em uma escala de 1 a 5, isso permitia aos participantes escolher mais de uma situação como muito importante. Quando perguntados sobre o que motiva a adoção de automação para serviços de TI, 33% consideraram como principal fator diminuir complexidades envolvidas na entrega de aplicações e serviços, 39% consideraram como fator mais importante o aumento da produtividade dos funcionários, permitir a entrega de aplicações mais flexíveis foi a escolha com 30%, 29% escolheu o aumento da entrega de aplicações alinhada com as necessidades do negócio, e 31% consideraram como mais importante aliviar o dia a dia das tarefas da equipe de TI permitindo que os profissionais se concentrem em outros projetos (BACHELDOR, 2011).

1.3 Justificativa e Objetivos

Com a complexidade e necessidade de dinamismo nas redes de computadores se tornando cada vez mais importantes, o estudo sobre os conceitos e softwares para automatizar as tarefas de administração se torna essencial para um profissional administrador de redes. Além disso, um estudo onde a aplicação de

conceitos acadêmicos e novas tendências da indústria podem ser aplicados a equipamentos amplamente utilizados pelo mercado auxilia no entendimento de como são estruturadas as redes de computadores atuais.

Como objetivo geral este projeto visa testar e implementar ferramentas existentes para automação e provisionamento de dispositivos de rede, e entender quais as vantagens e limitações existentes neste tipo de tecnologia. Além disso, serão desenvolvidas melhorias de usabilidade para a ferramenta que melhor se adaptar ao cenário de testes, pois sabe-se que estas não possuem interface de usuário, ou se possuem são necessárias licenças. Esta proposta de melhoria inclui a criação de uma interface de usuário para que seja possível a administradores de rede manipular com mais facilidade a ferramenta, tornando o processo de automação dos dispositivos ainda mais eficiente. A interface de usuário será construída com base nos padrões Web atuais e capaz de se adaptar a qualquer padrão de telas, incluindo dispositivos móveis, pois assim o administrador poderá recorrer até mesmo ao seu *smartphone* para mudanças rápidas de configuração.

O cenário de testes inclui *switchs* de rede *CISCO* modelo *Catalyst* 2960, existentes nos laboratórios do curso de Tecnologia em Redes de Computadores da UFSM. Foram testadas as possibilidades de automatizar a criação de interfaces de redes e suas respectivas *VLANs*. Após os testes com as ferramentas, uma foi escolhida para receber a interface de usuário proposta neste trabalho. Será também apresentado a integração da interface de usuário com a ferramenta escolhida, bem como sua implementação e testes. Como última etapa são discutidos os resultados, as vantagens e todas as limitações encontradas durante o projeto.

1.4 Estruturação do Trabalho

Este trabalho apresentará em um primeiro momento (Capítulo 2) estudos relacionados e tecnologias. Em um segundo momento (Capítulo 3) será apresentado como os estudos foram realizados, bem como o desenvolvimento e integração das soluções escolhidas. Posteriormente (Capítulo 4) serão apresentados alguns testes e resultados. E ao final do trabalho serão apresentadas as considerações finais e motivações para trabalhos futuros.

2 TECNOLOGIAS DISPONÍVEIS

2.1 Protocolos

Nesta seção do trabalho serão apresentados alguns protocolos que podem ser utilizados para o gerenciamento de redes de computadores, entre eles o OpenFlow, o Netconf e o SNMP.

2.1.1 OpenFlow

OpenFlow é um protocolo aberto para controle de dispositivos de rede proposto em 2008 por pesquisadores de algumas universidades americanas como *Stanford* e *Princeton*. A criação dessa proposta, segundo os pesquisadores, foi motivada pela dificuldade de inovar e de se realizar experimentos com equipamentos de rede, devido a grande base instalada de diferentes equipamentos e protocolos. Esse padrão proposto visa controlar as tomadas de decisão mais complexas de um *switch* através de um servidor central, deixando com que o *switch* processe apenas o encaminhamento dos pacotes. Ou seja, todas as configurações e tomadas de decisão adicionais seriam controladas pelo servidor central. Tornando *switches* e roteadores programáveis seria possível realizar experimentos utilizando as redes das universidades, sem necessitar que as empresas abrissem o código fonte de seus equipamentos proprietários. Lentamente várias empresas estão adicionando suporte ao *OpenFlow* em seus dispositivos. A Figura 1 demonstra como funcionaria uma rede administrada pelo *OpenFlow*, com um servidor central controlando um *switch* via software utilizando um canal seguro para alterar a tabela de fluxo (ANDERSON et al, 2008; OPENFLOW, 2015).

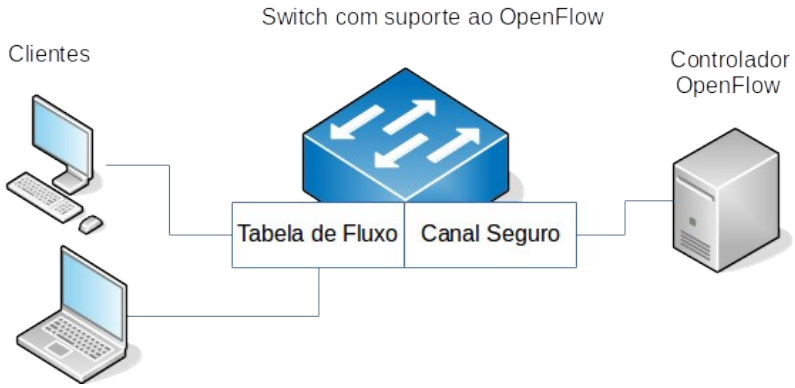

Figura 1 - Exemplo de Rede controlada pelo *OpenFlow*

2.1.2 Netconf

O *Network Configuration Protocol* (*Netconf*) é um protocolo criado pela *Internet Engineering Task Force* (*IETF*) com o objetivo de manipular configurações de dispositivos de rede sem a necessidade de se utilizar as interfaces de linha de comando proprietárias dos fabricantes. Foi criado originalmente pela RFC 4741 em 2006 e já recebeu diversas atualizações recentes, sendo a útima versão de 2011 com a RFC 6242 que resolveu alguns problemas de segurança. O protocolo transmite seus dados utilizando *SSHv2* na camada de transmissão, enquanto isso na camada de aplicação utiliza uma coleção de *Remote Procedure Call (RPC)* codificadas em arquivos *XML* (CISCO, 2013; TAIL-F, 2015). Assim como os outros protocolos, o *Netconf* necessita ser suportado pelo hardware das empresas que fornecem dispositivos de rede. A CISCO adquiriu a *Tail-F Systems* e seu sistema de gerenciamento utilizando *Netconf*, porém apenas os mais novos dispositivos possuem suporte ao protocolo (TAIL-F, 2015).

2.1.3 SNMP

O *Simple Network Management Protocol* é um protocolo inicialmente criado pela *Internet Engineering Task Force (IETF)* no final dos 80 com o objetivo de monitorar e administrar redes de computadores. Hoje o protocolo possuí três versões funcionando em regime de coexistência, são elas as versões 1, 2c e 3. Comparado com os outros protocolos e plataformas existentes o *SNMP* atualmente deixou de ser um protocolo simples, porém por ser antigo é amplamente suportado por qualquer fabricante de *hardware*, mesmo que esse suporte contenha algumas limitações. Portanto, ele ainda é considerado o protocolo padrão para a administração de redes (IBM, 2012; RUPP, 2014; LEVI, et. al, 2002, CASE, et. al, 1990).

A arquitetura do *SNMP* é baseada no que é chamado de *Management Information Base (MIB)* que descreve todas as informações gerenciáveis do dispositivo. Essas informações são catalogadas por um *Object ID (OID)* para cada elemento gerenciável. Cada um desses *OIDs* é único e segue padrões que devem ser adotado por toda a indústria. Esse padrão é composto por códigos registrados junto à *IETF*, que inclui dados de fabricante, a função do objeto, qual o tipo de equipamento e entre outros dados. O *SNMP* basicamente funciona através do envio de comandos pelo administrador e de resposta pelo dispositivo que está sendo gerenciado. Entre os comandos básicos estão:

- *GET:* requisição de informações de um *OID* do dispositivo gerenciado.
- *SET:* utilizado para definir valores para um *OID* no dispositivo.
- *TRAP:* comando que é inicializado pelo dispositivo, como um sinal de que ocorreu um determinado evento.
- *WALK*: permite listas todos os *OIDs* existentes na *MIB* do dispositivo.

As versões 1 e 2c do *SNMP* são praticamente idênticas em relação a sua configuração e sintaxe de comandos, pois ambas utilizam o conceito de comunidade para autenticação nos dispositivos. Enquanto isso a versão 3 inclui autenticação baseada em usuários e criptografia dos dados utilizando uma frase pré-definida (IBM, 2012; RUPP, 2014; LEVI, et. al, 2002, CASE, et. al, 1990).

2.2 Bibliotecas

Após a apresentação de alguns protocolos para gerenciamento, nesta parte será apresentada uma biblioteca programável também capaz de auxiliar na administração de redes de computadores.

2.2.1 Netmiko

O *Netmiko* é uma biblioteca lançada em junho de 2015, escrita em *Python* pelo desenvolvedor *Kirk Byers* e disponível no *GitHub.* Possuí como objetivo simplificar a biblioteca de *SSHv2* da linguagem *Python*, chamada de *Paramiko*. A biblioteca *Paramiko* é capaz de utilizar *SSH* para conexão com múltiplos dispositivos de rede de diferentes marcas e sistemas operacionais. Por exemplo, a *Netmiko* suporta dispositivos *CISCO* rodando os sistemas IOS, IOS-XE, IOS-XR, ASA e NX-OS. Bem como dispositivos *Arista* rodando o sistema vEOS, *Juniper* rodando *JunOS* e *HP ProCurve*. Além disso, suporte a outras marcas de dispositivos estão sendo incluídas, como *Brocade*, *Avaya* e *Huawei*, mas os testes com esses dispositivos no momento ainda são limitados (KIRK, 2015).

A forma como é feita essa simplificação das conexões pela ferramenta *Netmiko* acontece de várias formas. A primeira é estabelecer com sucesso uma conexão *SSH* com os dispositivos de forma transparente utilizando comandos simples. Posteriormente, é simplificada a execução de comandos e coleta de dados desses dispositivos, tratando qualquer formato em que esses dados são disponibilizados para um formato possível de ser manipulado utilizando Python. Além disso, o *Netmiko* trata a necessidade de se inserir senhas ou comandos extras, por exemplo, para administrar um dispositivo CISCO IOS é necessário além do usuário e senha do SSH, também a senha do dispositivo em si e a utilização do comando *Enable,* que pode ser reduzida a apenas uma chamada de função da biblioteca, que para este caso seria a chamada *net_connect.enable().* Uma outra funcionalidade da biblioteca está na execução múltipla de comandos, seja esta para um mesmo dispositivo ou para dispositivos diferentes como mostra a Figura 2 (KIRK,

2015).

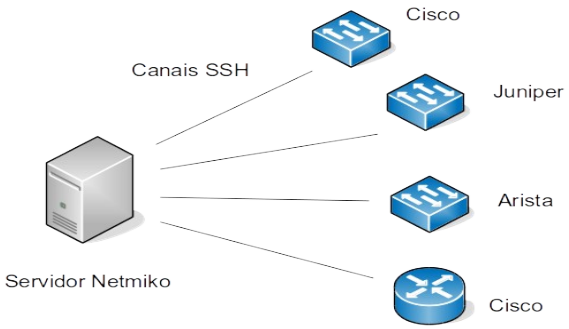

Figura 2 - Exemplo de funcionalidade da biblioteca *Netmiko*

Além disso, o *Netmiko* possuí um módulo chamado *NTC-ANSIBLE* que o adapta para funcionar em conjunto com o *Ansible*. Esse módulo foi desenvolvido pela empresa *Network to Code* e disponibilizado de forma *open source* no *GitHub* durante o segundo semestre de 2015. Apesar de ser um módulo novo, já apresenta uma boa quantidade de funcionalidades implementadas para a utilização em conjunto do *Ansible* e da biblioteca *Netmiko (*NETWORK TO CODE, 2015).

2.3 Aplicações

Esta parte do trabalho apresenta algumas aplicações que fazem uso dos protocolos ou biblioteca citados. Estas aplicações visam abstrair ainda as dificuldades encontradas no gerenciamento das redes de computadores através de um controle central ou automação de processos.

2.3.1 OpenDayLight

O *OpenDayLight* é um projeto *open source* mantido pela *Linux Foundation* que possuí como objetivo principal se tornar o padrão para redes definidas por software – *Software Defined Networks* (SDN). O projeto já está na sua terceira versão, é modular, multi-protocolo e funciona com equipamentos de diferentes marcas de hardware (OPEN DAY LIGHT, 2015). Grandes empresas como *IBM*,

Microsoft, RedHat, entre outras, patrocinam o projeto. Algumas, como a *CISCO* inclusive implementam a sua própria versão proprietária do *OpenDayLight*, modificada para funcionar apenas em seus dispositivos. Isso é possível pois o projeto foi pensado para suportar inúmeros protocolos, desde proprietários, ou *open source* como o *OpenFlow*. A Figura 3 mostra como é a estrutura do *OpenDayLight*, que inclui várias camadas, indo desde a interface do usuário até os protocolos que conectam dispositivos físicos ou virtuais (SULTAN, 2013).

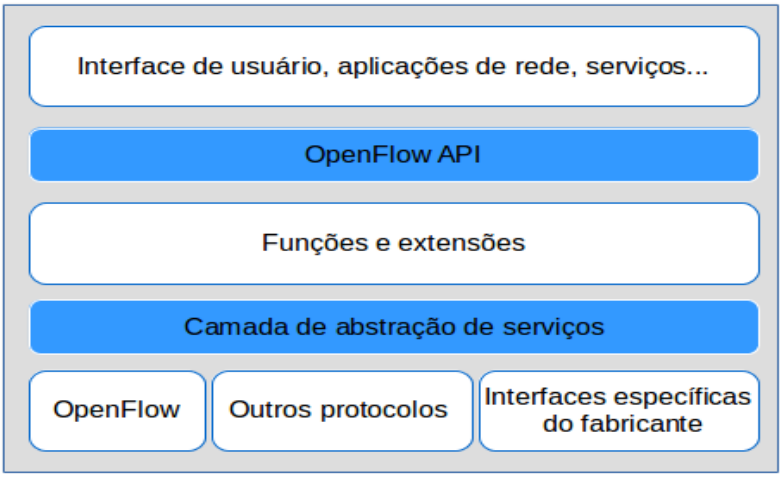

Figura 3 - Exemplo da estrutura de camadas do *OpenDayLight*

2.3.2 Ansible

O *Ansible* é uma plataforma *open source* para a criação de aplicações, módulos e scripts de automação, provisionamento, orquestração, entre outras funcionalidades, para sistemas e dispositivos que fazem parte de um ambiente de TI como servidores e dispositivos de rede. Desenvolvido em *Python*, foi lançado em 2012 pelo desenvolvedor independente *Michael DeHaan*, e hoje possui mais de mil colaboradores no *GitHub* se tornando o software de automação mais popular entre os desenvolvedores. Além desta popularidade, o site oficial do projeto informa que o Ansible já foi adotado por 20 das empresas na lista da *Fortune 100*, uma revista que lista as empresas com maiores rendimentos nos Estados Unidos e no mundo (ANSIBLE, 2015).

Recentemente a *Red Hat* adquiriu o *Ansible* por aproximadamente U$ 150 milhões de dólares. Segundo a empresa, o *Ansible* é simples de ser utilizado, é modular, e possui uma grande quantidade de desenvolvedores (LUNDEN, 2015, RED HAT, 2015). Porém, o principal interesse da *Red Hat* está em sua capacidade de automação de qualquer ambiente de TI, principalmente em relação à aplicações na nuvem. Além disso, o *Ansible* utiliza o conceito de *DevOps*, ou seja, visa permitir o desenvolvimento de aplicações para o controle da infraestrutura e entrega contínua de serviços alinhados às necessidades de negócio das empresas de uma forma mais prática e ágil (PERILLI, 2015).

A simplicidade do *Ansible* começa com o chamado *Playbook*, que consiste no *script* das tarefas a serem executadas. Esse *script* é escrito na linguagem *YAML,* um formato de serialização de dados que possuí como objetivo ser legível para humanos *(*EVANS, 2015*)*. A extensão utilizada pelo YAML é o '.yml'. Ou seja, o *Ansible* não exige que os scripts sejam escritos diretamente em *Python*, portanto não é necessário possuir conhecimentos avançados em programação para iniciar a utilizar o *Ansible* (ANSIBLE, 2015). O Quadro 1 demonstra um *Playbook* para configuração do servidor *web Apache.*

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum:
      name: httpd
      state: latest
  - name: write the apache config file
    template:
      src: /srv/httpd.j2
      dest: /etc/httpd.conf
    notify:
    - restart apache
```

```
- name: ensure apache is running
  service:
    name: httpd
    state: started
handlers:
  - name: restart apache
    service:
      name: httpd
      state: restarted
```

Quadro 1 - Exemplo de Playbook do Ansible

Percebe-se que utilizando o formato de dados *YAML* fica fácil de compreender as tarefas que serão realizadas. Além disso, o *Ansible* possuí uma outra facilidade, que é um cadastro prévio estático ou dinâmico do inventário da infraestrutura, permitindo assim que endereços, usuários e senhas de cada dispositivo fiquem salvos para utilização posterior, sem a necessidade de retrabalho (ANSIBLE, 2015).

A Figura 4 um demonstra um exemplo de como seria a arquitetura do *Ansible*, onde a aplicação é alimentada por dados de diversas fontes. E a partir disto, executa as tarefas nos alvos que podem ser qualquer dispositivo, desde de servidores a *switches* de rede, que fazem parte do escopo deste projeto. Essa execução das tarefas pelo *Ansible* é normalmente feita utilizando *SSHv2*, porém como o *Ansible* é modular outros protocolos como *SNMP* e *Netconf* também podem ser utilizados (ANSIBLE, 2015).

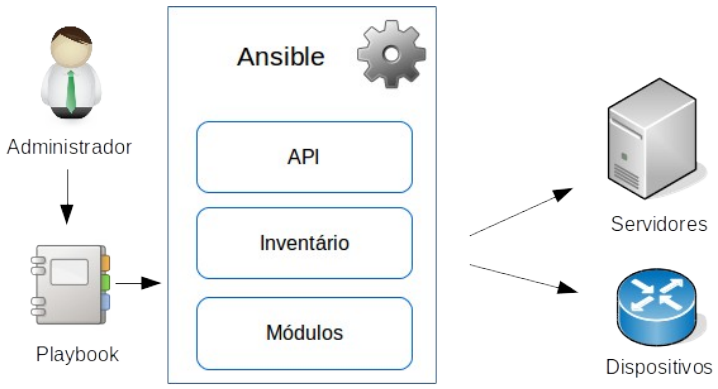

Figura 4 - Arquitetura do *Ansible*

2.3.3 Puppet

Outra plataforma para criação de scripts e módulos de automação de redes, o *Puppet* é mantido pela empresa *Puppet Labs* e considerado o líder em automação de ambientes de TI. A empresa emprega mais de 300 funcionários e conta com mais de U$ 86 milhões de dólares de investidores. Em 2013, possuía mais de 80 mil clientes. Escrito em *Ruby*, o *Puppet* possuí uma versão *open source* totalmente gratuita e uma versão paga e que acrescenta funcionalidades extras e proprietárias, não disponíveis em sua versão *open source* (PUPPET LABS, 2015).

Diferente do *Ansible*, a arquitetura do *Puppet* é baseada em agentes e um servidor principal. Os agentes são instâncias do *Puppet* instaladas em cada nó que precisa ser controlado, como por exemplo, servidores ou dispositivos de rede, conectados a um servidor principal, encarregado de controlar todos estes nós. Essa comunicação entre o servidor principal e os agentes acontece utilizando o protocolo HTTPS com identificação do cliente, ou seja, tanto o servidor quanto os nós precisam possuir um certificado válido, para isto, o *Puppet* inclui no servidor principal uma autoridade certificadora para controlar todos esses certificados, onde os agentes podem requisitar certificados ao servidor utilizando uma *API HTTP*. Devido a esta arquitetura e a necessidade de se utilizar nós, o *Puppet* necessita ser suportado pelo *hardware* a ser administrado. Como exemplo, dispositivos CISCO utilizando o sistema operacional NX-OS são suportados oficialmente, enquanto os dispositivos utilizando o sistema IOS são suportados apenas pela comunidade utilizando SSH como método de comunicação, porém sem atualização desde 2014. O SSH foi incluído como alternativa nas últimas versões do *Puppet* e funciona de forma similar ao empregado no *Ansible*, sem a utilização de agentes. Devido a essa atualização, os scripts e módulos precisam ser readaptados, porém muitos ainda não possuem o suporte adequado (PUPPET LABS, 2015).

2.3.4 Chef

O *Chef* é um *framework* de integração escrito na linguagem *Ruby* que segue a mesma filosofia de *DevOps* e concorre com as outras ferramentas possuindo uma versão *open source* e outra versão proprietária. Além das possibilidades de automação de todo ambiente de TI, o *Chef* ainda possuí funcionalidades diferentes das outras ferramentas, como por exemplo, a análise de todo o ambiente gerenciado e criação de relatórios de erros, bem como estatísticas de funcionamento de cada dispositivo. O *Chef* possuí uma arquitetura muito parecida com a do *Puppet*, onde é necessário a utilização de um tipo de agente nos nós que serão controlados. Devido a esta característica, em relação a dispositivos de rede CISCO o *Chef* possuí suporte apenas aos sistemas CISCO IOS-XR e NX-OS (CHEF, 2015).

2.3.5 SaltStack

Principal concorrente do *Ansible* como ferramenta de automação escrita em *Python*, o *SaltStack* possuí duas formas de controlar os dispositivos gerenciados por ele. A primeira, funciona como no *Puppet*, possuindo um servidor principal chamado *Master* e agentes que são instalados nos dispositivos gerenciáveis, que são chamados de *Minions*. Com essa forma de gerenciamento a plataforma também estaria limitada apenas a dispositivos que possuem suporte a determinados protocolos e tecnologias. Para contornar essa limitação, o *SaltStack* possuí uma segunda forma de gerenciamento, que funciona da mesma forma que o Ansible. Esse método é chamado *Proxy Minion*, onde o dispositivo pode ser gerenciado utilizando protocolos como o *SSH*. Devido a isso o SaltStack se torna um *framework* bastante flexível e customizável pois não fica atrelado a nenhuma tecnologia (SALTSTACK, 2015).

3 ESTUDO E DESENVOLVIMENTO

Este trabalho foi iniciado por um estudo da documentação das ferramentas e dos manuais dos dispositivos *CISCO Catalyst* 2960 disponíveis para testes a fim de entender se era possível administrar estes equipamentos. Se o resultado da pesquisa fosse positivo, seriam realizados testes para verificar o funcionamento da ferramenta em questão. Após o estudo e teste de algumas das ferramentas descobriu-se que as únicas opções viáveis para a automação do cenário disponível seria utilizando o *Ansible* em conjunto com o protocolo *SNMP* ou a biblioteca *Netmiko* em conjunto com o *SSH*. Como a biblioteca Netmiko é escrita em *Python* assim como o *Ansible*, ambos podem funcionar em conjunto, com scripts YAML do *Ansible* sendo executados para chamar a biblioteca *Netmiko* e executar comandos nos dispositivos remotos usando *SSH*, utilizando o módulo *NTC-ANSIBLE*. Porém, tanto a *Netmiko* quanto o *Ansible* precisam ser utilizados em uma interface de linha de comando, pois as suas funcionalidades visam total customização das tarefas a serem realizadas e foram criados com o intuito de serem básicos e modulares. Com isso, escolheu-se criar uma interface web capaz de realizar atividades básicas nos dispositivos utilizando o *Netmiko* ou exportando arquivos *YAML* para serem utilizados com o *Ansible*. Essa interface criada é responsiva e permite a utilização tanto em computadores como em dispositivos móveis como *smartphones*, facilitando a realização de tarefas triviais por parte do administrador da rede.

3.1 Cenário e Decisões Tecnológicas

Durante os testes com os equipamentos *CISCO Catalyst* 2960 descobriu-se que estes equipamentos podem ser administrados utilizando *SNMP* ou *SSH*. Porém, protocolos mais avançados como *OpenFlow* e *Netconf* não podem ser utilizados pois o dispositivo não os suporta. A fabricante *CISCO* incluiu suporte a estes protocolos apenas em alguns modelos mais recentes, não atualizando os anteriores para suportá-los. O *OpenDayLight* também não pode ser utilizado pelos mesmos motivos, e além disso existe uma versão proprietária da *CISCO* que depende de aquisição de

licença. A ferramenta de automação *Puppet* também apresenta limitações em relação ao cenário de testes, sendo que esta necessita utilizar a plataforma de desenvolvimento proprietária da *CISCO*, *onePK*, para a instalação da sua aplicação nos dispositivos, porém está plataforma necessita de uma licença adicional de desenvolvedor para a plataforma CISCO. Um módulo para o *Puppet* que permite a administração de equipamentos Cisco rodando o sistema IOS utilizando *SSH* foi testado sem sucesso. Segundo o site do *Chef*, este só possuí suporte a equipamentos *CISCO* com os sistemas IOS-XR e NX-OS, e portanto nem mesmo foi testado. O *SaltStack* foi testado e teve resultado positivo, conseguindo se conectar aos dispositivos utilizando *SSH*, porém bibliotecas específicas para *CISCO* não foram encontradas, tornando a sua utilização uma tarefa mais dispendiosa, pois este teria de ser adaptado para tratar as funções necessárias. Devido a essas limitações encontrou-se como melhor solução a utilização do Netmiko e do Ansible com SSH ou o apenas o Ansible utilizando SNMP.

3.1.1 SSH vs SNMP

A decisão entre os protocolos, *SSH* e *SNMP*, a ser utilizado para a transmissão de comandos aos equipamentos é importante e diferencia o desenvolvimento e funcionamento do trabalho pois o método de operação para cada protocolo é diferente. Com o *SNMP* seria utilizado um módulo criado para o *Ansible* pelo desenvolvedor independente *Patrick Ogenstad* e que está disponível livremente no *GitHub*. Para realizar a comunicação através de *SNMP*, esse módulo utiliza a biblioteca *PySNMP* que suporta as versões v1, v2c e v3 do SNMP, e a biblioteca é capaz de utilizar tanto *IPV4* como *IPV6* (PYSNMP, 2015). Porém, este módulo possuí limitações em suas funções e não possuí muita flexibilidade. Isso ocorre devido às limitações dos *OID* gerenciáveis via *SNMP* nos dispositivos *CISCO*, que apesar de permitir algumas opções de gerenciamento, ainda está longe das opções e flexibilidade disponíveis através da sua interface de linha de comando. Além disso, segundo o desenvolvedor o código ainda está em fase inicial e possuí alguns problemas já conhecidos. Durante os testes com a ferramenta foram encontrados erros de execução e problemas que precisaram ser contornados fazendo o

downgrade para uma versão mais antiga do *PySNMP*. Ou seja, o módulo pode se tornar incompatível a qualquer momento e programar diretamente na biblioteca *PySNMP* se tornaria trabalhoso e complexo, pois a simplicidade ao usar o módulo se perderia já que o motivo principal para a utilização desse módulo seria abstrair a administração dos *OID* necessários ao *SNMP*.

Por outro lado, a biblioteca *Netmiko* é mais simples e ao mesmo tempo possuí mais funcionalidades. Como é baseada em uma biblioteca para *SSH* já estabelecida e popular para desenvolvedores *Python* chamada *Paramiko*, suas funções *SSH* são executadas sem nenhum problema e de forma confiável. As funcionalidades ao utilizar a *Netmiko* aumentam em relação à utilização do módulo *SNMP* pois é possível utilizar qualquer comando disponível nos *CISCO* em sua plenitude. Bem como utilizar linha de comando em diferentes dispositivos, tornando o projeto mais robusto. Além disso, a biblioteca também possuí o módulo que a permite funcionar em conjunto com o *Ansible*. Portanto, a melhor solução encontrada foi utilizar a biblioteca *Netmiko* e executar os comandos de administração via *SSH*.

3.2 Interface de Usuário

Parte deste trabalho visa aplicar as tecnologias estudadas criando uma simples interface gráfica de usuário capaz de realizar tarefas básicas como criar uma *VLAN*. A motivação para a criação da interface gráfica de usuário não possuí como intuito a substituição da linha de comando. Existem discussões entre administradores de rede sobre as vantagens e desvantagens de se utilizar uma interface gráfica de usuário ou uma interface de linha de comando. O fato é que ambas tem suas vantagens e podem coexistir em uma mesma aplicação. Enquanto a interface de linha de comando é mais robusta, com maiores opções e flexibilidade, nem sempre ela é ideal para tarefas simples, pois muitas vezes ela exige um conhecimento prévio de funcionamento, bem como a memorização de comandos. Já a interface gráfica geralmente possuí menos opções, pois é menos flexível, mas também é mais prática para tarefas simples e rápidas, principalmente quando se desconhece o sistema que se deseja gerenciar (GARLING, 2012; POTT, 2012). Por exemplo, criar uma *VLAN* em um dispositivo específico em alguns cliques pode ser

mais eficiente do que ir até a linha de comando. Devido a isto, este projeto está estruturado de forma a possibilitar salvar os dados dos dispositivos e realizar tarefas básicas através da interface gráfica, mas também possuí a possibilidade de exportar estas tarefas e dados para um arquivo *YAML* que pode ser modificado e executado posteriormente, em caso de necessidade, via linha de comando utilizando o *Ansible*.

3.2.1 Tecnologias da Interface

Para o desenvolvimento da interface de usuário foi utilizada a linguagem Python na versão 2.7 e o sistema gerenciador de banco de dados MySQL versão 5.6. Como *framework web* para Python foi utilizado o Cherry Py. Que é *framework* minimalista e orientado a objetos de fácil utilização. Começou a ser desenvolvido em 2001, ou seja, já possuí um longo tempo de desenvolvimento e uma vasta quantidade de tutoriais e documentação na internet. Também é modular e permite a adição de *plugins* dos mais diversos tipos (CHERRYPY, 2015). Por ser minimalista permite que o projeto possua um código fonte pequeno e de fácil compreensão.

Além disso, foi utilizado o *framework Bootstrap* na versão 3.3.6. Criado pelo *Twitter* em 2010, é o mais popular *framework HTML, CSS e JavaScript* para a criação de projetos *web* responsivos. Com isso é possível criar uma interface adequada a qualquer tipo de tela, incluindo dispositivos móveis. O desenvolvimento é facilitado pois o *framework* possuí vários componentes que podem ser utilizados. Além disso, o *framework* segue os mais modernos padrões web tornando qualquer aplicação que o utilize compatível com a maioria dos navegadores do mercado (BOOTSTRAP, 2015).

3.2.2 Estrutura e Funcionamento

O projeto está estruturado de forma simples como mostra a Figura 5. Toda a lógica da aplicação, incluindo conexão com banco de dados e controle da interface, está contida em um único arquivo *Python*. O diretório *Static* contém os arquivos de estilo, JavaScript e fontes, onde estão localizados os ícones utilizados na aplicação.

O diretório *Templates* contém os arquivos html necessários à interface.

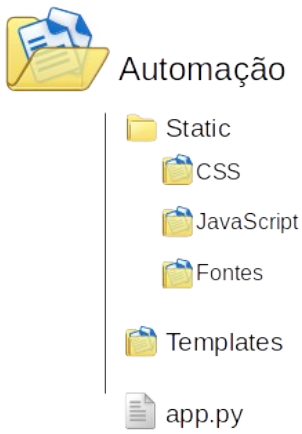

Figura 5 - Estrutura do Projeto

A base de dados está organizada com apenas duas tabelas. A tabela dispositivo e a tabela tarefas. As tarefas podem ser criadas com até três parâmetros e referenciarão ao menos um dispositivo através de uma chave estrangeira como mostra a Figura 6.

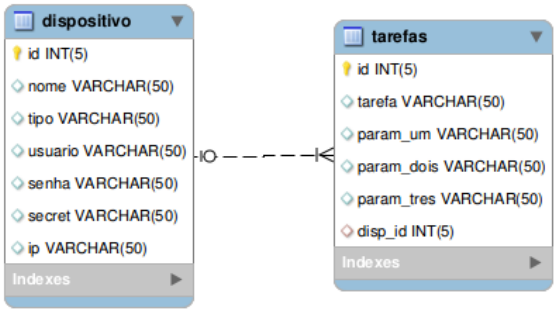

Figura 6 - Modelo Entidade Relacional da base de dados

O funcionamento da interface gráfica permitirá que o administrador execute tarefas diretamente, dessa forma a aplicação enviará comandos aos dispositivos utilizando a biblioteca Netmiko ou como alternativa poderá exportar um arquivo YAML. Com este arquivo YAML o administrador poderá executar as tarefas manualmente utilizando o *Ansible* como demonstra a Figura 7.

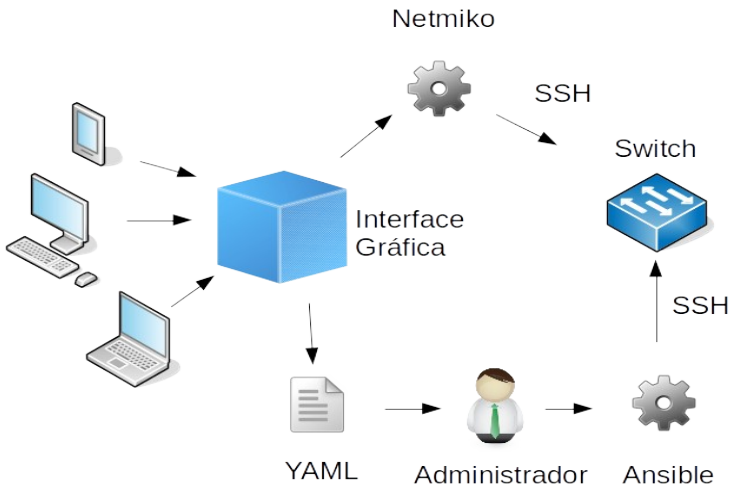

Figura 7 - Funcionamento da Aplicação

3.2.3 Usabilidade e Utilidade

O desenvolvimento de uma interface gráfica de usuário necessita o entendimento de quais e como serão as atividades a serem executadas por essa interface, bem como o entendimento sobre o comportamento humano para que a interface corresponda às capacidades e limitações das pessoas. Por isso, pensar em que tarefas a interface deverá ser usada para se tornar útil e como ela deverá ser manipulada pelo usuário é essencial. Ou seja, devemos pensar em utilidade e usabilidade (JACOB, 2000).

Enquanto a utilidade é simplesmente definida como algo que provê uma funcionalidade que o usuário realmente necessite, a usabilidade é um pouco mais complexa. Esta pode ser definida por cinco características diferentes. Primeira, é o quão fácil é para o usuário começar a realizar as tarefas que ele precisa. Segunda, é a eficiência na realização dessas tarefas após os usuários aprenderem a utilizar a

interface. Terceira, o quão fácil os usuários lembrarão como realizar as tarefas após um período sem utilizar a interface. Quarta, quantos erros os usuários cometem ao utilizar a interface. Quinta, qual a sensação de satisfação ao utilizar a interface. Ou seja, uma interface que contenha problemas em algumas dessas áreas poderá causar transtornos ao usuário. Portanto devemos pensar o quão útil é essa interface (NIELSEN, 2012).

Para evitar possíveis problemas com as características mencionadas, a interface gráfica de usuário do projeto visa ser simples ao realizar apenas o necessário. Além disso, interface foi pensada para funcionar tanto em computadores como dispositivos móveis, aumentando assim a sua utilidade, pois permite que o administrador da rede realize tarefas triviais utilizando apenas seu *smartphone*, sem a necessidade de estar em sua estação de trabalho.

Para manter a simplicidade a interface foi dividida em apenas quatro partes. Uma página inicial, uma página onde as tarefas serão adicionadas e ordenadas pelo usuário. Essa ordenação das tarefas é importante pois assim tarefas que dependem de outras podem ser definidas para serem executadas posteriormente. Uma página para adição dos dispositivos que serão gerenciados. Uma página para listar esses dispositivos, incluindo uma opção de remoção na listagem. E finalmente, uma página ao estilo *Wiki*, onde um pequeno manual para o funcionamento do sistema será inserido, ajudando o usuário em caso de dúvidas. A Figura 8 demonstra o resultado dessa interface quando utilizada em um computador e quando utilizada em um dispositivo móvel.

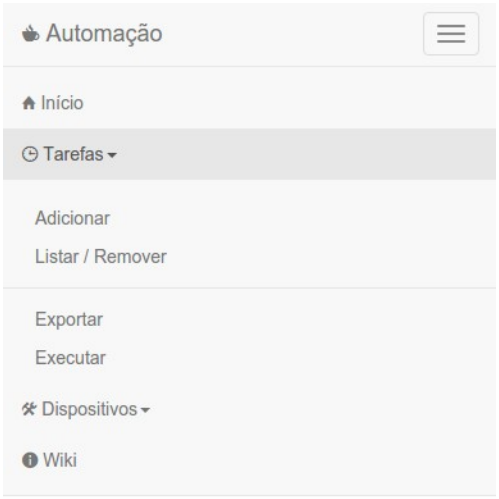

Figura 8 - Interface quando visualizada em dispositivos móveis

3.2.4 Funcionalidades

Entre as funcionalidades básicas da interface estão a adição de dispositivos assim como a listagem dos adicionados. Nesta listagem é possível excluir os dispositivos e também requisitar informações mais detalhadas. Quando o botão de informações adicionais é pressionado, uma nova tela é carregada contendo as informações das *interfaces* do dispositivo. A Figura 9 contém o formulário para a adição de dispositivos.

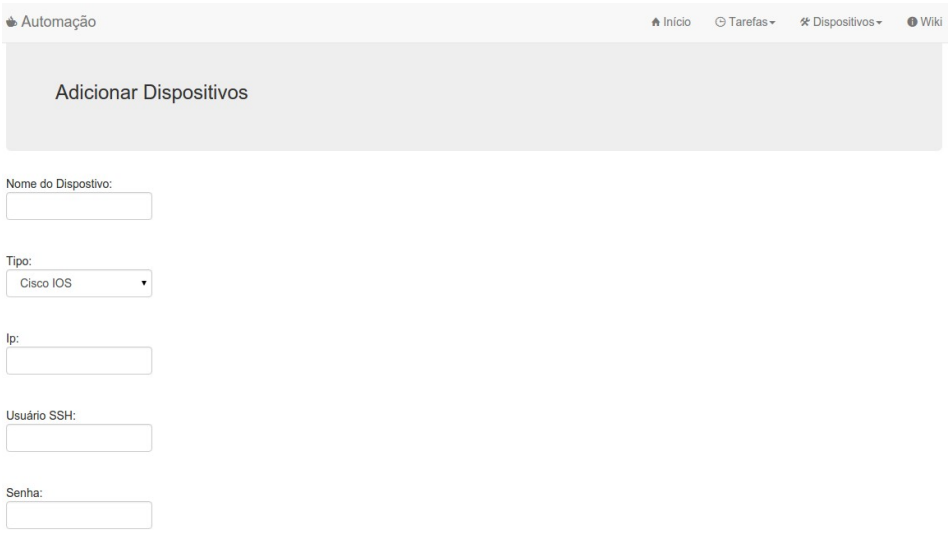

Figura 9 - Formulário para adição de dispositivos

A Figura 10 demonstra como é feita a listagem dos dispositivos cadastrados contendo as informações e os botões de ação.

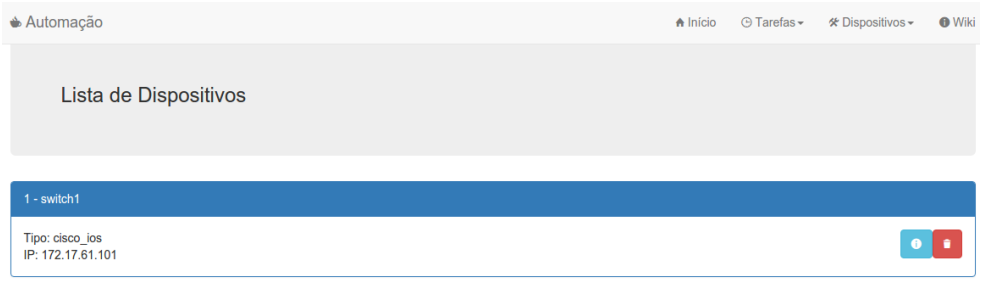

Figura 10 - Exemplo de listagem de dispositivos

A Figura 11 apresenta um exemplo de como funciona a visualização dos detalhes do dispositivo, onde é existe a interface, o IP de cada interface e o seu status.

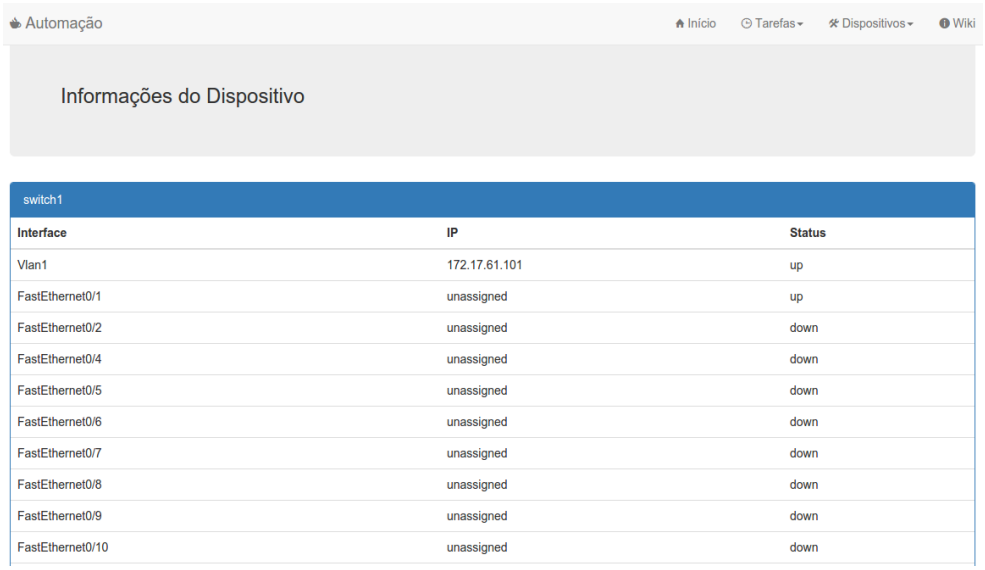

Figura 11 - Detalhes do Dispositivo

A interface também provê a adição de tarefas que podem ser executadas nos dispositivos. O protótipo desenvolvido, em sua primeira versão, inclui a possibilidade de se criar *VLANs* em dispositivos *CISCO*, outras funcionalidades ficarão reservadas para trabalhos futuros. A Figura 12 demonstra o formulário para a criação da tarefa de criação de *VLAN*.

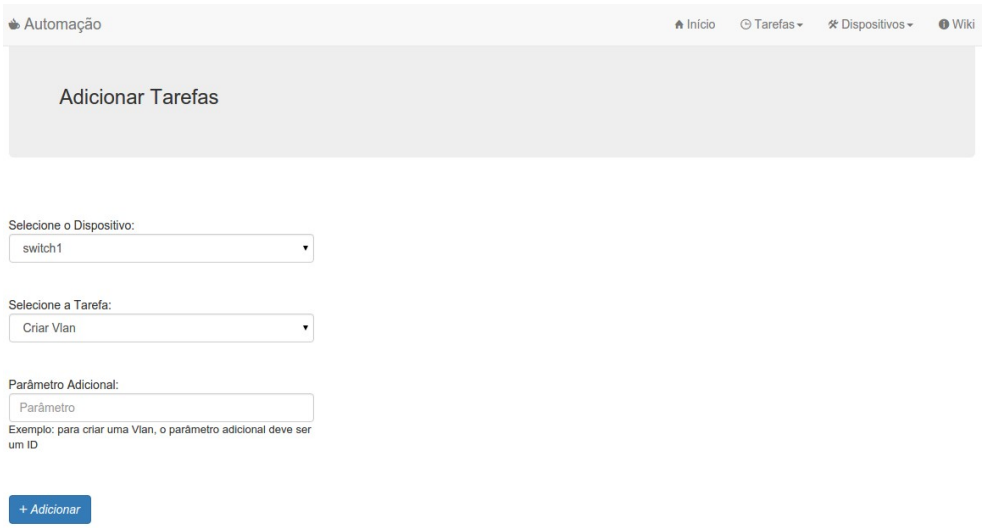

Figura 12 - Formulário para adição de tarefas

A Figura 13 apresenta a listagem das tarefas cadastradas e um botão de ação para excluir a tarefa.

Figura 13 - Listagem das tarefas existentes

Como mencionado anteriormente, a execução das tarefas criadas será feita pelo próprio sistema utilizando comandos disponíveis na biblioteca *Netmiko*, ou exportando os dados para *YAML*. Quando executado com o *Netmiko*, as tarefas a serem realizadas utilizarão as funções da biblioteca. A saída da execução dos comandos será mostrada na tela em uma caixa de texto que inclui a execução dos comandos e uma resumo das VLANs existentes no dispositivo para que o administrador possa verificar se tudo ocorreu corretamente como mostra a Figura 14.

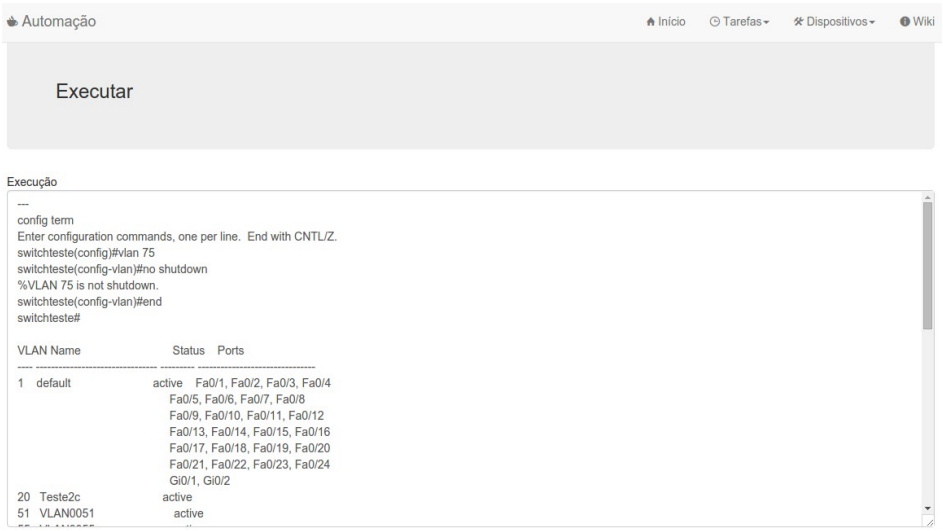

Figura 14 - Exemplo de execução de tarefas

O Quadro 2 apresenta um exemplo do código necessário para a execução desta tarefa utilizando a biblioteca *Netmiko*.

```
from netmiko import ConnectHandler

cisco_teste = {
        'device_type': 'cisco_ios',
        'ip':   '0.0.0.0',
        'username': 'teste',
        'password': 'senha',
        'port' : 22,
        'secret': 'teste',
        'verbose': True,
}

net_connect.enable()

config_commands = [ 'vlan 75' ,
                    'no shutdown']

output = net_connect.send_config_set(config_commands)

print output
```

Quadro 2 - Exemplo de script do *Netmiko*

Percebe-se que a execução de uma tarefa utilizando o *Netmiko* é simples. Porém esta mesma tarefa pode ser resumida no YAML exportado da Figura 15, que pode ser executado pelo *Ansible* e que utilizará a mesma biblioteca *Netmiko*.

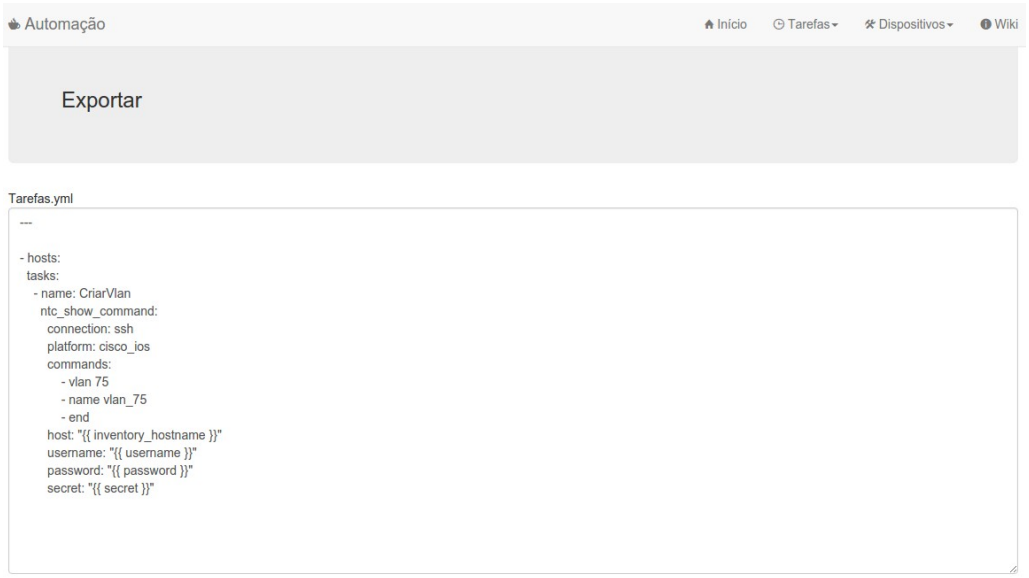

Figura 15 - Exemplo de exportação de tarefa para YAML

4 TESTES E RESULTADOS

Neste capítulo serão apresentados testes realizados com a ferramenta *Netmiko* e o módulo *NTC-Ansible*. Primeiramente serão apresentados testes de coleta de informações do dispositivo. Posteriormente os testes envolvendo a criação de VLAN no *switch*. Finalizando com testes utilizando a interface gráfica.

4.1 Teste de coleta de informações

O primeiro teste realizado foi a coleta de algumas informações do *Switch CISCO Catalyst* 2960 utilizando tanto o *Netmiko* diretamente, bem como um *script YAML* para o *Ansible* executar no Netmiko. No Quadro 3 pode-se visualizar o script utilizado para o teste direto com o *Netmiko*.

```
from netmiko import ConnectHandler

cisco_teste = {
        'device_type': 'cisco_ios',
        'ip':   '172.17.61.101',
        'username': 'teste',
        'password': 'senhateste',
        'port' : 22,
        'secret': 'teste',
        'verbose': True,
}

net_connect = ConnectHandler(**cisco_teste)

print "INFORMACOES DE INTERFACES"
output = net_connect.send_command('show ip int brief')
print output

print "TABELA ARP"
output = net_connect.send_command("show arp")
```

```
print output

print "INFORMACOES DE VLAN"
output = net_connect.send_command('show vlan')
print output

net_connect.disconnect()
```

Quadro 3 - Script de coleta de informações para o Netmiko

Percebe-se que a biblioteca Netmiko funciona de uma maneira bem simples abstraindo o processo de conexão e facilitando o envio de comandos. Na Figura 16 pode ser visto o retorno do estabelecimento da conexão e do comando '*show ip int brief*' com o resultado perfeitamente visível.

```
INFORMACOES DE INTERFACES
Interface              IP-Address      OK? Method Status                Protocol
Vlan1                  172.17.61.101   YES NVRAM  up                    up
FastEthernet0/1        unassigned      YES unset  up                    up
FastEthernet0/2        unassigned      YES unset  down                  down
FastEthernet0/3        unassigned      YES unset  administratively down down
FastEthernet0/4        unassigned      YES unset  down                  down
FastEthernet0/5        unassigned      YES unset  down                  down
FastEthernet0/6        unassigned      YES unset  down                  down
FastEthernet0/7        unassigned      YES unset  down                  down
FastEthernet0/8        unassigned      YES unset  down                  down
FastEthernet0/9        unassigned      YES unset  down                  down
FastEthernet0/10       unassigned      YES unset  down                  down
FastEthernet0/11       unassigned      YES unset  down                  down
FastEthernet0/12       unassigned      YES unset  down                  down
FastEthernet0/13       unassigned      YES unset  down                  down
FastEthernet0/14       unassigned      YES unset  down                  down
FastEthernet0/15       unassigned      YES unset  down                  down
FastEthernet0/16       unassigned      YES unset  down                  down
FastEthernet0/17       unassigned      YES unset  down                  down
FastEthernet0/18       unassigned      YES unset  down                  down
FastEthernet0/19       unassigned      YES unset  down                  down
FastEthernet0/20       unassigned      YES unset  down                  down
FastEthernet0/21       unassigned      YES unset  down                  down
FastEthernet0/22       unassigned      YES unset  down                  down
FastEthernet0/23       unassigned      YES unset  down                  down
FastEthernet0/24       unassigned      YES unset  down                  down
GigabitEthernet0/1     unassigned      YES unset  down                  down
GigabitEthernet0/2     unassigned      YES unset  down                  down
```

Figura 16 - Informações de interfaces com o Netmiko

Após isto, na Figura 17 tem-se o retorno do comando '*show arp*' que mostrando a tabela ARP do dispositivo, informando o *MAC Address* dos endereços assim como suas respectivas *VLANs*.

```
TABELA ARP
Protocol  Address          Age (min)  Hardware Addr   Type   Interface
Internet  172.17.12.219           5   f04d.a2e3.01ec  ARPA   Vlan1
Internet  172.17.23.56            4   02cf.a91a.746e  ARPA   Vlan1
Internet  172.17.8.40             6   0800.27ba.e2da  ARPA   Vlan1
Internet  172.17.8.29             0   0800.2780.c8c8  ARPA   Vlan1
Internet  172.17.61.101           -   0c85.2518.ea40  ARPA   Vlan1
```

Figura 17 - Informações da tabela ARP com o Netmiko

Por fim, tem-se o resultado do comando '*show vlan*' na Figura 18 onde existem informações mais detalhes sobre as *VLANs* existentes no *switch*.

```
INFORMACOES DE VLAN

VLAN Name                             Status    Ports
---- -------------------------------- --------- -------------------------------
1    default                          active    Fa0/1, Fa0/2, Fa0/3, Fa0/4
                                                Fa0/5, Fa0/6, Fa0/7, Fa0/8
                                                Fa0/9, Fa0/10, Fa0/11, Fa0/12
                                                Fa0/13, Fa0/14, Fa0/15, Fa0/16
                                                Fa0/17, Fa0/18, Fa0/19, Fa0/20
                                                Fa0/21, Fa0/22, Fa0/23, Fa0/24
                                                Gi0/1, Gi0/2
20   Teste2c                          active
51   VLAN0051                         active
55   VLAN0055                         active
75   VLAN0075                         active
1002 fddi-default                     act/unsup
1003 token-ring-default               act/unsup
1004 fddinet-default                  act/unsup
1005 trnet-default                    act/unsup

VLAN Type  SAID       MTU   Parent RingNo BridgeNo Stp  BrdgMode Trans1 Trans2
---- ----- ---------- ----- ------ ------ -------- ---- -------- ------ ------
1    enet  100001     1500  -      -      -        -    -        0      0
20   enet  100020     1500  -      -      -        -    -        0      0
51   enet  100051     1500  -      -      -        -    -        0      0
55   enet  100055     1500  -      -      -        -    -        0      0
75   enet  100075     1500  -      -      -        -    -        0      0
1002 fddi  101002     1500  -      -      -        -    -        0      0
1003 tr    101003     1500  -      -      -        -    -        0      0
1004 fdnet 101004     1500  -      -      -        ieee -        0      0
1005 trnet 101005     1500  -      -      -        ibm  -        0      0

Remote SPAN VLANs
------------------------------------------------------------------------------


Primary Secondary Type              Ports
------- --------- ----------------- ------------------------------------------
```

Figura 18 - Informações detalhadas de VLAN com o Netmiko

Posteriormente, foi realizado um teste utilizando o módulo *NTC-Ansible* que faz com que o *Ansible* que execute o *Netmiko* baseado em um *script YAML,* sem a necessidade de código em *Python*. O Quadro 4 contém o *script* executado, com as mesmas funções do *script* escrito diretamente em *Python*. Um detalhe importante para o *Ansible* é que as credenciais do dispositivo ficam salvas em um arquivo de

inventário separado chamado *hosts* como demonstrado no Quadro 5.

```
---

- name: TESTE DE COLETA DE INFORMAÇÕES
  hosts: cisco_ios
  connection: local
  gather_facts: False

  tasks:
    - name: COLETA INFORMAÇÕES INTERFACES
      ntc_show_command:
        connection: ssh
        platform: cisco_ios
        command: "show ip int brief"
        host: "{{ inventory_hostname }}"
        username: "{{ username }}"
        password: "{{ password }}"
      register: results

    - debug: var=results.response

    - name: COLETA ARP
      ntc_show_command:
        connection: ssh
        platform: cisco_ios
        command: "show ip arp"
        host: "{{ inventory_hostname }}"
        username: "{{ username }}"
        password: "{{ password }}"
      register: results

    - debug: var=results.response

- name: COLETA INFORMAÇÕES DE VLAN
      ntc_show_command:
        connection: ssh
        platform: cisco_ios
        command: "show vlan"
```

```
    host: "{{ inventory_hostname }}"
    username: "{{ username }}"
    password: "{{ password }}"
  register: results

- debug: var=results.response
```

Quadro 4: Exemplo de YML para coleta de informações

```
[switch1]
172.17.61.101 username=senhateste password=teste secret=teste
```

Quadro 5 - Exemplo de arquivo de hosts para o ansible

A Figura 19 mostra parte do retorno do comando '*show ip int brief*' executado pelo *Ansible*. Percebe-se que a formatação é diferente do que quando executado diretamente pelo *Netmiko*. Enquanto a Figura 20 mostra o resultado do comando 'show ip arp'.

```
PLAY [TESTE DE COLETA DE INFORMAÇÕES] ***************************************

TASK: [COLETA INFORMAÇÕES INTERFACES] ***************************************
ok: [172.17.61.101]

TASK: [debug var=results.response] ******************************************
ok: [172.17.61.101] => {
    "var": {
        "results.response": [
            {
                "intf": "Vlan1",
                "ipaddr": "172.17.61.101",
                "proto": "up",
                "status": "up"
            },
            {
                "intf": "FastEthernet0/1",
                "ipaddr": "unassigned",
                "proto": "up",
                "status": "up"
            },
            {
                "intf": "FastEthernet0/2",
                "ipaddr": "unassigned",
                "proto": "down",
                "status": "down"
            },
            {
                "intf": "FastEthernet0/3",
                "ipaddr": "unassigned",
                "proto": "down",
                "status": "administratively down"
            },
            {
                "intf": "FastEthernet0/4",
                "ipaddr": "unassigned",
                "proto": "down",
                "status": "down"
```

Figura 19: Coleta de informações de interface com o *Ansible*

```
PLAY [SHOW IP ARP] *************************************************************

TASK: [COLETA ARP] *************************************************************
ok: [172.17.61.101]

TASK: [debug var=results.response] *********************************************
ok: [172.17.61.101] => {
    "var": {
        "results.response": [
            {
                "address": "172.17.12.219",
                "age": "11",
                "interface": "Vlan1",
                "mac": "f04d.a2e3.01ec",
                "type": "ARPA"
            },
            {
                "address": "172.17.23.56",
                "age": "4",
                "interface": "Vlan1",
                "mac": "02cf.a91a.746e",
                "type": "ARPA"
            },
            {
                "address": "172.17.8.40",
                "age": "12",
                "interface": "Vlan1",
                "mac": "0800.27ba.e2da",
                "type": "ARPA"
            },
            {
                "address": "172.17.8.29",
                "age": "0",
                "interface": "Vlan1",
                "mac": "0800.2780.c8c8",
                "type": "ARPA"
            },
```

Figura 20 - Coleta de informações *ARP* utilizando o *Ansible*

Já a Figura 21 demonstra que o módulo *NTC-Ansible* ainda não está maduro o suficiente, pois não possui *templates* para o tratamento das informações do comando *'show vlan'*. Isso significa basicamente que não existem os filtros para este comando e portanto sem saber como tratar os dados o *Ansible* reportará um erro. Porém, é possível modificar o módulo e incluir os filtros necessários. Está modificação será inclusa em trabalhos futuros.

```
TASK: [COLETA INFORMAÇÕES DE VLAN] *********************************************
failed: [172.17.61.101] => {"error": "No template found for attributes: \"{'Platform': 'cisco_ios', 'Command': 'show vlan'}\"", "failed": true}
msg: parsing error

FATAL: all hosts have already failed -- aborting

PLAY RECAP *********************************************************************
        to retry, use: --limit @/home/ubuntu/teste1.retry

172.17.61.101              : ok=2    changed=0    unreachable=0    failed=1
```

Figura 21 - Erro ao coletar informações de VLAN com o Ansible

4.2 Teste de configuração

Os testes de configuração foram realizados com o objetivo de testar a criação de uma *VLAN*, novamente utilizando o *Netmiko* diretamente e através do *Ansible*. Desta vez para ambos os casos tem-se a execução de comandos em sequência, como pode ser visto nos códigos a seguir. No Quadro 6 tem-se o exemplo para a execução do *Netmiko* diretamente. Enquanto a Figura 22 demonstra o resultado do script.

```
from netmiko import ConnectHandler

cisco_teste = {
        'device_type': 'cisco_ios',
        'ip':   '172.17.61.101',
        'username': 'teste',
        'password': 'senhateste',
        'port' : 22,
        'secret': 'teste',
        'verbose': True,
}

net_connect = ConnectHandler(**cisco_teste)

net_connect.enable()

config_commands = [ 'vlan 55' ,
                'no shutdown',
                'end']
output = net_connect.send_config_set(config_commands)
print output

net_connect.disconnect()
```

Quadro 6 - Script de configuração de VLAN para Netmiko

```
SSH connection established to 172.17.61.101:22
Interactive SSH session established
config term
Enter configuration commands, one per line.  End with CNTL/Z.
switchteste(config)#vlan 55
switchteste(config-vlan)#no shutdown
%VLAN 55 is not shutdown.
switchteste(config-vlan)#end
switchteste#
```

Figura 22 - Configuração de VLAN com o Netmiko

Já no segundo bloco de código abaixo, tem-se o *script YML* para se criar uma VLAN com o *Ansible*. É importante lembrar que as informações de credenciais dos dispositivos ficam declaradas em um arquivo separado como variáveis.

```
---

- name: TESTE VLAN
  hosts: cisco_ios

  tasks:
    - name: CRIAR VLAN
      ntc_show_command:
        connection: ssh
        platform: cisco_ios
        commands:
          - vlan 80
          - no shutdown
          - end
        host: "{{ inventory_hostname }}"
        username: "{{ username }}"
        password: "{{ password }}"
        secret: "{{ secret }}"
```

Quadro 7: YML para teste de VLAN com Ansible

Ao executar este *script* se percebe pela Figura 23 que o *Ansible* reporta um erro de credenciais. Porém, este *bug* na execução do módulo *NTC-Ansible* aparentemente acontece pois o comando *'enable'* não é passado antes da tentativa de executar os comandos posteriores. Isso significa que o módulo não terá

permissão para executar comandos de configuração. Como tentativa de solucionar o problemas duas configurações diferentes foram utilizadas, porém sem sucesso.

```
PLAY [TESTE VLAN] ************************************************************

GATHERING FACTS **************************************************************
fatal: [cisco5] => SSH Error: Permission denied (keyboard-interactive,password).
    while connecting to 172.17.61.101:22
It is sometimes useful to re-run the command using -vvvv, which prints SSH debug output to help diagnose the issue.
fatal: [cisco1] => SSH Error: Permission denied (keyboard-interactive,password).
    while connecting to 172.17.61.101:22
It is sometimes useful to re-run the command using -vvvv, which prints SSH debug output to help diagnose the issue.

TASK: [CRIAR VLAN] ***********************************************************
FATAL: no hosts matched or all hosts have already failed -- aborting
```

Figura 23 - Falha ao criar *VLAN* utilizando o *Ansible*

4.3 Testes com a interface de usuário

Para testar a interface foi feita a tentativa de se criar as *VLANs* 90 e 95 no dispositivo utilizando diretamente a biblioteca *Netmiko*. Para isto foram incluídas tarefas com os parâmetros necessários como demonstra a Figura 24. Posteriormente ocorreu a execução das tarefas como demonstra a Figura 25. Após a execução das tarefas o comando *'show vlan'* também foi executado e teve seu resultado impresso na tela para que houvesse confirmação da criação das *VLANs* como mostra a Figura 26.

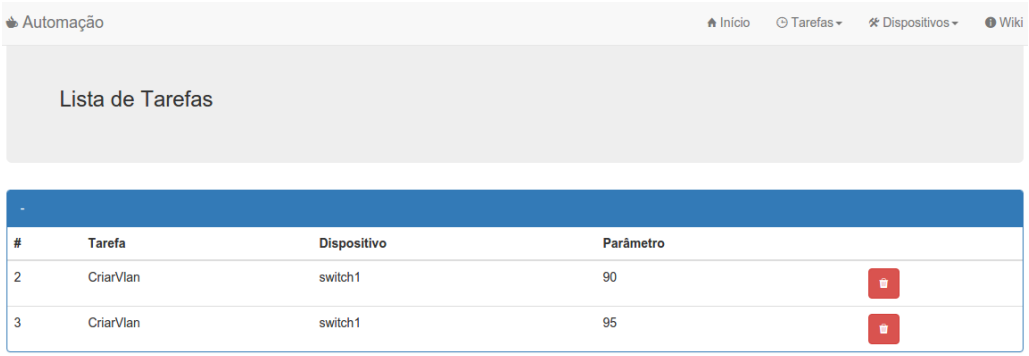

Figura 24 - Lista de tarefas para teste da interface

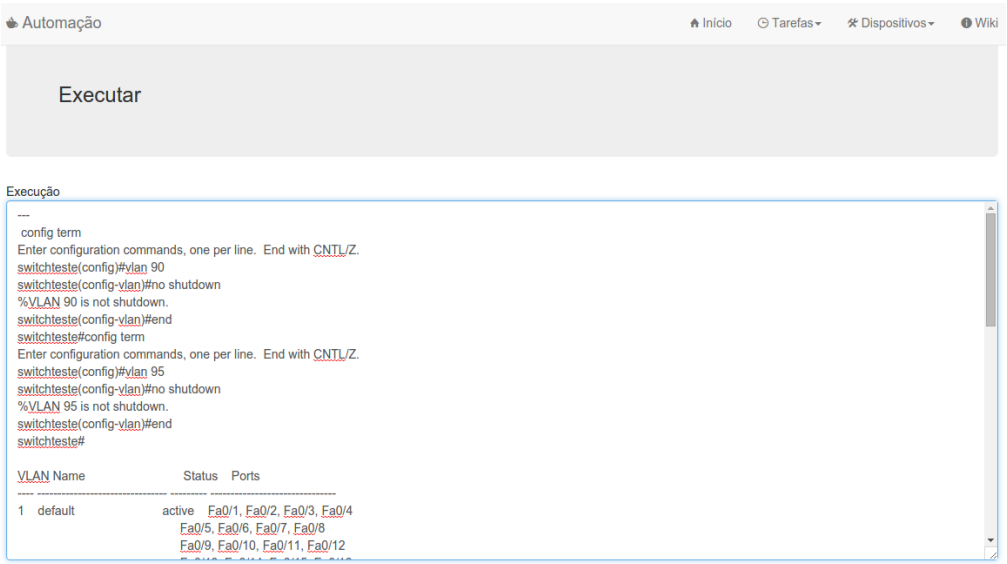

Figura 25 - Execução de testes com a interface

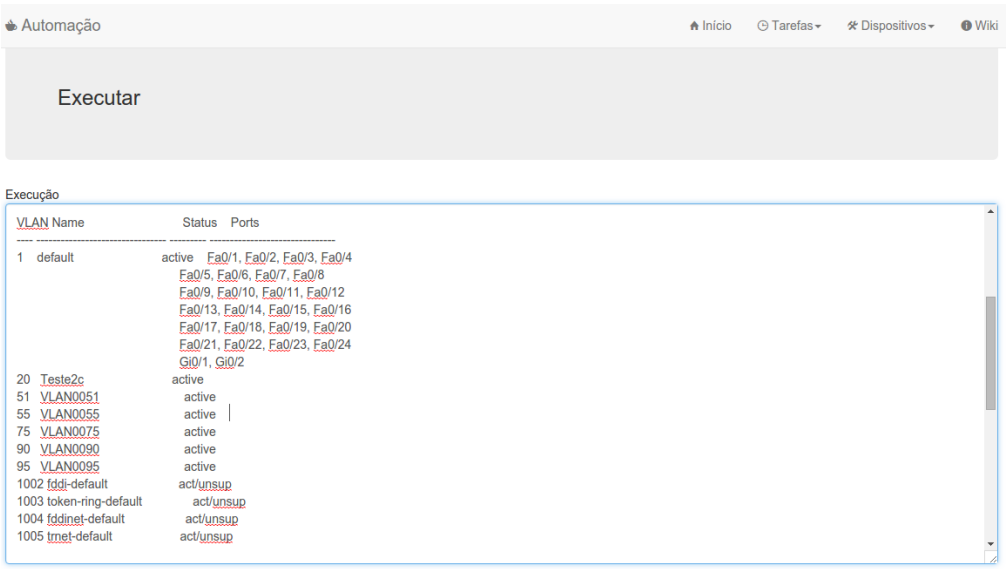

Figura 26 - Resultado dos testes com a interface

5. CONSIDERAÇÕES FINAIS

Com o objetivo de estudar ferramentas de automação e possibilidades de gerenciar as redes de computadores de forma mais prática, este trabalho estudou algumas das ferramentas disponíveis para a automação de dispositivos de rede e apresentou uma solução simples para a realização de tarefas triviais em *switches CISCO Catalyst* 2960.

Durante o período de estudo foi feita a leitura da documentação das ferramentas e equipamentos, bem como testes práticos onde estes eram possíveis. Concluiu-se quem nem todas as ferramentas disponíveis eram capazes de gerenciar este modelo de equipamento por causa de suas limitações tecnológicas. Inicialmente também estavam disponíveis *switches HP/3COM 1910*, porém estes demonstraram serem ainda mais limitados que os *CISCO* ao totalmente dependentes de sua interface proprietária de administração.

Do conjunto de ferramentas estudado e testado, foi escolhida uma combinação para ser testada e implementada. Essa combinação foi formada pela biblioteca *Netmiko* e pelo *software* de automação *Ansible*. Conjunto que se mostrou robusto em termos de funcionalidade e de simples implementação para o cenário de testes disponível.

Por fim foi proposta a criação de uma *interface* gráfica de usuário para auxiliar no gerenciamento e execução de tarefas básicas nos equipamentos. Interface que foi construída seguindo padrões *web* de forma a ser responsiva e funcionar tanto em computadores como dispositivos móveis.

Trabalhos futuros podem ser desenvolvidos melhorando a interface e adicionando funcionalidades, pois a biblioteca utilizada permite que dispositivos de diversas fabricantes sejam automatizados e o desenvolvimento deste trabalho não foi possível testar todas as variáveis possíveis.

REFERÊNCIAS

ANDERSON et. al. **OpenFlow: Enabling Innovation in Campus Networks.** 14 mar. 2008. Disponível em: <http://archive.openflow.org/documents/openflow-wp-latest.pdf>. Acesso em: 05 nov. 2015.

ANSIBLE. **Documentation.** Disponível em: <http://docs.ansible.com/>. Acesso em: 14 ago. 2015.

ARCHLINUX. **Secure Shell.** Disponível em: <https://wiki.archlinux.org/index.php/Secure_Shell>. Acesso em: 01 dez. 2015.

BABCOCK, C. Cisco Brings DevOps To The Network. **Information Week**, 21 jul. 2014. Disponível em: <http://www.informationweek.com/cloud/platform-as-a-service/cisco-brings-devops-to-the-network/d/d-id/1297424>. Acesso em: 14 ago. 2015.

BACHELDOR, Beth. **IT Automation.** 2011. Disponível em : <http://www.ca.com/~/media/files/whitepapers/st_of_itauto_ca_v5a.pdf>. Acesso em: 05 nov. 2015.

BURKE, A. Pull My Strings, I'm Your Puppet: Juniper Bringing DevOps To Networking. **Packet Pushers**, 21 fev. 2013. Disponível em: <http://packetpushers.net/pull-my-strings-im-your-puppet-juniper-bringing-devops-to-networking/>. Acesso em: 14 ago. 2015.

BOOTSTRAP. **The world's most popular mobile-first and responsive front-end framework.** Disponível em: <http://getbootstrap.com/>. Acesso em: 05 nov. 2015.

CASE, et. al. **A Simple Network Management Protocol (SNMP).** Mai. 1990. Disponível em : <https://tools.ietf.org/html/rfc1157>. Acesso em: 05 nov. 2015.

CHEF. **Documentation.** Disponível em: <https://docs.chef.io/>. Acesso em: 14 ago. 2015.

CHERRYPY. **A Minimalist Python Web Framework.** Disponível em: <http://www.cherrypy.org/>. Acesso em: 01 dez. 2015.

CISCO. **Network Configuration Protocol.** Disponível em: <http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/cns/configuration/xe-3s/cns-xe-3s-book/cns-netconf.html>. Acesso em: 05 nov. 2015.

_____. **Cisco Open SDN Controller.** Disponível em: <http://www.cisco.com/c/en/us/products/cloud-systems-management/open-sdn-controller/index.html>. Acesso em: 05 nov. 2015.

CONDE, Daniel . **Network Automation Benefits.** 07 jun. 2015b. Disponível em : <http://www.networkcomputing.com/networking/network-automation-benefits/a/d-id/1321132>. Acesso em: 05 nov. 2015.

DUVALL, Paul. **Agile DevOps: Infrastructure automation.** 11 set. 2012. Disponível em : <http://www.ibm.com/developerworks/library/a-devops2/>. Acesso em: 05 nov. 2015.

EVANS, Clark C. **YAML Ain't Markup Language** Disponível em: <http://www.yaml.org/>. Acesso em: 15 nov. 2015.

FEAMSTER, N.; Rexford, K.; ZEGURA, E. The Road to SDN. **Queue,** vol. 11, no. 12. dez. 2013. Disponível em: <http://queue.acm.org/detail.cfm?id=2560327>. Acesso em: 14 ago. 2015.

GARLING, Caleb. **Why the GUI Will Never Kill the Sacred Command Line.** 07 de out. 2012. Disponível em: <http://www.wired.com/2012/07/command-line/>. Acesso em:

01 dez. 2015.

GREENE, D. What is DevOps? **Tech Crunch,** 15 mai. 2015. Disponível em: <http://techcrunch.com/2015/05/15/what-is-devops/>. Acesso em: 14 ago. 2015.

HIGGINBOTHAM, S. Cisco gets the devops religion as software eats networking. **GIGAOM**, 21 jul. 2014. Disponível em: <https://gigaom.com/2014/07/21/cisco-gets-the-devops-religion-as-software-eats-networking/>. Acesso em: 14 ago. 2015.

IBM. **The SNMP architecture.** Mai. 2012. Disponível em : <https://www-01.ibm.com/support/knowledgecenter/SSGU8G_11.70.0/com.ibm.snmp.doc/ids_snmp_009.htm>. Acesso em: 05 nov. 2015.

JACOB, Robert. **User Interfaces.** 2000. Disponível em: <http://web.media.mit.edu/~anjchang/ti01/rjp.html>. Acesso em: 05 nov. 2015.

JUNIPER NETWORKS. **Network Automation and Orchestration**, abr. 2015. Disponível em: <https://www.juniper.net/assets/us/en/local/pdf/whitepapers/2000541-en.pdf>. Acesso em: 14 ago. 2015.

KIM, H.; FEAMSTER, N. Improving Network Management with Software Defined

KIRK, Byers. **Netmiko Library.** 19 jun. 2015. Disponível em: <https://pynet.twb-tech.com/blog/automation/netmiko.html>. Acesso em: 01 dez. 2015.

_____. **Netmiko Library.** Disponível em: <https://github.com/ktbyers/netmiko>. Acesso em: 01 dez. 2015.

LEVI, et. al. **Simple Network Management Protocol (SNMP) Applications.** Dez. 2002. Disponível em : <https://tools.ietf.org/html/rfc3413>. Acesso em: 05 nov. 2015.

LERNER, A. Network Automation. **Gartner**, 5 ago. 2014. Disponível em: <http://blogs.gartner.com/andrew-lerner/2014/08/05/network-automation/>. Acesso

em: 14 ago. 2015.

LUNDEN, Ingrid. **Red Hat Is Buying IT Automation Startup Ansible, For $150M.** 16 out. 2015. Disponível em : <http://techcrunch.com/2015/10/16/red-hat-is-buying-it-automation-startup-ansible-reportedly-for-around-100m/#.4o4mjvl:U7wb>. Acesso em: 05 nov. 2015.

NIELSEN, Jakob. **Usability 101: Introduction to Usability.** 04 jan. 2012. Disponível em: <http://www.nngroup.com/articles/usability-101-introduction-to-usability/>. Acesso em: 05 nov. 2015.

Networking. **Georgia Institute of Technology**, fev. 2013. Disponível em: <http://gtnoise.net/papers/2013/kim-ieee2013.pdf>. Acesso em: 14 ago. 2015.

NETWORK TO CODE. **NTC-ANSIBLE.** Disponível em: <https://github.com/networktocode/ntc-ansible>. Acesso em: 01 dez. 2015.

OGENSTAD, Patrick. **Ansible modules using SNMP to manage Cisco devices.** Disponível em: <https://github.com/networklore/ansible-cisco-snmp>. Acesso em: 05 nov. 2015.

OPEN DAY LIGHT. **Lithium Overview.** Disponível em: <https://www.opendaylight.org/lithium>. Acesso em: 14 ago. 2015.

OPEN NETWORKING FOUNDATION. **Software-Defined Networking (SDN) Definition**, 2015. Disponível em: <https://www.opennetworking.org/sdn-resources/sdn-definition>. Acesso em: 14 ago. 2015.

OPENFLOW. **OpenFlow Learn more.** Disponível em : <http://archive.openflow.org/wp/learnmore/>. Acesso em: 05 nov. 2015.

PERILLI, Alessandro. **Why Red Hat Acquired Ansible.** 16 out. 2015. Disponível em : <http://www.redhat.com/en/about/blog/why-red-hat-acquired-ansible>. Acesso

em: 05 nov. 2015.

POTT, Trevor. **Sysadmins! There's no shame in using a mouse to delete files.** 26 jul. 2012. Disponível em: <http://www.theregister.co.uk/2012/07/26/cli_vs_gui/>. Acesso em: 01 dez. 2015.

PUPPET LABS. **Puppet Documentation Index.** Disponível em: <http://docs.puppetlabs.com/puppet/>. Acesso em: 14 ago. 2015.

_____. **Top 10 Business Benefits of IT Automation.** Vídeo (30 min 34 seg.). 02 dez. 2014. Disponível em : <https://puppetlabs.com/webinars/top-10-business-benefits-it-automation>. Acesso em: 05 nov. 2015.

PYSNMP. **SNMP library for Python.** Disponível em : <http://pysnmp.sourceforge.net/>. Acesso em: 05 nov. 2015.

RED HAT. **Red Hat to Acquire IT Automation and DevOps Leader Ansible.** 16 out. 2015. Disponível em : <https://www.redhat.com/en/about/press-releases/red-hat-acquire-it-automation-and-devops-leader-ansible>. Acesso em: 05 nov. 2015.

RUPP, Jens; ZOBEL, Daniel. Introducing SNMP. In: **Quo Vadis, SNMP?** Mar. 2014. Disponível em : <https://www.paessler.com/press/whitepapers/introducing_snmp/part-1>. Acesso em: 05 nov. 2015.

SALTSTACK. **Documentation.** Disponível em: <http://docs.saltstack.com/en/latest/>. Acesso em: 14 ago. 2015.

SLATTERY, Terry. **Top 7 reasons for network automation.** 25 ago. 2010. Disponível em : <http://www.netcraftsmen.com/top-7-reasons-for-network-automation/>. Acesso em: 05 nov. 2015.

SULTAN, Omar. **OpenDaylight: The Start of Something Big for SDN.** 08 abr. 2013. Disponível em: <http://blogs.cisco.com/datacenter/opendaylight-the-start-of-something-big-for-sdn>. Acesso em: 05 nov. 2015.

TAIL-F. **What is NETCONF?** Disponível em: <http://www.tail-f.com/education/what-is-netconf/>. Acesso em: 05 nov. 2015.

TIMMS, Natalie. **NETCONF: Introduction To An Emerging Networking Standard.** 20 jun. 2014. Disponível em: <http://www.networkcomputing.com/networking/netconf-introduction-to-an-emerging-networking-standard/d/d-id/1278724>. Acesso em: 05 nov. 2015.

TECHNET. **What Is SNMP?** 28 mar. 2003. Disponível em : <https://technet.microsoft.com/en-us/library/cc776379%28v=ws.10%29.aspx?f=255&MSPPError=-2147217396#w2k3tr_snmp_what_smbc>. Acesso em: 05 nov. 2015.

YLONEN, T; LONVICK, Ed.**The Secure Shell (SSH) Protocol Architecture.** Jan. 2006. Disponível em: <https://tools.ietf.org/html/rfc4251>. Acesso em: 01 dez. 2015.

WILSON, N. The rise of the DevOps culture, and why it's important. **Tech Radar**, 19 jan. 2015. Disponível em: <http://www.techradar.com/news/world-of-tech/management/the-rise-of-the-devops-culture-and-why-it-s-important-1281130>. Acesso em: 14 ago. 2015.