UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RAFAEL LUÍS LOURENÇO

IMPLEMENTAÇÃO DA AUTOMAÇÃO DE REDES ATRAVÉS DO ANSIBLE PARA CONFIGURAÇÃO DE UMA ARQUITETURA DE REDE MARKET DATA

Porto Alegre

2021

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

RAFAEL LUÍS LOURENÇO

IMPLEMENTAÇÃO DA AUTOMAÇÃO DE REDES ATRAVÉS DO ANSIBLE PARA CONFIGURAÇÃO DE UMA ARQUITETURA DE REDE MARKET DATA

Relatório Final do Projeto de Diplomação II apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para Graduação em Engenharia Elétrica

Orientador: Prof. Dr. Ivan Müller

Porto Alegre
2021

RAFAEL LUÍS LOURENÇO

IMPLEMENTAÇÃO DA AUTOMAÇÃO DE REDES ATRAVÉS DO ANSIBLE PARA CONFIGURAÇÃO DE UMA ARQUITETURA DE REDE MARKET DATA

Relatório Final do Projeto de Diplomação II apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para Graduação em Engenharia Elétrica

---

Prof. Dr. Ivan Müller
Orientador - UFRGS

Prof. Dr. Roberto Petry Homrich
Chefe do Departamento de Engenharia Elétrica (DELET) - UFRGS

BANCA EXAMINADORA

Prof. Dr. Ivan Müller
UFRGS

Prof. Dr. Marcelo Götz
UFRGS

Dr. Deivid Antunes Tesch
PUCRS

Aos meus pais, Adão e Giane, por todo amor, apoio e suporte proporcionados durante a minha trajetória

AGRADECIMENTOS

Agradeço, primeiramente, ao meu pai Adão e minha mãe Giane pela construção de toda a base que me fez chegar até aqui. Agradeço por todo amor, suporte e compreensão que me foram desprendidos, sempre me proporcionando as melhores condições possíveis e impossíveis. Essa conquista é para vocês.

Agradeço a minha irmã Carolina e minha irmã Luísa, as quais sinto muitas saudades da convivência diária, por sempre me motivarem a ser a melhor versão de mim mesmo. Eu tenho um orgulho imenso das duas.

Agradeço a minha namorada Amanda por ser meu Porto Seguro. Agradeço por todo amor, pelo companheirismo, pela cumplicidade e pelos incríveis e motivadores planos futuros. Agradeço, ainda, pela paciência em lidar com minhas angústias neste período turbulento de conclusão. Tive a sorte enorme em te encontrar, obrigado por todos os nossos momentos e que venham muitos mais.

Agradeço aos demais membros da minha extensa família, ao qual tenho o privilégio de fazer parte, por sempre me acolherem e estarem na torcida por mim. Posso afirmar que me sinto em casa em todos os lares por onde vou.

Agradeço à Universidade Federal do Rio Grande do Sul por me proporcionar um ensino superior de alta qualidade e uma oportunidade de conhecer diversos amigos nesses anos de graduação.

Agradeço ao Professor Ivan Müller, meu orientador, por além de ser um profissional com grandes conhecimentos técnicos, também ser um professor atencioso e estar sempre disposto a auxiliar todos os alunos. Não tenho dúvidas que as disciplinas ministradas por ele aguçaram minha curiosidade sobre o mundo das redes.

Agradeço aos meus colegas de apartamento Felipe, Leonardo B. e Leonardo V. por toda a parceria e união formada nestes anos. Agradeço pela forma de como cada um, do seu jeito, contribui para meu modo de ver e pensar o mundo. Jamais vou esquecer das produtivas discussões sentados à mesa de jantar.

Agradeço ao Unidos do Andrelson e agregados por tornarem a vida mais leve e feliz. Na companhia de vocês, nunca há tempo ruim. Sou muito grato por todas risadas compartilhadas.

Agradeço à Associação Atlética da Escola de Engenharia por me proporcionar as melhores experiências no ambiente extraclasse durante a graduação. Em especial, agradeço a Bruna, ao Jaison e a Mariana pelo carinho genuíno e por nossa "família".

Agradeço ao Grupo de Dúvidas pelas longas horas de estudo e incentivo que culminaram em uma grande amizade. Vocês foram peça fundamental do início ao fim do curso.

Agradeço ao Deivid pela capacidade de transmitir diversos conhecimentos complexos de forma didática e clara. Agradeço pela disponibilidade e paciência em sempre ajudar a todos ao redor. Admiro o teu profissionalismo que me serve de exemplo e inspiração.

Agradeço aos meus demais colegas de RBS TV e Nelogica por terem compreendido meus períodos de ausência e tornarem a jornada de trabalho mais agradável.

A mente que se abre a uma nova ideia
jamais voltará ao seu tamanho original

Albert Einstein

RESUMO

A crescente demanda global do tráfego de dados resulta em infraestruturas de rede cada vez mais amplas, tornando a configuração manual dos diversos equipamentos pouco eficiente e com uma maior propensão a erros humanos. No intuito de aprimorar os processos envolvidos na administração dessas infraestruturas, a automação de redes apresenta-se como uma solução emergente para essa finalidade. No presente trabalho, foi proposto e implementado, no *software* de emulação EVE-NG, um modelo de arquitetura de rede para *Market Data* que inclui as fontes para transmissão de *Market Data* (Bolsa de Valores), o provedor de serviços financeiros e os consumidores finais dos dados (Corretoras). A topologia proposta divide-se entre dois *Feeds* de dados, *Feed A* e *Feed B*, constituídos por roteadores com sistema operacional Cisco IOS. Para a implementação do *Feed A* foram configurados manualmente: uma *Multicast VPN* na rede MPLS *Core* e o protocolo *PIM Sparse Mode* nos roteadores de borda. As configurações no *Feed B* seguiram os mesmos critérios, porém foram executadas através da automação de redes utilizando o Ansible. Foram escritos os *playbooks* responsáveis pela configuração de todos os roteadores dispostos no *Feed B*. Validou-se as configurações implementadas através de transmissões *multicast* geradas na rede, executando comandos de verificação nos roteadores da topologia. Por fim, analisou-se a viabilidade do Ansible para automatizar as configurações de rede, comparando as configurações dos roteadores do *Feed B* antes e depois da execução dos *palybooks* desenvolvidos. Verificou-se que as configurações foram aplicadas conforme o esperado, mostrando a eficácia do Ansible na automação de configurações em roteadores *Cisco* IOS.

Palavras-chave: Automação de Redes, Ansible, *Market Data*, *multicast*.

ABSTRACT

The global raise demand for data traffic results in an increase of large network infrastructure, making the manual settings of several devices inefficient and prone to human errors. In order to improve the process involved in the infraestructure administration, the network automation presents itself as an emerging solution for this purpose. In this work, a network architecture model for Market Data was proposed and implemented in the EVE-NG network emulation software. The scope of the Market Data Network Architecture includes the sources for market data streams (stock exchanges), the Financial Service Providers and the final consumers of the data (Brokerage Houses). The proposed topology was divided between two data Feeds, A Feed and B Feed, consisting of Cisco IOS operating systems routers. The implementation of A Feed was manually configured a Multicast VPN on the MPLS Core network and the PIM Sparse Mode protocol on the edge routers. The settings in B Feed followed the same criteria, but were performed through Ansible network automation. It was written the playbooks in charge for configuring all the routers of B Feed. The implemented configurations were validated through multicast streams in the network by the execution of show commands on the topology routers. Finally, it was analyzed the feasibility of Ansible for network automation, comparing the configurations of the B Feed routers before and after the execution of the developed Ansible playbooks. The configurations were applied as expected, showing the effectiveness of Ansible for network automation on Cisco IOS routers.

Keywords: Network Automation, Ansible, Market Data, multicast.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AES	*Advanced Encryption Standard*
API	*Application Programming Interface*
ASN	*Autonomous System Number*
BGP	*Border Gateway Protocol*
CE	*Customer Edge*
CLI	*Command Line Interface*
DM	*Dense Mode*
DR	*Designated Router*
FAST	*FIX Adapted for Streaming*
FEC	*Forwarding Equivalence Class*
FIX	*Financial Information eXchange*
GCP	*Google Cloud Platform*
HTTPS	*Hyper Text Transfer Protocol Secure*
IGMP	*Internet Group Management Protocol*
IP	*Internet Protocol*
LAN	*Local Area Network*
LDP	*Label Distribution Protocol*
LSP	*Label Switch Path*
LSR	*Label Switch Router*
MAC	*Media access control*
MDT	*Multicast Distribution Tree*
mLDP	*Multicast Label Distribution Protocol*
MP2MP	*Multipoint-to-Multipoint*

MPLS	*Multiprotocol Label Switching*
MVPN	*Multicast Virtual Private Network*
NAT	*Network Address Translation*
NFV	*Networking function virtualisation*
ONF	*Open networking foundation*
OSPF	*Open Shortest Path First*
P	*Provider*
P2MP	*Point-to-Multipoint*
PE	*Provider Edge*
PIM	*Protocol Independent Multicast*
RIB	*Routing Information Base*
RP	*Rendezvous Point*
SDN	*software defined network*
SM	*Sparse Mode*
SOH	*Start of Header*
SSH	*Secure Shell*
SSL	*Secure Sockets Layer*
SSM	*Source Specific Multicast*
TCP	*Transmission Control Protocol*
TI	*Tecnologia da Informação*
UDP	*User Data Protocol*
UTF	*Unicode Transformation Format*
VPN	*Virtual Private Network*
VRF	*Virtual Routing and Fowarding*
WAN	*Wide Area Network*
XML	*eXtensible Markup Language*
YAML	*YAML Ain't Markup Language*

SUMÁRIO

1 INTRODUÇÃO

As redes de computadores têm crescido exponencialmente (COMER, 2016). De acordo com a pesquisa realizada por (SYNERGY, 2019), a receita dos fabricantes de *switches* e roteadores ultrapassou a marca de 44 bilhões de *dólares* em 2018, registrando o recorde histórico. Em função desse constante crescimento, as redes, com uma ampla quantidade de dispositivos, tornam-se cada vez mais complexas e dinâmicas (KIM; FEAMSTER, 2013). Além disso, as redes comumente expandem-se de forma heterogênea, possuindo ativos com *hardware* e *software* proprietários desenvolvidos por diferentes empresas (MENZEN, 2015). O protocolo TCP/IP permite a interoperabilidade entre essa pluralização de equipamentos, porém, traz desafios no gerenciamento de configurações para os administradores de rede.

Para (FEAMSTER; REXFORD; ZEGURA, 2013), essa diversificação acaba por ser uma barreira administrativa, principalmente tratando-se do provisionamento e manutenção dos dispositivos. Para determinar as políticas que dão o comportamento às redes, os administradores, usualmente, executam as configurações via CLI (interface de linha de comandos). Nesse sentido, conforme (YUNGA, 2018), em uma rede com um grande número de ativos e diferentes sistemas operacionais, torna-se inviável a gestão individual e a escalabilidade dos equipamentos.

Como forma de contornar esses e outros desafios, surgiu a ideia de redes programáveis. As SDN (Redes definidas por *software*), conforme especificada em (NUNES et al., 2014), é um paradigma de rede no qual o controle é centralizado logicamente em controladores baseados em *software* e os dispositivos físicos tornam-se apenas encaminhadores de pacotes. Apesar de apresentar-se como uma solução para os desafios citados anteriormente, a implementação de uma SDN exige um processo extremamente complexo, principalmente em ambientes produtivos. Todavia, diversos conceitos surgiram a partir dos estudos de SDN, entre eles, a capacidade de automatizar o provisionamento e os processos de configuração e administração de toda a infraestrutura de rede (MELO, 2018).

A automação de redes é o processo que automatiza a configuração, o gerenciamento, os testes, a implantação e a operação dos dispositivos físicos e virtuais em uma rede. (CISCO, 2019). De acordo com (CARDOSO, 2019), a automação surgiu como principal aliada na eficiência e agilidade de diversos processos de rede, reduzindo a execução de tarefas morosas e a propensão de erros humanos. Uma característica na automação é tratar a infraestrutura como código. A infraestrutura como código permite transpor a infraestrutura de rede em arquivos de configuração, trazendo provisionamento e manutenção da infraestrutura com uma abordagem similar ao empregado em *software* (HüTTERMANN, 2012).

Ainda conforme (HüTTERMANN, 2012), um dos impulsionadores da implementação da automação de redes é o conceito cultural de *DevOps*, que surgiu em 2008 e deriva da ideia de metodologia ágil e livre de erros. A palavra *DevOps* é a união de desenvolvimento e operações, sendo cada vez mais amplamente utilizada na indústria de TI. No âmbito corporativo, a equipe de *DevOps* é responsável por fazer o uso de *frameworks* de automação de infraestrutura, como *Ansible*, *Chef*, *Puppet* e *SaltStack*, que podem contemplar desde equipamentos de rede até servidores (MENZEN, 2015). A pesquisa realizada por (MARKETS&MARKETS, 2021), indica que a projeção do mercado de automação de redes para 2025 é de aproximadamente 8,9 bilhões de dólares, um crescimento de 24,8% em relação a 2020.

Com base na relevância do tema proposto, o presente trabalho apresenta a construção de um modelo de uma arquitetura de rede *Market Data* em um *software* de emulação de dispositivos de rede. Para fins de análise da automação de redes, parte das configurações presentes nos equipamentos desta arquitetura foram realizadas pelo Ansible, uma das principais ferramentas de automação de redes existentes no mercado. O *Market Data* é um termo utilizado no mercado de capitais para referenciar-se aos dados de difusão do mercado, incluindo o livro de ofertas do mercado de ações e derivativos (B3, 2019). De acordo com (CISCO, 2008), o escopo de uma arquitetura de rede *Market Data* inclui: as fontes para transmissão de *Market Data* (Bolsa de Valores), o provedor de serviços financeiros e os consumidores finais dos dados (Corretoras).

A escolha por automatizar uma arquitetura de rede *Market Data* deu-se, principalmente, pela ampla quantidade de roteadores e protocolos de rede envolvidos. Ainda, a complexidade da configuração devido às características de resiliência e baixa latência que podem induzir a erros de configuração, o que possibilita explorar de forma abrangente as funcionalidades do Ansible. A transmissão de *Market Data*, do ponto de vista do protocolo TCP/IP, é um *stream multicast* transportado sobre o protocolo UDP (*User Datagram Protocol*). Para que os dados sejam transmitidos, por exemplo, das Bolsas de Valores até as Corretoras, é necessária uma rede MPLS (*Multiprotocol Label Switching*), fornecida por um provedor de serviços, com suporte a tráfego *multicast*. Para isso, configura-se uma MVPN (*Multicast Virtual Private Network)* entre os roteadores de borda do provedor de serviços e o protocolo PIM-SM (*Protocol Indepent Multicast - Sparse Mode*) nos roteadores de borda das Bolsas e Corretoras.

Nesse sentido, cada Bolsa de Valores distribui o mesmo sinal de *Market Data* em dois canais distintos, conhecidos por *Feed A* e *Feed B*, a fim de garantir uma redundância física do sistema. Dessa forma, a validação do uso da automação com o Ansible ocorre na configuração, gerada pelos *playbooks* desenvolvidos, dos roteadores do *Feed B*. Os resultados obtidos revelam a viabilidade do Ansible para automatizar as configurações de rede em roteadores *Cisco* IOS.

Este trabalho está estruturado em cinco tópicos principais:

- Referencial Teórico: uma abordagem breve dos conceitos utilizados no trabalho;
- Materiais e Métodos: onde são descritos os materiais utilizados e as etapas que guiaram o desenvolvimento da solução proposta;
- Resultados e Discussões: apresenta os resultados das análises da topologia implementada em computador e dos testes de resiliência realizados;
- Conclusões e trabalhos futuros: considerações finais sobre os resultados obtidos e necessidades de melhorias. Apresentação de ideias de novos trabalhos que surgiram após a realização deste trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo serão descritos os conceitos de automação de redes, o princípio de funcionamento do Ansible e os fundamentos presentes em uma arquitetura de rede *Market Data*.

2.1 Fundamentos da Automação de Redes

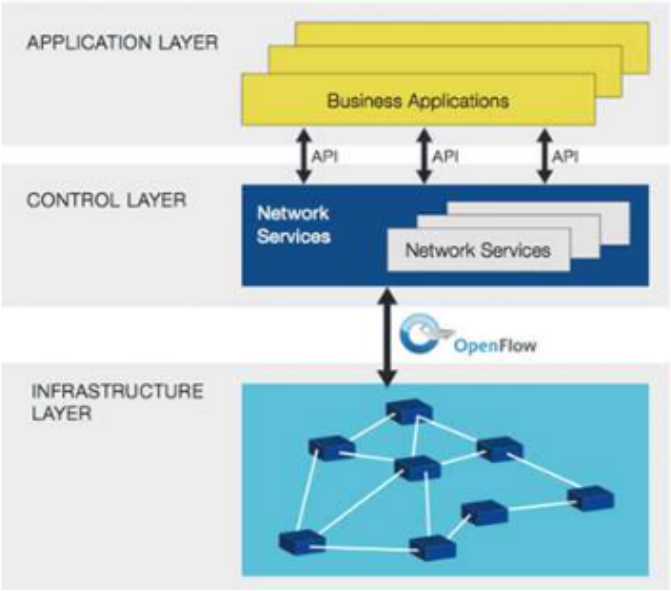
O mundo das redes está passando por uma grande mudança nos últimos anos (VENKATESH, 2018). Com o crescimento da demanda por novos serviços, aplicações e quantidade de dados a serem armazenados, é exigido dos engenheiros uma rede cada vez mais ágil e eficiente. Além disso, outro requisito é a visibilidade global da rede de ponta a ponta e o fluxo de controle granular. Para esses ambientes, que crescem em ritmo acelerado, a agilidade da rede pode ser alcançada por meio das SDNs, NFV (*Network Function Virtualization*) e redes programáveis.

2.1.1 Redes Definidas por Software

O conceito de SDN surgiu em 2008 nas Universidades de Berkeley e Stanford com a premissa de desagregar as funções de encaminhamento de dados das funções de controle. Nas redes tradicionais, uma vez que a política de encaminhamento dos pacotes tenha sido definida, a única maneira de realizar uma mudança é alterando as configurações dos dispositivos de rede (GIESEN; OLIVEIRA, 2017). Com abordagem distinta, a SDN é caracterizada pela existência de um controle, realizado por *software*, que permite inspecionar, definir e alterar entradas da tabela de fluxo. A arquitetura proposta pela ONF (*Open Network Foundation*) é um modelo de SDN com três camadas: a camada de aplicação, que serve para os usuários finais realizarem a implantação dos serviços de rede; a camada de controle, responsável pelo controle centralizado e supervisão do encaminhamento dos dados e a camada de infraestrutura, que consiste nos dispositivos de rede que realizam o encaminhamento dos pacotes. Na Figura 1 está ilustrado o modelo de SDN.

Embora a SDN possua as características de agilidade, escalabilidade e flexibilidade, faz-se necessária uma mudança fundamental na forma como as redes são construídas e operadas. A SDN é uma solução bastante relevante para os provedores de serviço, aqueles nos quais o ambiente é de grande porte. As empresas ainda estão encontrando dificuldades em adotar essa tecnologia. Ainda, conforme (GEDIA; PERIGO, 2018), integrar SDN com as redes tradicionais é difícil devido à diferença de operação entre elas: as redes tradicionais

Figura 1 – Camadas do Modelo SDN



Fonte: retirado de (GIESEN; OLIVEIRA, 2017)

operam através do *MAC Adress* e das tabelas de roteamento, enquanto SDN com *OpenFlow* usa entradas de fluxo em tabelas de fluxo. Dessa forma, as redes programáveis são a resposta para as necessidades das empresas, porque podem automatizar e gerenciar a rede através de código sem alterar a infraestrutura existente (VENKATESH, 2018).

2.1.2 Automação de Infraestrutura de TI

A automação, conforme descrito por (CAMOES; SILVA, 2018), é a técnica de tornar um processo ou sistema automático, permitindo melhorar diversas operações que controlam, regulam e administram máquinas, com pouca ou nenhuma influência humana. Espera-se que um sistema automatizado realize uma função de forma mais confiável, eficiente e precisa do que um operador humano. Para (MENZEN, 2015), a automação também auxilia na redução das complexidades dos processos, aumenta a produtividade, a inovação e a solidez nas entregas ágeis. Segundo (SCARPATI, 2017), a automação pode ser aplicada em qualquer tipo de rede, incluindo as redes de área local (LANs), redes de longa distância (WANs), redes de *Datacenters*, redes de *clouds* e redes sem fio. Ou seja, qualquer recurso de rede configurado via linha de comando ou API (*Application Programming Interface*) pode ser automatizado.

De acordo com (YUNGA, 2018), existem dois tipos de automação de redes: orientada por *scripts* e por *software*. A automação orientada por *scripts* emprega linguagens de *scripting* para executar sempre a mesma tarefa de forma consistente. Linguagens de

programação como *Python* e *Ruby* estão sendo cada vez mais utilizadas devido ao seu fácil uso e flexibilidade, apesar de ainda existir diversos *scripts* escritos em linguagens legadas como *Perl* e *Tcl*. Conforme indicado por (SATO, 2014), os *scripts* executam tarefas que alteram o estado dos equipamentos, podendo causar problemas se a mudança for mal executada. Além disso, essa forma de automação não é facilmente reutilizável em outras situações, conforme cresce o número de equipamentos. Já a automação de rede baseada em *software*, também conhecida como automação de rede inteligente de acordo com (SCARPATI, 2017), é administrada por uma ferramenta de automação que elimina a necessidade de desenvolver *scripts* manualmente na maioria dos casos. Isso ocorre porque as ferramentas geralmente fornecem modelos para criar e executar tarefas com base em políticas de linguagem simples.

A automação baseada em *software* permite tratar a infraestrutura como código, ou seja, é uma prática em que a infraestrutura é provisionada e gerenciada utilizando técnicas de desenvolvimento de código, funcionando como um controle de versão e integração contínua (CAMOES; SILVA, 2018). Dessa forma, segundo (REDHAT, 2018), a automação de infraestrutura de TI traz diversos benefícios:

- Provisionamento: é o processo de definição da infraestrutura de TI. Ele também se refere às etapas necessárias para gerenciar o acesso aos dados e recursos e para disponibilizá-los a usuários e sistemas.
- Gerenciamento de configuração: responsável por manter a padronização e o *compliance* e agir em sintonia com as regras e normas de políticas;
- Orquestração: possibilita automatizar processos ou fluxos de trabalho que incluem várias etapas em diferentes sistemas. Depois de incorporar a automação aos processos, é possível orquestrá-los para execução automática.
- Gerenciamento de estados: sendo a principal característica das ferramentas que gerenciam configurações, porque é possível definir os estados desejados para o sistema. Uma abordagem diferente do que ocorre com *scripts*.
- Segurança e conformidade: sendo a padronização de processos de segurança e fluxos de trabalho facilitadores da conformidade e da auditoria.

2.1.3 Ferramentas para Automação de Redes

Existem dois tipos principais de metodologias de automação usando ferramentas de *DevOps*: sem e com agente. No primeiro, *agentless*, não é requerida nenhuma aplicação de *software* adicional, ou seja, não é necessária a instalação de nenhum *software* nos *switches* e roteadores. Para ser funcional, esse modelo depende de um processo já existente para

comunicar-se com equipamentos de rede, como o SSH *Secure Shell*. Dessa forma, pode ser considerada como um sistema *push*, em que as configurações são "empuradas" da estação de gerenciamento para os equipamentos de rede. Já o tipo de metodologia baseada em agente requer um cliente instalado nos *switches* e roteadores para comunicar-se com o servidor de controle. Nesse caso, um agente fica em execução nos equipamentos de rede e a função dele é conectar-se com o servidor de gerência para solicitar as alterações e reportar informações. Esse método chama-se *pull*.

A vantagem do método *push* é a execução remota imediata de configuração, ou seja, assim que realizada a alteração, a nova configuração já passa a valer instantaneamente. A desvantagem desse método é justamente a necessidade de um operador ou processo iniciar a mudança, além do fato dos equipamentos de rede precisarem estar disponíveis nesse momento. Com relação às vantagens do método *pull*, o agente instalado nos clientes "puxa" a configuração do servidor mestre, logo não é necessário inciar as configurações manualmente. Porém, as configurações só serão validadas após um determinado período de tempo. O modo *pull* é mais escalável, em contrapartida é mais difícil de controlar quando todos os clientes estão requisitando informações do servidor mestre. Dentre as ferramentas que utilizam ambas metodologias, serão abordados os conceitos das principais presentes no mercado que, de acordo com (YUNGA, 2018), são: Chef, Puppet, SaltSatck e Ansible. Dentre as ferramentas citadas, o presente trabalho explora apenas o Ansible. Porém, por serem alternativas para automação cada vez mais populares na cultura *DevOps*, optou-se por apresentá-las de forma sucinta.

2.1.4 Ansible

O Ansible é uma plataforma *Open Source* para criação de aplicações, módulos e *scripts* de automação, provisionamento e orquestração para infraestrutura de TI. O Ansible é desenvolvido em *Python* e foi concebido em 2012 por Michael DeHaan. Ele trabalha com a metodologia *push* e ganhou elevada popularidade devido a sua simplicidade de operação, ser modular e possuir uma grande quantidade de desenvolvedores. Atualmente, o Ansible possui mais de 5000 colaboradores na sua comunidade do *Github*. A ferramenta foi adquirida pela empresa *Red Hat* em 2013 que lançou sua versão *Enterprise*, o Ansible Tower. De acordo com a Red Hat, o Ansible é uma ferramenta desenhada para suportar a implantação e configuração de qualquer ambiente de TI, especialmente redes, nuvem e até contêineres.

O Ansible usa arquivos do tipo YAML (*YAML Ain't Markup Language*) como principal fonte de informações em tempo de execução (HEAP, 2016). O YAML é uma linguagem de representação de dados geralmente usada para configurações e é um formato legível por humanos. Os *scripts* do Ansible são chamados de *Playbooks*, que também é um arquivo do tipo *YAML*. O objetivo de um *playbook* é aplicar configurações em um conjunto

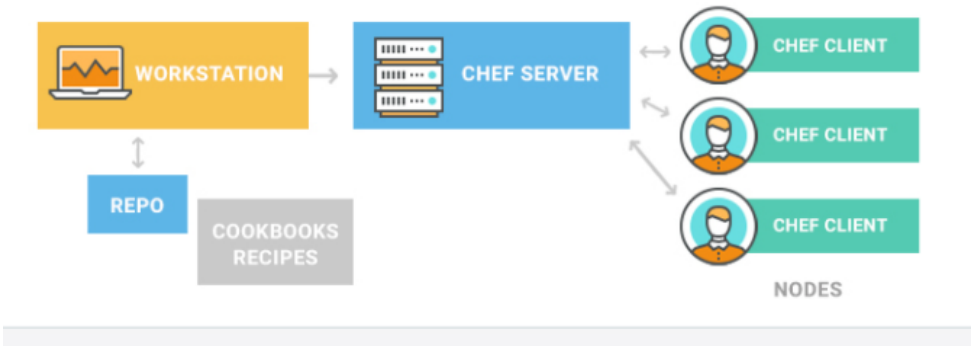
de dispositivos a partir da execução de uma sequência de *roles*. As *roles* são agrupadas por várias *tasks*, que são entradas que designam uma única ação. As *roles* executarão várias *tasks*, assegurando que o estado de cada dispositivo é como se pretende que seja. Um aprofundamento maior do Ansible será apresentado na seção 2.2.

2.1.5 Chef

Conforme descrito por (DUVALL, 2012), o Chef é uma ferramenta de administração de configuração de código aberto desenvolvida pela comunidade Opscode em 2008. O Chef é escrito em *Ruby* e é compatível com uma variedade de sistemas operacionais. O Chef, assim como o Ansible, possui uma versão *Open Source* e outra versão proprietária. A ferramenta, além da capacidade de automação, também pode realizar a análise de todo o ambiente gerenciado e criar relatórios de erro. O Chef é *framework* do tipo *pull*, ou seja, faz-se necessário a utilização de um tipo de agente nos nós que serão controlados.

A arquitetura do Chef está ilustrada na Figura 2 e demonstra seus três componentes principais: Chef *server*, Chef *nodes* e Chef *workstation*. O Chef executa *cookboks*, consistindo em *recipes* que realizam ações nos nós. O Chef *workstation* funciona como um repositório local onde está instalado o *Knife*, que permite carregar os *cookbooks* em um Servidor Chef. Para que as ações possam ser executadas, o cliente Chef se comunica com o Servidor para obter as configurações e executa as operações necessárias.

Figura 2 – Arquitetura do Chef



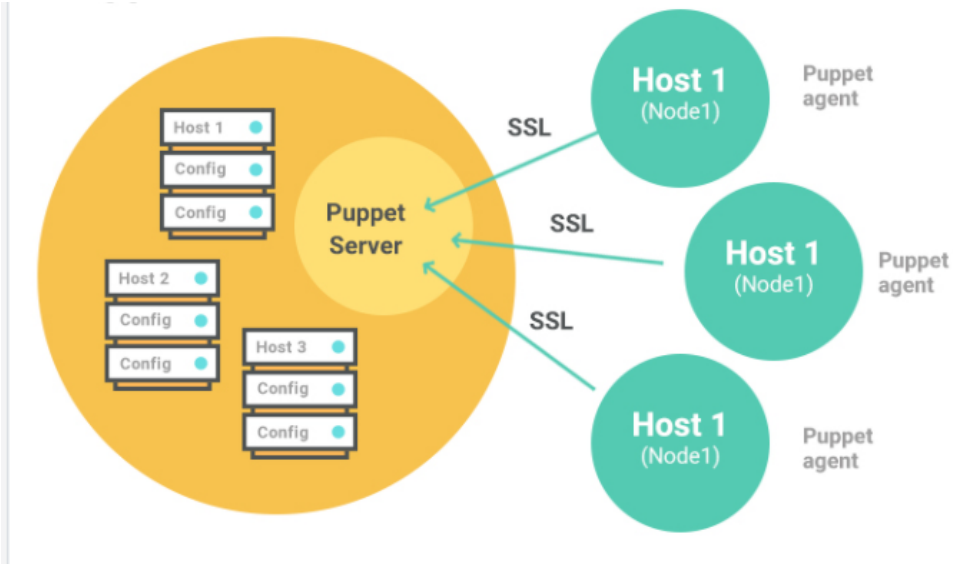
Fonte: retirado de (YIGAL, 2017)

O Chef é uma ferramenta bastante estável e possui uma documentação robusta, sendo recomendada para implementação em larga escala. Porém, acaba por ser uma ferramenta complexa e de difíceis configurações iniciais (RAZA, 2016).

2.1.6 Puppet

O Puppet, outra ferramenta de automação disponível no mercado, é mantido pela empresa Puppet Labs e escrito em *Ruby*. O Puppet surgiu como um projeto de código aberto e também passou a ter uma versão paga chamada Puppet Enterprise. A arquitetura do Puppet está demonstrada na Figura 3 e, assim como Chef, é do tipo cliente-servidor *(pull)*.

Figura 3 – Arquitetura do Puppet



Fonte: retirado de (YIGAL, 2017)

Os agentes são instâncias do Puppet instaladas em cada *Host*, como por exemplo, servidores ou dispositivos de rede. Os arquivos de configuração com as receitas escritas em linguagem declarativa são armazenadas no nó do servidor denominado como Puppet *Master*. Este nó principal é responsável por compilar as receitas e gerar catálogos, que são arquivos XML (*eXtensible Markup Language*) reconhecidos pelos agentes. Os clientes comparam, de tempos em tempos, o catálogo recebido do servidor com catálogo da sua *cache* local, implementando as alterações e reportando-as para o servidor principal.

Essa comunicação entre o servidor principal e os agentes ocorre através do protocolo HTTPS (*Hyper Text Transfer Protocol Secure*) com identificação do cliente, ou seja, tanto o servidor quanto os nós precisam possuir um certificado SSL (*Secure Socket Layer*) válido. Devido a necessidade do certificado, os dispositivos a serem administrados necessitam suportar o Puppet.

2.1.7 SaltStack

O SaltSatack foi projetado para permitir a comunicação de baixa latência e alta velocidade para coleta de dados e execução remota em infraestruturas de rede. Pode ser considerada como uma ferramenta para orquestração orientada a eventos e com um sistema altamente modular. A plataforma é escrita em *Python* e pode usar o modelo *push* ou *pull*, dependendo da necessidade da infraestrutura. No método *push*, os comandos são executados via protocolo SSH. O SaltStack permite a execução paralela de vários comandos criptografados via AES (*Advanced Encryption Standard*) e oferece escalonamento vertical e horizontal.

A ferramenta também baseia-se no modelo cliente-servidor, sendo um o servidor chamado de Salt *Master* e os agentes chamados de Salt *Minion*. O servidor é responsável por controlar os Salt *Minions*, que executam as tarefas e devolvem os dados resultantes ao Salt *Master*. A comunicação é iniciada pelos clientes e é realizada através das mensagens ZeroMQ através de chaves. Os scripts no SaltStack são escritos em YAML e a comunidade de desenvolvedores é bastante ativa no *Github*.

2.2 Conceitos e Componentes do Ansible

Existem dois tipos de máquinas na arquitetura do Ansible: o nó de controle e os *hosts* gerenciados. O *software* Ansible é instalado e executado a partir de um nó de controle, em que estão armazenados todos os seus componentes. Um nó de controle pode ser um computador de um administrador, um sistema compartilhado entre inúmeros administradores ou um servidor rodando Ansible Tower.

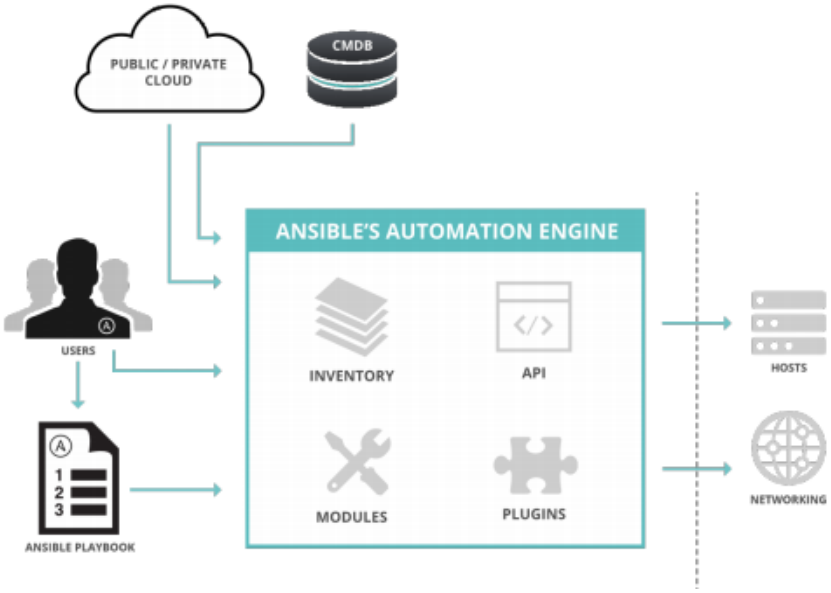
Os *hosts* gerenciados são listados em um inventário, responsável também por organizar esses sistemas em grupos para facilitar o gerenciamento coletivo. O inventário pode ser definido em um arquivo de texto estático, ou dinamicamente determinados por *scripts* que recebem a informação através de fontes externas (CHANG et al., 2016).

Os usuários do Ansible criam ações de alto nível para garantir que um *host* ou grupo de *hosts* esteja em um estado em particular. Uma ação realiza uma série de tarefas no *host* ou nos *hosts* na ordem especificada pela ação. Essas ações são expressas em formato YAML em um arquivo de texto. Um arquivo que contém uma ou mais ações e chamado de *playbook*. Cada tarefa executa um módulo com argumentos específicos. O Ansible tem centenas de módulos úteis que podem realizar uma grande variedade de tarefas de automação. Eles podem agir em arquivos de sistema, instalar software ou fazer chamadas de API.

Quando usado em uma tarefa, um módulo geralmente garante que algo em particular na máquina esteja em um determinado estado. Como por exemplo, de acordo com (PILLA,

2020), uma tarefa que usa um módulo pode garantir que um arquivo exista e tenha permissões e conteúdo particulares, enquanto uma tarefa que usa outro módulo pode garantir que um sistema de arquivos especifico seja montado. Se o sistema não estiver nesse estado, a tarefa deverá colocá-o em tal estado. Se o sistema já estiver nesse estado, a tarefa não fará nenhuma atuação.

Figura 4 – Arquitetura do Ansible



Fonte: retirado de (CHANG et al., 2016)

O Ansible também usa *plugins*. *Plugins* são códigos que podem ser adicionados para estendê-lo e adaptá-lo a novos usos e plataformas. Na Figura 4 está ilustrada a arquitetura de funcionamento do Ansible presente nos nós de controle.

2.2.1 Inventário

Um inventário define uma coleção de *hosts* em que o Ansible faz o gerenciamento. Os inventários de *hosts* são divididos de duas maneiras: um inventário de *host* estático e um inventário de *host* dinâmico. Um inventário estático é definido por um arquivo de texto, enquanto um inventário de *host* dinâmico pode ser gerado por um *script* consumindo as informações a partir de uma fonte externa (CHANG et al., 2016).

Um arquivo de inventário estático é um arquivo de texto INI que especifica os *hosts* alvos do Ansible. O formato de arquivo INI é um padrão informal para arquivos de configuração para algumas plataformas ou *software*. Arquivos INI são arquivos de texto simples com uma estrutura básica composta de “seções” e “propriedades”. Em sua forma

mais simples, como demonstrado na Figura 5, um inventário estático é uma lista de nomes ou de endereços IP dos *hosts* gerenciados, cada um descrito em uma única linha.

Figura 5 – Exemplo de Inventário Estático.

```
web1.example.com
web2.example.com
db1.example.com
db2.example.com
192.0.2.42
```

Fonte: retirado de (CHANG et al., 2016)

Usualmente, a fim de organizar uma infraestrutura ampla, os *hosts* são dispostos em grupos. Os grupos de *hosts* permitem que sejam executados *playbooks* em uma coleção de sistemas. Nesse caso, cada seção começa com um nome de grupo entre colchetes. Além do mesmo *host* poder ser membro de diversos grupos, o arquivo de inventário estático também permite a inclusão de grupos "filhos". Dessa forma, o grupo "filho" é acompanhando com o sufixo ":children". A Figura 6 demonstra um grupo chamado *north-america*, que inclui todos os *hosts* dos grupos *usa* e *canada*.

Figura 6 – Exemplo de Inventário Estático com Grupos "Filhos".

```
[usa]
washington1.example.com
washington2.example.com

[canada]
ontario01.example.com
ontario02.example.com

[north-america:children]
canada
usa
```

Fonte: retirado de (CHANG et al., 2016)

Ainda, como forma de simplificação de sintaxe, o inventário do Ansible suporta *ranges* nos nomes dos *hosts* ou endereços IP. Os *ranges* correspondem a todos os valores do início ao fim, como por exemplo, pode-se declarar os *hosts* como *server[01:20].example.com* (CHANG et al., 2016).

2.2.2 Introdução ao YAML

O YAML é um formato de serialização de dados legível por humanos, que se inspirou em conceitos de linguagens como o XML, C, Python e também no formato do correio eletrônico especificado no RFC 2822. Apesar de contraditório, o acrônimo recursivo YAML significa "YAML não é uma linguagem de marcação" e foi proposto por Clark Evans em 2001 (FONSECA; SIMOES, 2007).

Embora não seja menos genérico que o XML, o YAML é normalmente mais simples de ler, editar, modificar e produzir do que o XML. Ainda de acordo com (FONSECA; SIMOES, 2007), o YAML foi definido para suportar apenas caracteres no sistema UTF8 e UTF16. O YAML foi criado com a crença de que todos os dados podem ser adequadamente representados por combinações de listas, dicionários e escalares. Uma lista é uma coleção ordenada de zero ou mais valores; um dicionário é uma coleção ou mais de pares nome/valor; um escalar representa *strings*, inteiros, datas e outros tipos de dados.

As sequências são listas:

```
- Item 1
- Item 2
```

Os dicionários são pares de valores-chave:

```
- key1: value1
- key2: value2
```

Os escalares podem ser expressos com aspas duplas, aspas simples ou sem aspas:

```
- Isso é uma string
- 'Isso é uma string'
- "Isso é uma string"
```

Os arquivos YAML, opcionalmente, começam com três travessões e terminam com três pontos. Entre os marcadores iniciais e finais do documento, a linguagem YAML usa identação com caracteres de espaço para indicar a estrutura dos seus dados. YAML não impõe requisitos sobre quantos espaços são usados para identação, mas seguem duas regras básicas (EDELMAN, 2016):

- Os elementos de dados na mesma hierarquia (como itens na mesma lista) devem ter a mesma identação.
- Itens que são "filhos" de outro item devem ser mais identados do que seu superior.

Um comentário explícito é marcado por um indicador "#". Os comentários devem ser separados de outros *tokens* por caracteres de espaço em branco.

2.2.3 Anatomia de um *Playbook*

O *playbook* é o objeto de mais alto nível na estrutura do Ansible e pode ser definido como o arquivo que possui as instruções para as configurações dos *hosts*. Um *playbook* é escrito com a linguagem de marcação YAML contendo uma ou mais *plays* e é usado para definir o estado desejado de um sistema.

Como forma de ilustrar os componentes de um *Playbook*, foi retirado um simples exemplo de (CHANG et al., 2016).

Figura 7 – Exemplo de um *Playbook*.

```
- name: Deploy and start Apache HTTPD service
  hosts: webserver
  vars:
    web_pkg: httpd
    firewall_pkg: firewalld
    web_service: httpd
    firewall_service: firewalld
    python_pkg: python-httplib2
    rule: http

  tasks:
    - name: Required packages are installed and up to date
      yum:
        name:
          - "{{ web_pkg  }}"
          - "{{ firewall_pkg }}"
          - "{{ python_pkg }}"
        state: latest

    - name: The {{ firewall_service }} service is started and enabled
      service:
        name: "{{ firewall_service }}"
        enabled: true
        state: started
```

Fonte: retirado de (CHANG et al., 2016)

No exemplo da Figura 7 está demonstrada a anatomia de um *playbook*, contendo *plays*, *tasks*, módulos e variáveis que serão aprofundados no decorrer desta seção. Nesse caso, o Ansible irá instalar nos hosts ou grupos de hosts *webserver* os pacotes descritos em *vars* através do módulo *yum* e habilitar e iniciar o *Firewall* do servidor.

As *plays* consistem em um conjunto ordenado de tarefas a serem executadas em *hosts* selecionados no arquivo de inventário do Ansible. De acordo com (HOCHSTEIN; MOSER, 2017), todas as *plays* devem conter o seguinte:

- Um conjunto de *hosts* para configurar
- Uma lista de tarefas para ser executada nestes *hosts*

Além de especificar os *hosts* e *tasks*, as *plays* também suportam configurações adicionais. As três principais são *name*, *become* e *vars*. Elas estão definidas a seguir:

- *name*: é um comentário que descreve sobre o que refere-se o *play* e é exibida na tela pelo ansible quando uma *play* é iniciada.
- *become*: se for verdadeira, o *Ansible* irá executar executar todas as tarefas por padrão como sendo o usuário administrador.
- *vars*: é uma lista de variáveis e valores a serem utilizadas pelo Ansible.

O atributo das *tasks* faz parte dos *plays* que listam em ordem as tarefas que o Ansible irá realizar e a ordem de execuçao das mesmas. As *tasks* representam o que se automatiza de maneira declarativa sem preocupar-se com a sintaxe ou como é realizada a operação. Cada *task* na lista é uma coleção de pares chave-valor. As *tasks* também suportam a configuração do atributo *name*, assim como as *plays*. Da mesma forma como as *plays*, o texto é arbitrário e demonstrado quando um *playbook* é executado para melhorar a legibilidade e repassar as informações durante a execução de um *playbook*. Abaixo, uma simples *task* retirada de (CHANG et al., 2016).

```
tasks:
   - name: newbie exists with UID 4000
     user:
        name: newbie
        uid: 4000
        state: present
```

O parâmetro *user* determina o módulo para execução da *task*. Os módulos são *scripts* que vem embalados com o Ansible e realizam algum tipo de ação em um *host*. Existe uma grande variedade de módulos para o Ansible, desde equipamentos de rede até integração com uma provedora de serviços na nuvem.

Nos módulos, seus argumentos são passados como uma coleção de pares chave-valor que são atributos "filhos" do módulo. Qualquer linguagem de programação pode ser usada para escrever um módulo do Ansible desde que seja possível retornar um JSON com pares chave-valor. A maioria dos módulos desenvolvidos para o Ansible são escritos em *Python*.

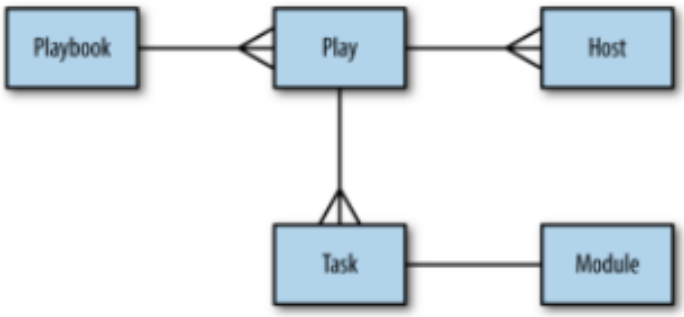
Os módulos suportam a inserção de atributos dinâmicos através das variáveis do Ansible. As variáveis possuem nomes que devem começar com uma letra e podem conter somente letras, números e *underlines*. Ao escrever um *playbook*, os administradores podem usar suas próprias variáveis e chamá-las em uma tarefa. As variáveis dentro de um *playbook*

são apresentadas entre chaves duplas. As variáveis podem ser simplificadas em três níveis de escopo.

- Escopo Global: variáveis estabelecidas desde a linha de comando ou configuração do Ansible.
- Escopo de *Playbooks*: variáveis estabelecidas nos *plays* e estruturas relacionadas.
- Escopo de *Hosts*: variáveis estabelecidas dentro dos *host_vars* e *hosts* individuais pelo inventário.

A Figura 8 descreve a relação que existe entre cada um dos componentes de um *playbook* respeitando a hierarquia que existe entre os mesmos.

Figura 8 – Anatomia de um *Playbook*



Fonte: retirado de (HOCHSTEIN; MOSER, 2017)

2.3 Arquitetura de Rede *Market Data*

O termo *Market Data* é conhecido no mercado de capitais para referenciar-se aos dados de negociação de mercado. O *Market Data* abrange uma gama de informações, como preço, cotações de compra/venda e volume de mercado e está disponível em milhares de mercados globais, incluindo ações, câmbio e *commodities*. Na Bolsa de Valores Brasileira (B3), o *Market Data* são mensagens estruturadas pelo protocolo FIX (*Financial Information eXchange*). Uma mensagem FIX é uma *string* ASCII constituída de campos *tag*=valor separados pelo caractere SOH (*Start of Header*). As mensagens possuem um cabeçalho, um corpo, e um rodapé. No exemplo de mensagem FIX abaixo, o caractere SOH foi substituído por "||".

```
|1128=9|35=X|34=656307|52=20140610-13:26:15.156|75=20140610|
```

```
|268=1|279=2|269=0|22=8|207=BVMF|48=3809779|83=20178|
|273=61:51:55|272=20140610|37=80531933204|288=8|290=7|
```

Por ser um volume elevado de dados, é necessária uma compressão do *Market Data*. Existem múltiplos protocolos utilizados para a compressão de *Market Data* nas diversas Bolsas de Valores globais. O protocolo mais utilizado para a compressão de *Market Data* é o protocolo FAST (*FIX Adapted for STreaming*). O protocolo FAST, desenvolvido pelo *FIX Market Data Optimization Working Group*, é um algoritmo para compressão de dados que reduz significativamente os requisitos de largura de banda e latência entre remetente e destinatário (B3, 2019).

O serviço de distribuição de *Market Data* é um típico exemplo de uma aplicação que necessita entregar o mesmo *stream* de dados para centenas e potencialmente milhares de usuários finais. Inicialmente, os serviços de *Market Data* foram implementados com TCP ou UDP via *broadcast* na camada de rede, porém ambas implementações possuem escalabilidade limitada (CISCO, 2008). Com o protocolo TCP, é requerido um *socket* individual com janela deslizante no servidor para cada destinatário, enquanto o UDP via *broadcast* precisa enviar uma cópia do *stream* para todas as sub-redes de destino.

Ambos os métodos citados acima iriam exaurir os recursos dos servidores e também da rede. No lado da rede, a largura de banda exigida para a aplicação teria um crescimento de forma linear. Por exemplo, para enviar um *stream* de 1 *Mbps* para 1000 destinatários utilizando TCP seria necessário uma largura de banda de 1 *Gbps*. Já com relação a UDP via *broadcast*, apesar de resolver a necessidade de múltiplos *streams*, todos os equipamentos da rede precisariam lidar com os pacotes, estando ou não interessados nos dados.

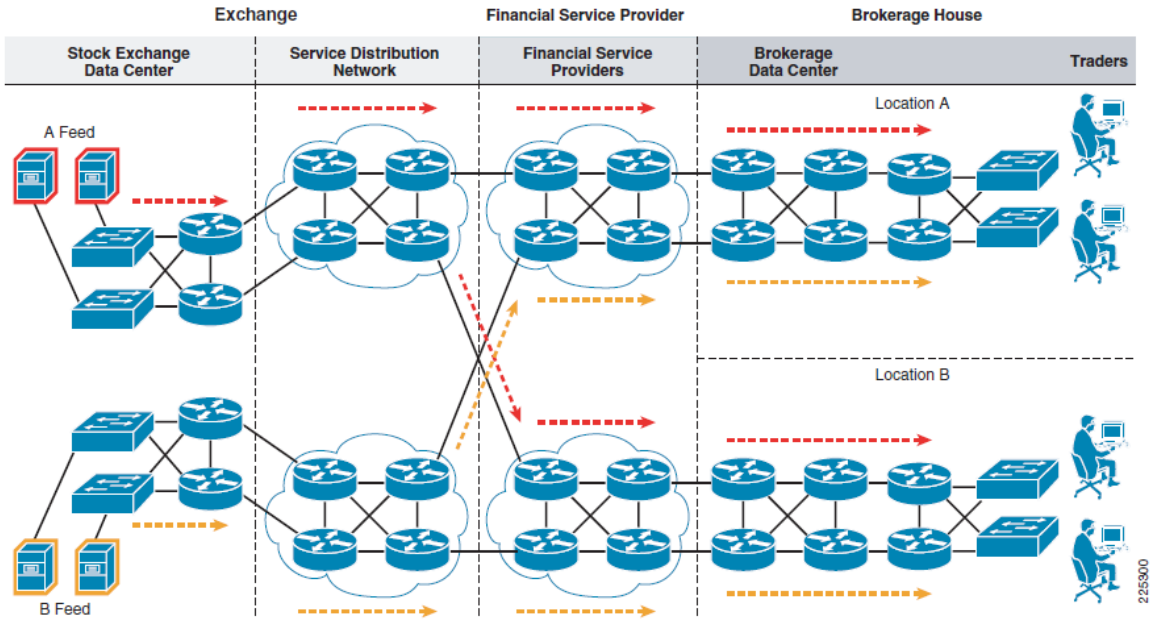
Logo, *IP multicast* é a única maneira de escalar a entrega de *Market Data*. Para enviar um *stream* de 1 *Mbps* para 1000 destinatários, com o *multicast*, a largura de banda necessária é 1 *Mbps*. Além disso, se apenas poucos servidores estiverem interessados em receber determinado canal *multicast*, o *stream* será enviado apenas para os mesmos.

Conforme descrito por (CISCO, 2008), o escopo de uma arquitetura de rede *Market Data* inclui: as fontes para transmissão de *Market Data* (Bolsa de Valores), o provedor de serviços financeiros e os consumidores finais dos dados (Corretoras). Os componentes da arquitetura podem ser divididos em três itens principais:

- A rede da Bolsa de Valores
- O provedor de serviços financeiros
- A rede da corretora

Uma ilustração dos componentes da arquitetura de rede *Market Data* está exibido na Figura 9.

Figura 9 – Componentes da Arquitetura de Rede *Market Data*



Fonte: retirado de (CISCO, 2008)

Uma arquitetura de rede *Market Data* deve ser resiliente, redundante e eficiente para não perder um único pacote em caso de falha em um roteador (CISCO, 2008). Ainda, o *stream* deve possuir a menor latência possível, visto que os dados de *Market Data* são o suporte para transações em tempo real. Portanto, cada Bolsa de Valores produz dois canais de dados redundantes chamados de *Feed A* e *Feed B*. Cada um desses *Feeds* possui o mesmo dado, porém são transmitidos usando diferentes endereços de grupos *multicast* e portas.

2.3.1 O Protocolo PIM-SM

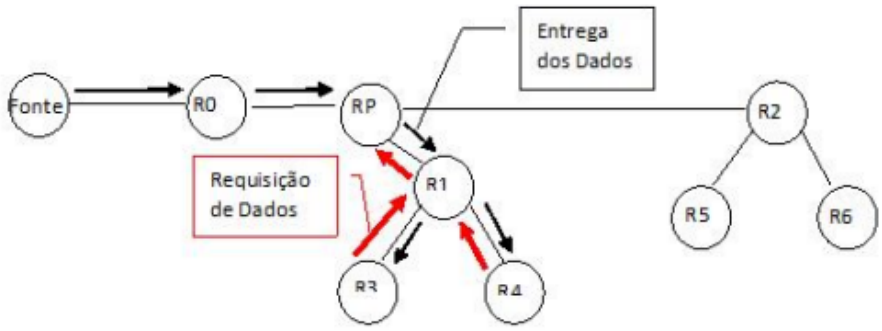
O PIM *Protocol Independent Multicast* é o protocolo de roteamento *multicast* mais usualmente encontrado. Conforme descrito por (MORAES; DUARTE, 2004), existem três modos de operação deste protocolo: o denso (PIM-DM), o esparso (PIM-SM) e o SSM (*Source Specific Multicast*). Os dois primeiros modos estão associados distribuição dos receptores na rede. Um grupo é considerado denso quando a probabilidade de uma determinada área ter pelo menos um membro do grupo é alta. Do contrário, se a probabilidade for pequena, o grupo é considerado esparso. O terceiro modo implementa o serviço SSM.

O *PIM Sparse Mode* foi projetado para o roteamento em grande escala, em que a distribuição dos receptores é esparsa. O PIM-SM constrói árvores compartilhadas unidirecionais a partir de mensagens de inscrição (*join*) em um grupo *multicast* enviadas ao

nó de "ponto de encontro" conhecido por RP (*Redzvous Point*). As fontes também enviam seus dados para o RP, que por sua vez distribui a informação na árvore *multicast*. Para fontes com alta taxa de transmissão, é possível mudar o tipo de árvore de distribuição, ou seja, usar uma árvore por fonte. Para tanto, basta que as mensagens de inscrição passem a ser enviadas diretamente para a fonte.

De acordo com (FRANCA; JUNIOR; BRITTO, 2012), os *hosts* que desejam fazer parte de um grupo *multicast* mandam mensagens explícitas de associação aos roteadores que estão no caminho do RP escolhido. Para isso, ele cria um cartão com informações para a entrada *multicast*, denominada entrada (,G), por onde serão enviados os pacotes através da Árvore RTP exibida na Figura 10.

Figura 10 – Árvore RTP *Rendzvous Point Tree*



Fonte: retirado de (CABRAL; KRONBAUER; MARTINS, 1998)

É de responsabilidade dos *hosts* informarem aos roteadores sobre a sua participação em grupos *multicast*. O protocolo padrão para realizar esta tarefa é o IGMP (*Internet Group Management Protocol*), que é executado no roteador responsável pela gerência dos grupos *multicast*, o DR (*Designated Router*).

Quando não existirem mais membros locais conectados ao grupo, o DR fica sabendo através do IGMP (*Internet Group Management Protocol*) (CABRAL; KRONBAUER; MARTINS, 1998). Se não houver *downstream* necessitando de pacotes do par (Grupo, Fonte), a entrada (,G) é removida. Quando um *host* inicia a transmissão de pacotes de dados *multicast* para um grupo, inicialmente, seu DR entrega cada pacote ao RP, encapsulando-os em pacotes *unicast*. O RP desencapsula os pacotes e os envia para os membros do grupo pela árvore RPT.

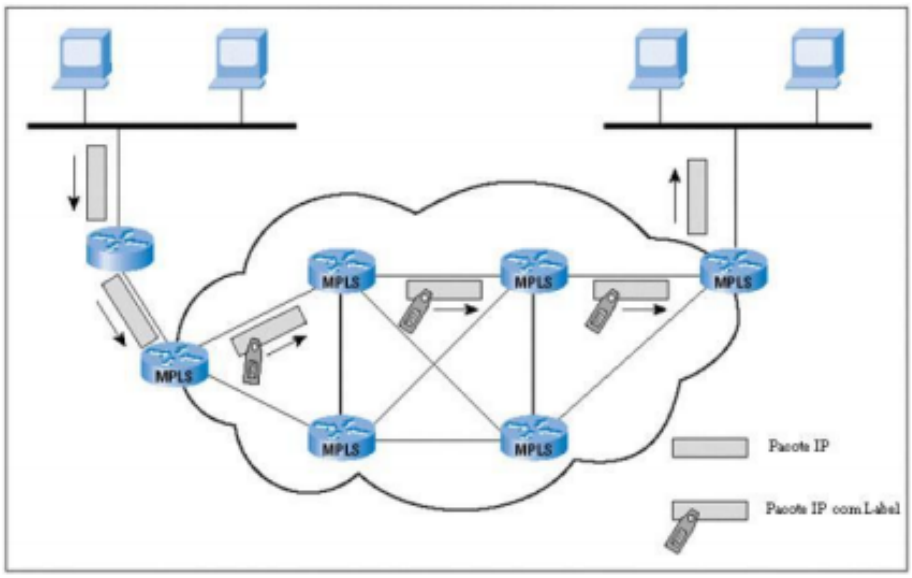
2.3.2 Redes MPLS

A solução MPLS/BGP VPN consiste em um conjunto de métodos utilizados por um provedor de serviços para oferecer VPNs (*Virtual Private Networks*) IP para seus

clientes. O protocolo MPLS (*Multiprotocol Label Switching*) é utilizado para realizar o transporte dos pacotes no *backbone* do provedor e o BGP (*Border Gateway Protocol*) é utilizado para distribuir as rotas das VPNs nesse *backbone* (WESTPHAL, 2011).

O MPLS é um protocolo de transporte que faz a comutação de pacotes rotulados (*labels*). Um label é um valor identificar de 20 bits que é adicionado no *header* de um pacote IP. Os *labels* são inseridos quando os pacotes entram na rede MPLS e, dessa forma, os roteadores encaminham os pacotes apenas verificando o *label* e não mais o endereço IP, tornando o processo mais eficiente. Cada vez que o pacote é repassado o rótulo é substituído até que o pacote chegue ao roteador de saída da rede MPLS. A Figura 11 ilustra o funcionamento básico do MPLS.

Figura 11 – Funcionamento básico do MPLS



Fonte: retirado de (LINHARES, 2010)

O LSP (*Label Switch Path*) é definido como o caminho por onde os pacotes irão trafegar em uma rede MPLS. No momento em que um pacote entra numa rede MPLS, este é associado a uma FEC (classe de equivalência) e assim é criado um LSP relacionado a esta FEC (AIRES, 2004). Como a criação de uma LSP só ocorre na entrada de uma rede MPLS, os roteadores LSR (*Label Switch Router*) do núcleo da rede só terão o trabalho de fazer as trocas dos *labels*, encaminhando assim o pacote de acordo com o LSP determinado anteriormente, não havendo mais necessidade de fazer o roteamento dos pacotes. O protocolo responsável por essa distribuição de *labels* é o LDP *(Label Distribution Protocol)* e, de acordo com (LINHARES, 2010), possui quatro funções principais:

- A descoberta dos LSRs que estão executando o LDP

- O estabelecimento e a manutenção de sessões
- O anúncio de mapeamento de *labels*
- A manutenção de sessões LDP por meio de notificação

O MPLS permite a criação de VPN porque garante um isolamento completo do tráfego com a criação de tabelas de *labels* usadas para roteamento exclusivas de cada circuito virtual. Uma VPN MPLS é implementada sobre duas redes: a rede do provedor e a rede do cliente. A rede do cliente possui, ao menos, um roteador CE *(Customer Edge)* que se conecta diretamente com roteadores de borda da rede do provedor, os PEs *(Provider Edge)*. Os roteadores do *backbone* que não se conectam com nenhum CE são chamados de roteadores P (*Provider*). A Figura 12 demonstra os componentes.

Figura 12 – Terminologia da MPLS VPN

Fonte: retirado de (WESTPHAL, 2011)

Para realizar o isolamento entre os clientes, considerando que a solução MPLS VPN não requer um PE dedicado para cada ponto de acesso, é introduzido o conceito de VRF (*Virtual Routing and Fowarding*). Uma VRF é uma tecnologia que virtualiza uma tabela de roteamento, deixando-a separada da tabela de roteamento global. Dessa forma, utilizando múltiplas VRFs, um mesmo PE pode ser compartilhado por vários clientes (WESTPHAL, 2011).

A fim de possibilitar a comunicação entre PEs distintos, as VRFs de cada PE devem ser populadas com as rotas dos seus sites locais e anunciar os seus prefixos conhecidos para os outros PEs do *backbone*. Para isso, utiliza-se um único protocolo de roteamento que que rode diretamente entre os PEs, como por exemplo, o protocolo iBGP (*internal BGP*).

Para que os exista conexão dentro do *backbone*, é necessário um protocolo de roteamento interno, como por exemplo, o OSPF (*Open Shortest Path First*). O protocolo de roteamento OSPF, definido no RFC 2328, é um protocolo de roteamento baseado no estado de enlace (PEREIRA, 2004). O OSPF permite a divisão de uma rede em áreas e

torna possível o roteamento dentro de cada área e entre as diferentes áreas, usando os chamados roteadores de borda. Com isso, usando o OSPF, é possível criar redes hierárquicas de grande porte, sem que seja necessário que cada roteador tenha uma tabela de roteamento gigantesca, com rotas para todas as redes, como seria necessário no caso do RIP. O OSPF é projetado para intercambiar informações de roteamento em uma interconexão de rede de tamanho grande ou muito grande, como por exemplo a Internet.

2.3.3 *Multicast* VPN

O recurso MVPN (*Multicast VPN*) fornece a capacidade de suportar *multicast* em uma VPN de camada 3. À medida que as empresas estendem o alcance de seus aplicativos *multicast*, os provedores de serviços podem acomodar essas empresas em sua rede principal de MPLS. Anteriormente, os túneis ponto a ponto eram a única maneira de passar o tráfego *multicast* por meio de uma VPN, o que gerava diversos problemas de escalabilidade (CISCO, 2020).

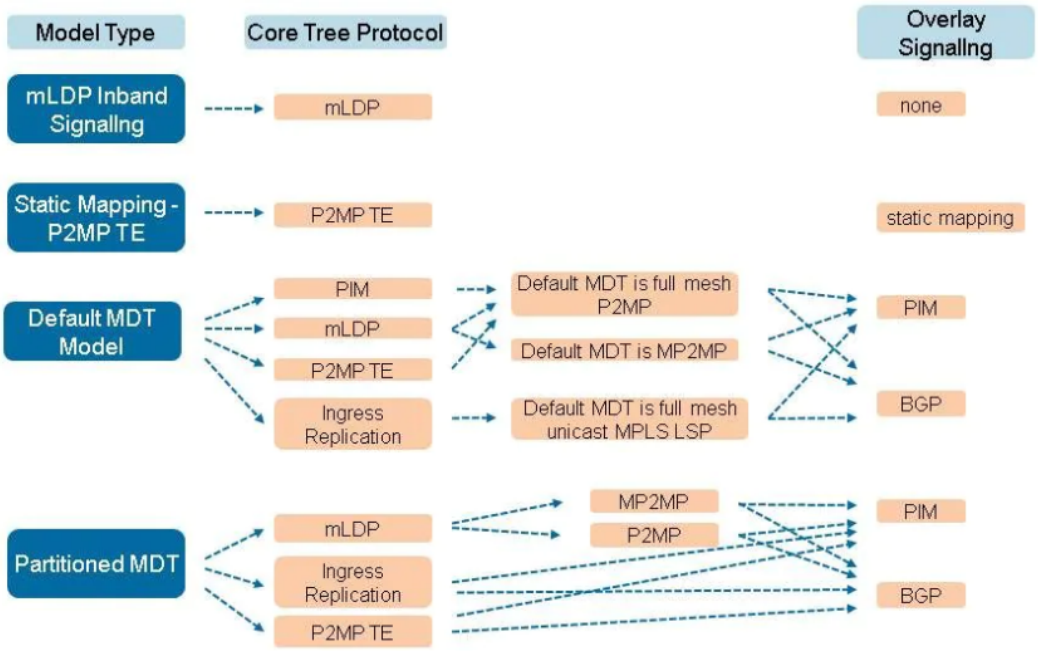
A MVPN é o mecanismo para transportar *multicast* dentro do *backbone* da operadora de serviços que elimina a necessidade da configuração do PIM dentro do *Core MPLS*. Dessa forma, a MVPN permite que uma empresa conecte de forma transparente a sua rede privada em um *backbone*, formando uma adjacência PIM entre seus roteadores.

A Figura 13 mostra as diferentes maneiras de implementar u*multicast* sobre MPLS. Há quatro tipos de protocolos *core tree*: PIM, mLDP, P2MP TE e *Ingress Replication*.

A MVPN implementada através do mLDP (*multicast label distribution protocol*) fornece uma extensão para o LDP para a configuração de P2MP (ponto-para-multiponto) e MP2MP (multiponto-para-multiponto) LSPs (CISCO, 2018). A implementação mLDP estabelece uma árvore de distribuição *multicast* (MDT) para cada domínio *multicast* na rede. A *MDT default* define a *path* usada pelo roteador PE para enviar dados *multicast* e mensagens de controle para todos os demais PEs que estão no mesmo domínio *multicast*. Uma *MDT default* é criada na rede *Core* usando um único MP2MP LSP. Uma *MDT default* comporta-se como uma LAN virtual, conforme demonstrado na Figura 14.

A mLDP-MVPN também suporta a criação dinâmica de *MDT datas* para transmissões com alto tráfego. Para fontes com alto *data rates*, uma *MDT data* é criada usando P2MP LSPs para reduzir o tráfego da *MDT default* a fim de evitar desperdício desnecessário de largura de banda em roteadores PEs que não deram um *join* no fluxo *multicast*. A criação da *MDT data* é sinalizada dinamicamente usando mensagens *MDT Join TLV*. As *MDT datas* são um recurso exclusivo do *software* Cisco IOS. Quando uma transmissão *multicast* excede um determinado limite configurado, é criada uma *MDT data*. Neste momento, é enviada uma mensagem UDP que contém a informação sobre a *MDT data*

Figura 13 – Visão Geral dos Perfis MVPN



Fonte: retirado de (CISCO, 2020)

para todos os dispositivos que encontra-se na *MDT default*. A ilustração da *MDT data* está na Figura 15.

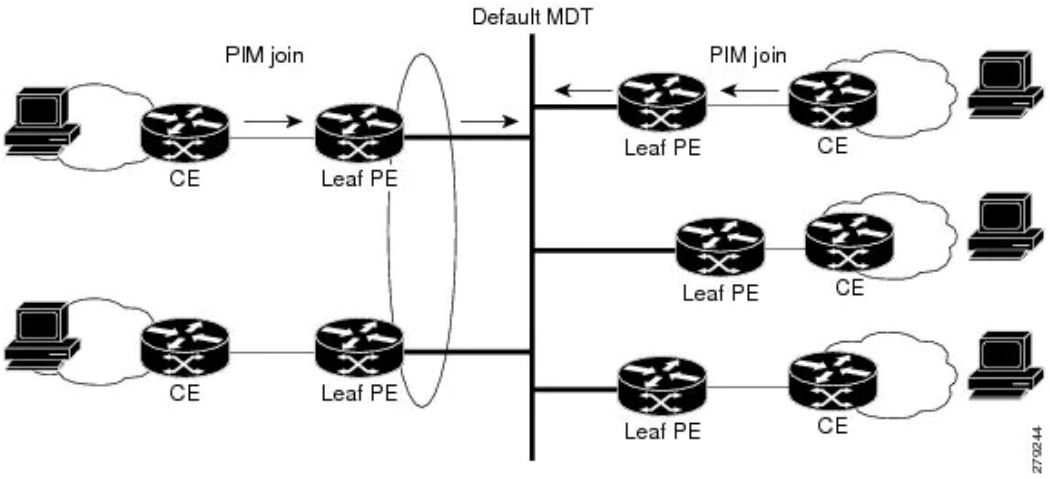
As *MDTs data* são criadas somente para a rotas *multicast* do tipo (S,G). Elas não são criadas para entradas (*,G) porque dependem da taxa de transmissão de dados da fonte. O mLDP cria as ávores *multicast* da seguinte forma:

- A *MDT default* usa MP2MP LSPs.
- Suporta baixa largura de banda e tráfego de controle entre as VRFs
- O *MDT data* usa P2MP LSPs.
- Suporta um único *stream* com uma alta largura de banda em cada VRF

Todas as outas operações da MVPN permanecem as mesmas, independente do mecanismo de tunelamento:

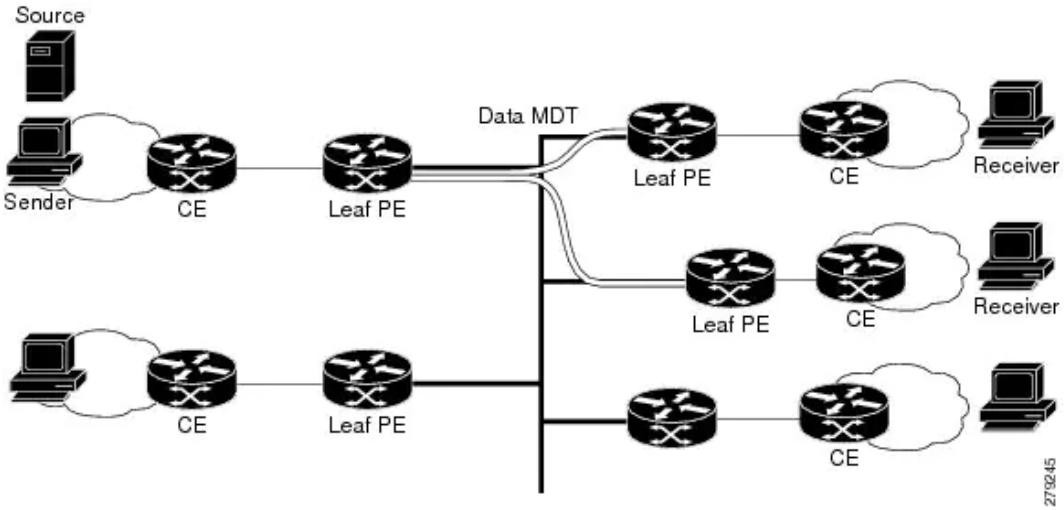
- Os vizinhos PIM em uma VRF são vistos em uma interface virtual *Label Switched Path - Lspvif*
- O estado de *multicast VPN* é sinalizado pelo PIM.

Figura 14 – mLDP com cenário *MDT default*



Fonte: retirado de (CISCO, 2018)

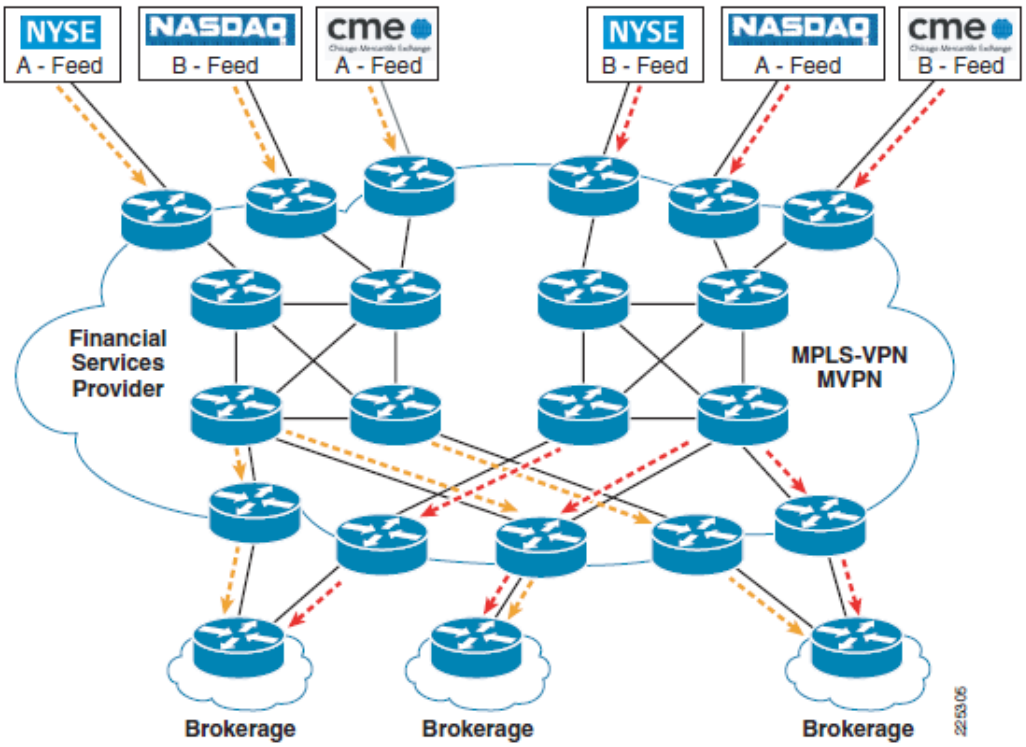
Figura 15 – mLDP com cenário *MDT data*



Fonte: retirado de (CISCO, 2018)

A rede *Market Data* é um caso de uso para uma MVPN. Nos Estados Unidos, as empresas provedoras de serviços financeiros, por exemplo *Reuters* e *Bloomberg*, fazem a conexão entre as Bolsas de Valores e as Corretoras. Para uma solução integrada de *unicast* e *multicast* são utilizados MPLS-VPN e MVPN. Na Figura 16, pode-se analisar a separação física dos *Feeds* redundantes da mesma Bolsa de Valores, a fim de garantir a alta disponibilidade.

Figura 16 – Exemplo de uma aquitetura de rede *Market Data* nos Estados Unidos.



Fonte: retirado de (CISCO, 2008)

Nesse sentido, como o *Market Data* é um *stream* com um alto volume de tráfego, é interessante para o desempenho a criação de uma *MDT data*.

3 MATERIAIS E MÉTODOS

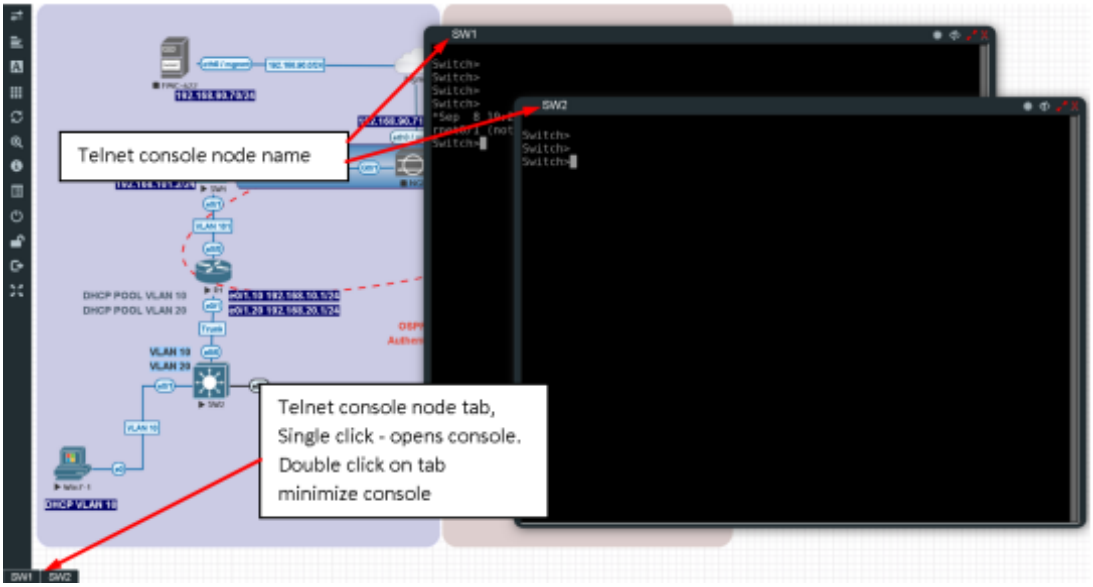
Neste capítulo serão descritos os materiais utilizados e métodos empregados nas etapas que guiaram o desenvolvimento da solução proposta.

3.1 Materiais

3.1.1 Emulador EVE-NG

Um emulador é uma ferramenta que reproduz uma plataforma virtualizada que permite que uma dada arquitetura de computador consiga executar sistemas que foram desenvolvidos para outra arquitetura específica (SOUSA et al., 2016). Dessa forma, o EVE-NG é um emulador de dispositivos de rede que permite a construção de topologias de rede em um ambiente virtualizado, compatível com diversos sistemas operacionais. Por ser um *software* emulador, os equipamentos de rede inseridos no EVE-NG possuem uma grande similaridade comparando-os com equipamentos reais, o que leva a uma redução de custos na montagem de um laboratório. Além disso, a plataforma também suporta Dynamips, funcionalidade que permite emular um roteador e executar imagens com o sistema operacional IOS da Cisco (DZERKALS, 2020).

Figura 17 – Demonstrativo da Interface EVE-NG.



Fonte: retirado de (DZERKALS, 2020)

A interface gráfica do EVE-NG, na versão 2.0.3, apresentada na Figura 17, pode ser acessada através de um navegador. Além disso, o EVE-NG provê uma solução de console HTML5, ou seja, a gerência dos dispositivos podem ser acessada diretamente via navegador utilizando a *engine* do Apache Guacamole. Esse método também pode ser integrado à ferramenta Wireshark, sendo extremamente útil para fazer as validações da topologia e eventual *troubleshooting*.

Conforme (SOUSA et al., 2016), a limitação do EVE-NG fica por conta do desempenho do elemento emulado, notadamente inferior ao de um elemento físico. Por este motivo, é desaconselhável a aplicação de elementos emulados em testes de desempenho. Como o presente trabalho possui foco na implementação de configurações e não no desempenho da rede, entendeu-se que o EVE-NG supre as necessidades requeridas.

3.1.2 *Google Cloud Platform*

Para a instalação do EVE-NG, de acordo com o manual do fabricante, os requisitos recomendados de *hardware* estão listados na Tabela 1.

Tabela 1 – Requisitos de sistema recomendados para instalação do EVE-NG

Requisitos de *hardware* para PC	
CPU	Intel I7 (8 núcleos)
RAM	32 GB
HDD Space	200GB
Rede	LAN/WLAN

Devido ao *hardware* disponível não ser suficiente para atender aos requisitos exigidos pelo fabricante, buscou-se outras alternativas para tornar possível a instalação do *software*. Logo, a solução mais prática encontrada foi o uso da virtualização de uma máquina através de uma plataforma provedora de serviços na nuvem.

Depois da análise de algumas provedoras de serviços na nuvem, a escolhida para a máquina virtual foi a *Google Cloud Platform* (GCP) por possuir um passo a passo de instalação no manual do EVE-NG e U$ 300 de crédito gratuito para novos usuários.

Uma máquina virtual provisionada na nuvem pode ser chamada de instância. Nas instâncias do *Google Compute Engine*, é possível executar tanto as imagens públicas para Linux e Windows Server fornecidas pela Google quanto as imagens privadas personalizadas pelo próprio usuário. Além disso, a *Google Cloud Platform* fornece instâncias do tipo preemptiva, que permite a criação e execução de instâncias a um preço muito mais baixo se comparado com instâncias normais. No entanto, essas instâncias são sempre interrompidas após 24 horas ou quando o *Google Compute Engine* necessita usar os recursos em outras tarefas.

3.1.3 *VLC Media Player*

As ferramentas para *stream* de *Market Data* são bastante restritas e dificilmente disponibilizadas para domínio público. Sendo assim, como o foco do trabalho está nas configurações dos equipamentos de rede, buscou-se uma ferramenta capaz de realizar uma transmissão de dados *multicast* via UDP e, no lado do receptor, enviar uma mensagem de *join* para o RP assinando o canal *multicast* de interesse. Para o presente trabalho, utilizou-se o VLC na versão 3.0.14.

Logo, para fins de teste de funcionamento da topologia de rede implementada, optou-se por utilizar uma transmissão de vídeo por ser mais acessível e, sob a óptica do modelo TCP/IP, uma semelhança com a transmissão de *Market Data* até a camada de transporte.

O *VLC Media Player*, de acordo com o *site* oficial, é um reprodutor multimídia livre e de código aberto que possui diversos protocolos de transmissão de rede. Para configurar uma transmissão UDP no VLC basta abrir a aba de *stream*, selecionar o arquivo que deseja-se transmitir, configurar o grupo *multicast* e a porta e o formato do vídeo. Já no receptor, conforme demonstrado na Figura 18, configura-se na URL de rede *udp://@canal:porta*.

3.1.4 *Visual Studio Code*

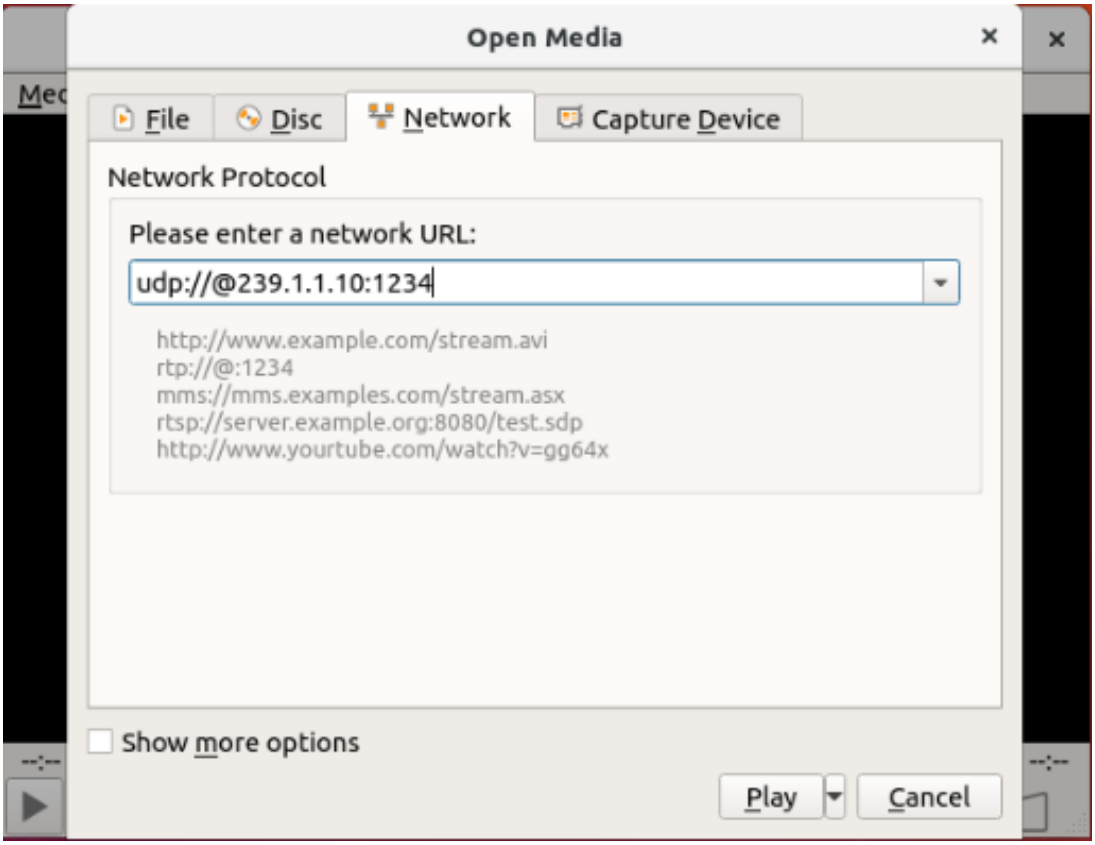
Os *playbooks* para automação das configurações de rede e os arquivos de inventário do Ansible foram desenvolvidos no editor de código fonte *Visual Studio Code*. Ainda, o *Visual Studio Code* possui extensão para realizar conexões SSH. Portanto, os comandos para execução dos *playbooks* também foram executados via terminal a partir do editor conectando remotamente no servidor do Ansible. A versão utilizada do *Virtual Studio Code* foi a 1.56.

O editor é desenvolvido pela Microsoft e possui versões para Windows, Linux e macOS. Ele inclui suporte para depuração, controle de versão incorporado, realce de sintaxe e complementação inteligente de código. Trata-se de um software livre e de código aberto, apesar do *download* oficial estar sob uma licença proprietária. A Figura 19 mostra sua interface com parte do *playbook* para configuração dos roteadores PEs em linguagem *YAML*.

3.1.5 *Github*

GitHub é uma plataforma de hospedagem de código-fonte com controle de versão, que permite a interação e a contribuição de múltiplos usuários cadastrados em projetos privados ou de código aberto. A escolha por utilizar esta plataforma vem da necessidade de um controle de versão dos *playbooks* do Ansible e das configurações dos roteadores.

Figura 18 – Configurando recepção de vídeo *multicast* no VLC.



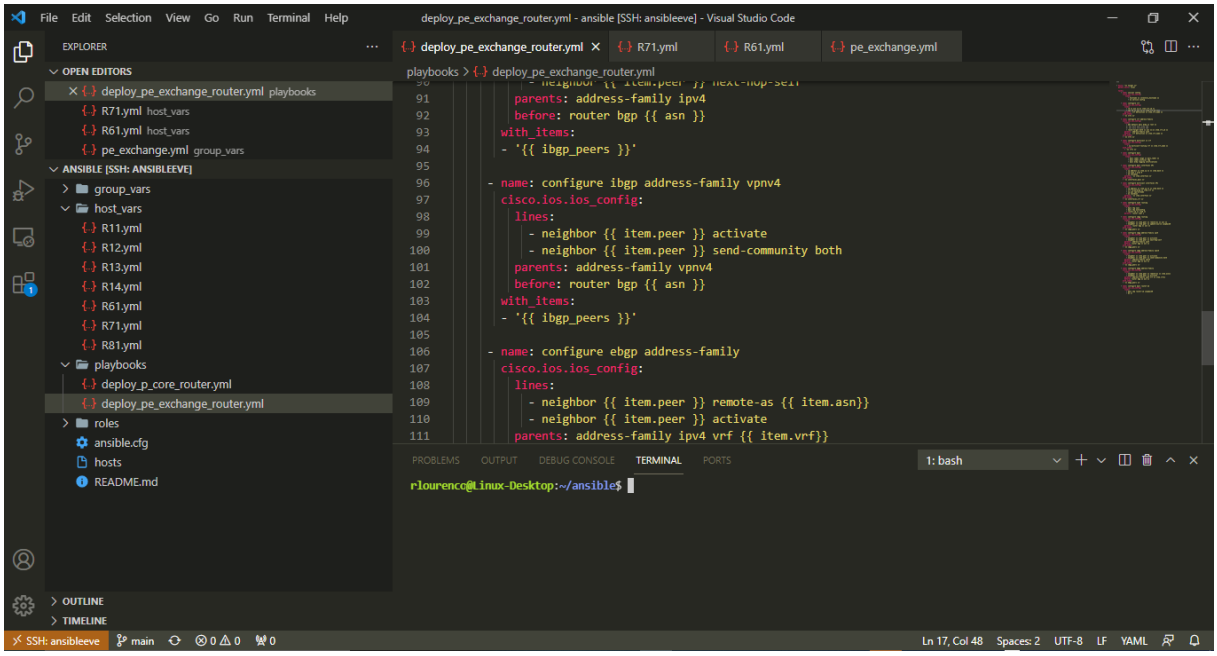
Fonte: O autor

As etapas para o desenvolvimento dos códigos deste trabalho podem ser verificadas na aba *Commits* do projeto *ansible*, o qual está configurado como privado e ligado à conta *rafaelllourenco*, como pode ser visto na Figura 20. As alterações dos arquivos são inseridas com suas respectivas datas e código de identificação (*commit hash*). Através desse número é possível retornar a estados anteriores do projeto, em que constam preservados todos os arquivos existentes na época.

3.1.6 *Wireshark*

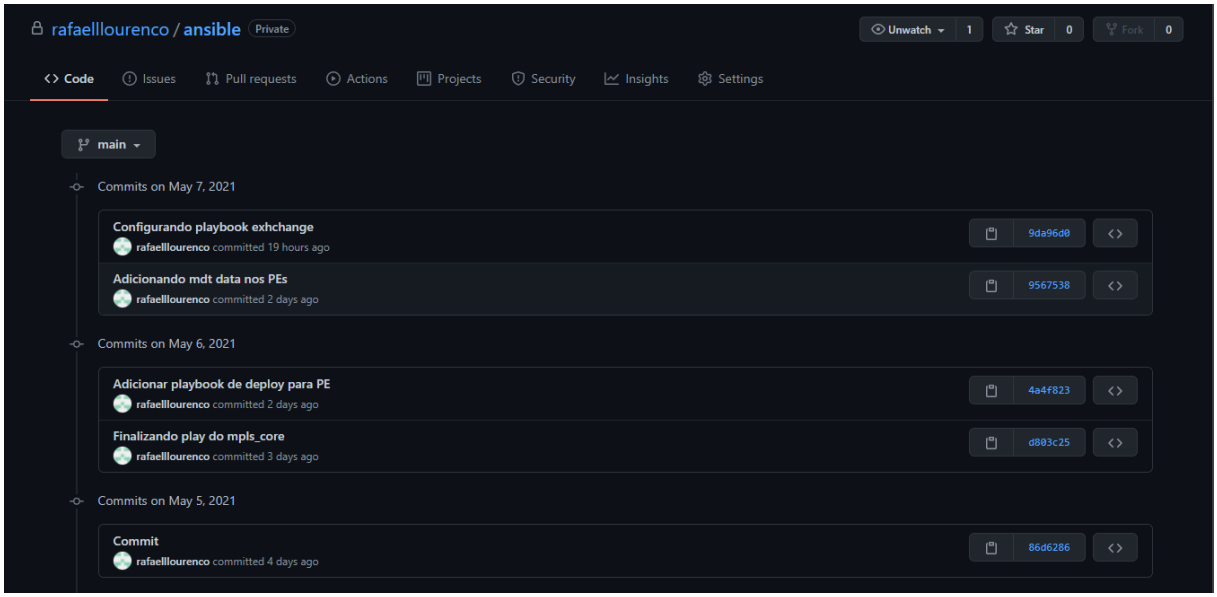
O *Wireshark* é uma ferramenta de captura de tráfego de rede e permite visualizar das mais variadas formas os pacotes que passam por uma determinada interface (FARRUCA, 2009).Devido a esta funcionalidade, ele é muito utilizado como visualizador e *debugger* de rede, permitindo verificar quais os pacotes recebidos, enviados, quais os protocolos usados, etc. É utilizado através de uma interface gráfica de modo a permitir ao utilizador visualizar e selecionar o tráfego que deseja. Trata-se de uma aplicação open source e permite que seja alterada ou expandida, por exemplo, através da introdução de novos filtros de pacotes, ou do reconhecimento de novos protocolos.

Figura 19 – *Visual Studio Code* exibindo o *playbook* dos PEs e o terminal



Fonte: O autor

Figura 20 – Atualizações dos *playbooks* do projeto *ansible* via plataforma *Github*



Fonte: O autor

No presente trabalho foi utilizado o *Wireshark* com a versão 3.0.6, a partir da console nativa do EVE-NG. De acordo com (FARRUCA, 2009), o *Wireshark* permite não só fazer a análise do tráfego capturado, mas também a análise e visualização de ficheiros

que contenham tráfego capturado e guardado num dos vários formatos reconhecidos pela aplicação. Os formatos mais comuns são suportados, tais como o do TCPDUMP. Esta ferramenta pode também guardar em ficheiro o tráfego capturado.

3.2 Métodos

Esta seção trata dos métodos adotados para o desenvolvimento deste trabalho, os quais utilizaram os materiais descritos na seção 3.1 e possibilitaram a obtenção dos resultados descritos no capítulo 4.

3.2.1 Preparação do ambiente de emulação

Buscando emular os roteadores e simular a transmissão de dados *multicast*, realizou-se primeiramente a instalação do *software* EVE-NG. Para a execução das etapas de configuração e instalação utilizou-se a documentação disponível em (DZERKALS, 2020) a partir da página 43 na seção "3.4 *Google Cloud Platform*". A seguir será descrito o procedimento de instalação praticado de forma mais compactada.

O primeiro passo é acessar a plataforma da *Google Cloud* com uma conta *Google* e criar um novo projeto. Depois, através do *Cloud Shell* (um ambiente para gerenciar os recursos da *Google Cloud* a partir de um *shell* para Linux) foi executado o comando listado abaixo para criação de uma imagem de Ubuntu 16.04 com virtualização aninhada, ou seja, permitindo a virtualização de dispositivos em uma máquina virtual.

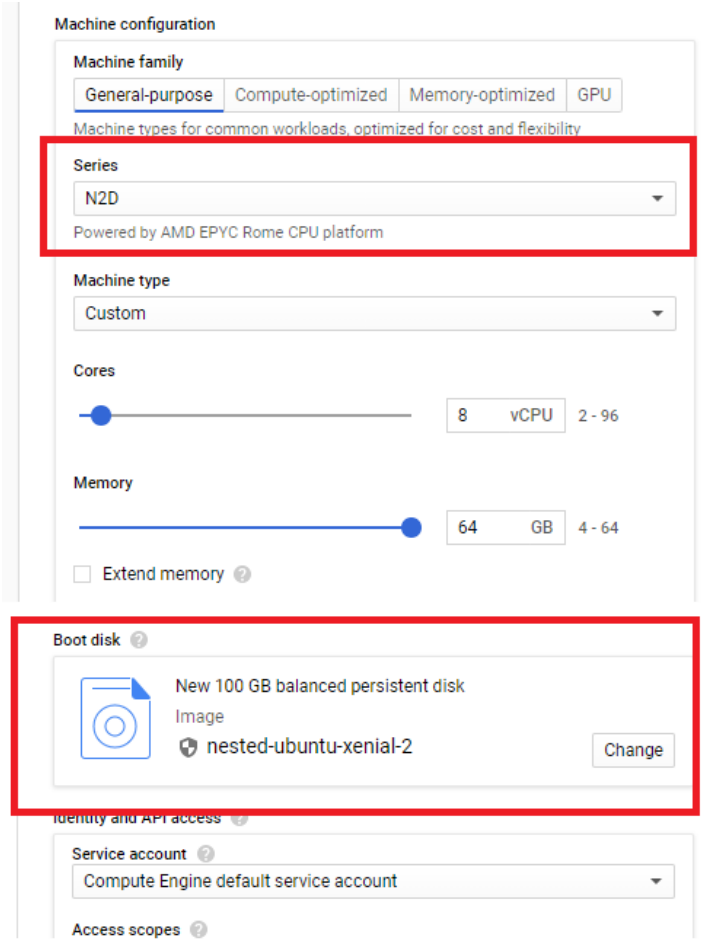
```
gcloud compute images create nested-ubuntu-xenial --source-imageproject=
    ubuntu-os-cloud --source-image-family=ubuntu-1604-lts --licenses="https
    ://www.google.com/compute/v1/projects/vmoptions/global/licenses/enable-
    vmx"
```

Com a imagem disponível na plataforma, partiu-se para a criação da máquina virtual. Conforme demonstrado na Figura 21, o processador selecionado foi o *Intel Cascade Lake* com 8vCPU e 64GB de RAM. A imagem previamente criada é utilizada para no disco *boot* do sistema.

Com a instância iniciada, realizou-se uma conexão SSH a partir do portal da *Google Cloud* e executou-se o comando abaixo no terminal para instalar o EVE-NG.

```
wget -O - http://www.eve-ng.net/repo/install-eve.sh | bash -i
```

Figura 21 – Especificações da máquina virtual criada na plataforma *Google Cloud Platform*



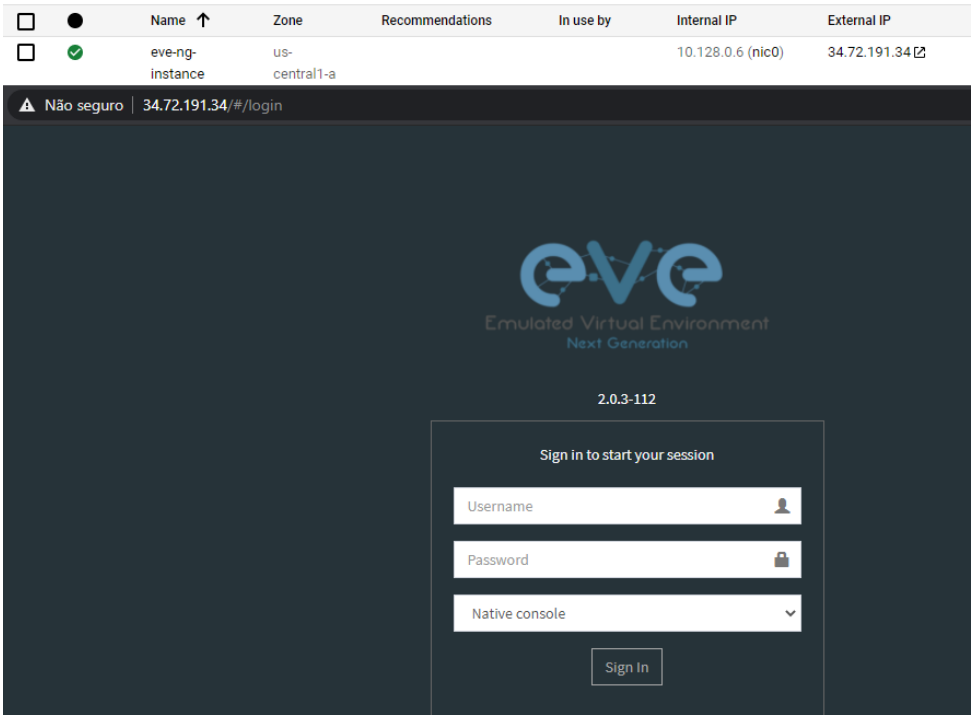
Fonte: O autor

Depois de reiniciar a máquina virtual, acessou-se o endereço http://34.72.191.34, que é o IP do servidor EVE-NG, no navegador local e a tela de *login* do EVE-NG estava disponível conforme demonstrado na Figura 22.

Ao realizar o *login* com o usuário padrão admin, foi criado o novo laboratório *TCC_Rafael_Lourenco.unl* onde foram desenvolvidos a topologia e configurações de rede demonstrados nas próximas etapas do presente trabalho.

Posteriormente, foram adicionas as imagens que serão utilizadas para emulação no servidor do EVE-NG. As imagens de *Cisco IOL*, que também é chamado de *IOS On Linux*, refere-se a versão de IOS sobre Linux compilado para a a arquitetura i386. Além das imagens de roteador e *switch*, foi adicionado uma imagem de *Ubuntu 18.04* disponível no site da EVE-NG. A máquina virtual Linux será utilizada para a instalação do *VLC Media Player* para validar a transmissão e o recebimento de dados *multicast*. Após a adição da imagem qemu no diretório do servidor do EVE-NG */opt/unetlab/addons/qemu* e a

Figura 22 – Instância na *Google Cloud Platform* e tela de *login* do EVE-NG



Fonte: O autor

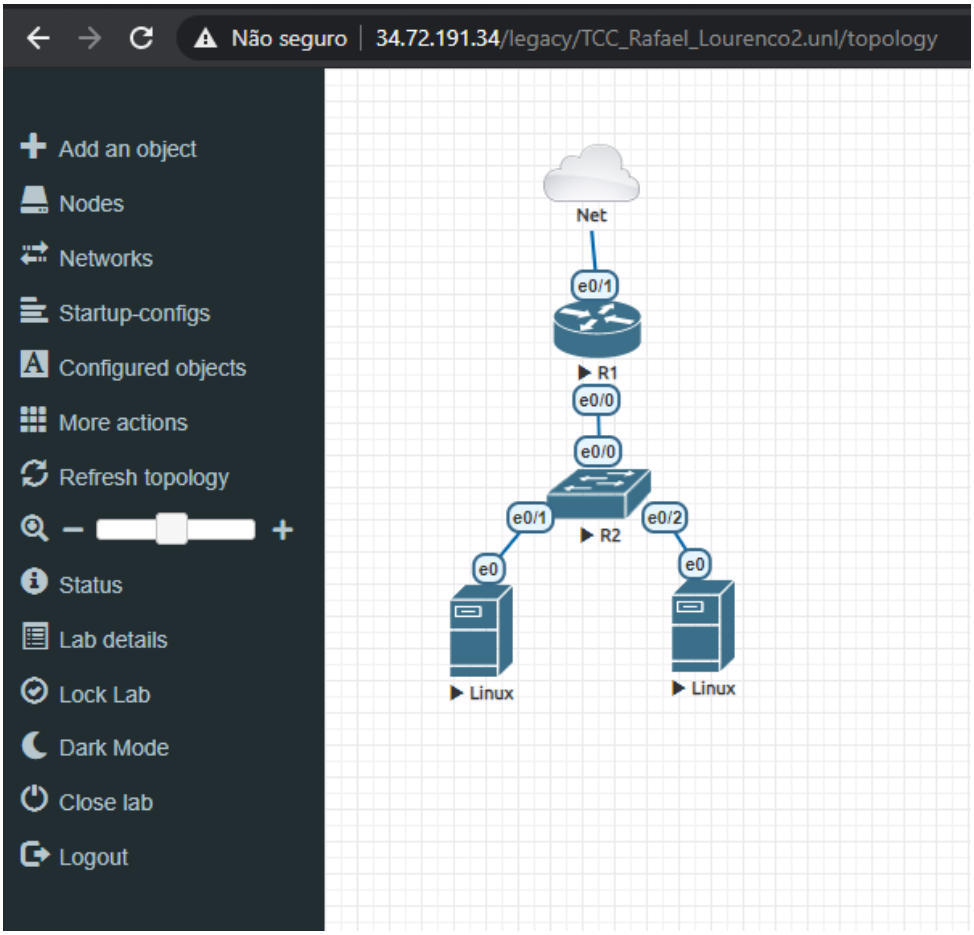
execução do comando */opt/unetlab/wrappers/unl_wrapper -a fixpermissions*, as imagens já estavam disponíveis para utilização dentro do laboratório conforme exemplo ilustrado na Figura 23.

Para finalizar a preparação do ambiente, foi configurado uma alternativa a fim de fazer com que os elementos emulados pudessem acessar à *Internet*. Quando a aplicação do EVE-NG é instalada, ela cria diversas interfaces de rede chamadas *Cloud* e disponibiliza-as internamente no laboratório como forma de redes. Portanto, foi necessário fazer um redirecionamento em que todo tráfego destinado internamente para a rede *Cloud1* fosse para a interface *eth0* do servidor o EVE-NG na *Google Cloud Platform*. Para realizar esse redirecionamento, foi adicionado um IP estático de 10.199.199.1/24 na interface pnet1 do servidor do EVE-NG e executado o comando listado abaixo.

```
iptables -t nat -A POSTROUTING -s 10.199.199.0/24 -o pnet0 -j MASQUERADE
```

Dessa forma, qualquer dispositivo emulado com uma interface conectada na rede *Cloud1* e endereçamento IP entre 10.199.199.2 e 10.199.199.254 é capaz de navegar na *Internet* e, por exemplo, fazer o *download* e a instalação de aplicações.

Figura 23 – Topologia Exemplo de *Router On a Stick* no EVE-NG



Fonte: O autor

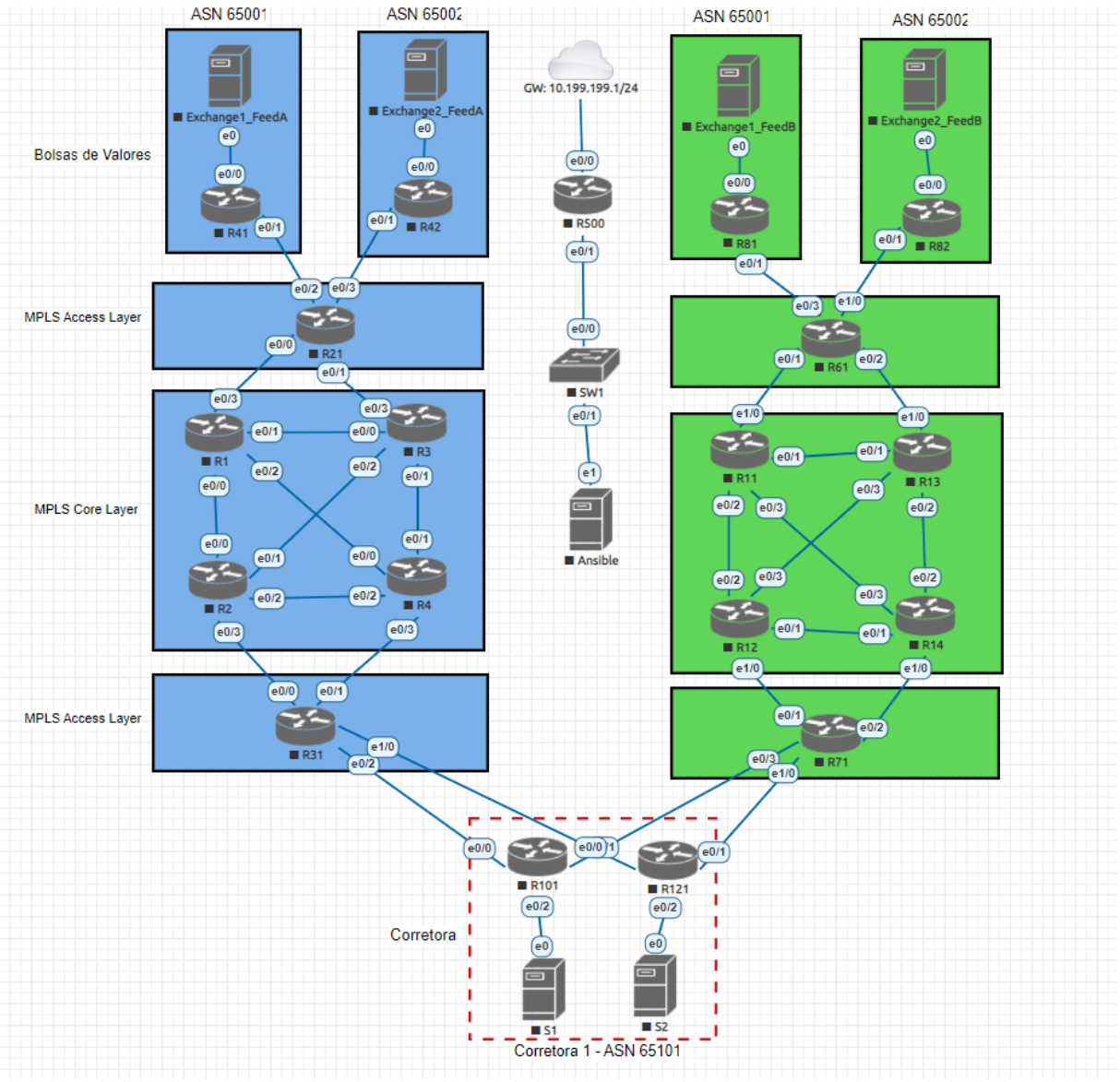
3.2.2 Caracterização da topologia de rede proposta

A próxima etapa foi a estruturação dos roteadores para a construção da topologia de rede proposta. As especificações da arquitetura foram baseadas no documento *Market Data Network Architecture* da fabricante Cisco (CISCO, 2008). A topologia de rede proposta está representada na Figura 24.

Para representação da separação física, o *Feed* A das Bolsas de Valores trafega pelos roteadores separados nos blocos em azul e o *Feed* B nos roteadores separados nos blocos em verde. Os critérios estabelecidos que devem ser atendidos pela infraestrutura são:

- As redes das Bolsas de Valores devem possuir o mesmo endereçamento IP Privado nos *Feeds* iguais, porém diferentes grupos *multicast* e ASN (*Autonomous System Number*);

Figura 24 – Topologia Proposta da Arquitetura de Rede



Fonte: O autor

- O RP deve ser auto anunciado e configurado em todos os roteadores CEs das Bolsas de Valores;
- O PIM-SM deve estar habilitado somente nas conexões entre os roteadores CEs e PEs;
- O *Core MPLS* deve executar o protocolo de roteamento OSPF;
- A *Multicast VPN* configurada entre os PEs deve seguir o Perfil 1 proposto pela Cisco (MDT-mLDP);

- A corretora deve receber o *Feed A* e *Feed B* da mesma Bolsa de Valores em diferentes roteadores;

No intuito de não comprometer visualmente a Figura 24, ficaram ocultas as ligações de todos os roteadores com o *switch* SW1. Essas conexões servem para que o Ansible possa conectar-se de forma remota em todos os roteadores da topologia para executar a automação. Essas conexões funcionam de forma análoga a uma interface do tipo *out-of-band*.

Outra questão pertinente à topologia são as interfaces de *Loopback* dos roteadores e o endereçamento IP das redes de trânsito. O endereçamento IP das interfaces de *Loopback* de todos os roteadores seguem a lógica do numeral do *hostname* do respectivo roteador repetido nos 4 octetos com a máscara de sub-rede 255.255.255.255 (/32). Para exemplificar, o roteador R11 possui um endereço IP na interface de *Loopback* de 11.11.11.11/32.

Já o endereçamento das redes de trânsito também utiliza o numeral do *hostname* de cada roteador. O esquema do endereçamento IP das interfaces *ethernet* está apresentado na Tabela 2.

Tabela 2 – Esquemático do endereçamento IP para as interfaces *ethernet*

Octeto	Valor
1º octeto	10
2º octeto	numeral do roteador com valor menor
3º octeto	numeral do roteador com valor maior
4º octeto	numeral do respectivo roteador
Todas as máscaras de sub-rede são 255.255.255.0 (/24)	

Portanto, a interface *ethernet* do roteador R1 que conecta com o roteador R2 possui endereço IP 10.1.2.1/24, enquanto o endereçamento IP na interface do roteador R2 é 10.1.2.2/24 e assim sucessivamente. Todas as interfaces de gerência dos roteadores utilizadas para conexão SSH via Ansible foram alocada na rede 172.16.1.0/24.

Os grupos *multicast* que serão utilizados na implementação também foram definidos e estão na Tabela 3.

Tabela 3 – Esquemático do endereçamento dos grupos *multicast*

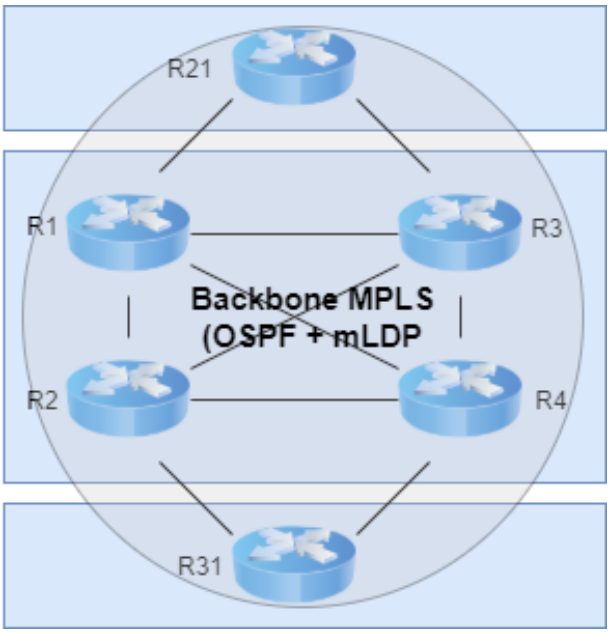
Bolsa de Valores	*Feed*	Grupo	*Wildcard Mask*
Exchange1	A	239.1.1.0	0.0.0.255
Exchange2	A	239.2.1.0	0.0.0.255
Exchange1	B	239.1.2.0	0.0.0.255
Exchange2	B	239.2.2.0	0.0.0.255

Para a geração dos 4 *streams* (um para cada servidor situado no bloco das Bolsas de Valores) utilizou-se o endereçamento IP com último octeto .10 para cada grupo *multicast* definido.

3.2.3 Configuração da rede MPLS

Com as definições e requisitos propostos para a arquitetura de rede, iniciou-se as configurações dos roteadores da camada de *MPLS Core*, demonstrada na Figura 25.

Figura 25 – Construção da camada *MPLS Core*



Fonte: O autor

Partiu-se para as configurações de IPs e da interface *Loopback0* seguindo as regras pré-estabelecidas na seção 3.2.2. Para que exista uma rede *MPLS* funcional, precisa-se que os roteadores da camada de *MPLS Core* sejam capazes de realizar o roteamento IP entre eles. No presente trabalho, o protocolo de roteamento escolhido para a camada de *MPLS Core* foi o OSPF. A identificação de cada roteador é o endereço IP da interface de *Loopback0* declarada previamente e todos os roteadores foram configurados com o OSPF *Single Area*, ou seja, todas as interfaces que compõem o MPLS estão na área 0. Para ilustrar, listou-se abaixo os comandos executados no roteador R1.

```
R1(config)#router ospf 1
R1(config-router)#router-id 1.1.1.1
R1(config)#int range e 0/0-3
R1(config-if-range)#ip ospf 1 area 0
```

```
R1(config)#int loo 0
R1(config-if)ip ospf 1 area 0
```

Em conjunto ao OSFP, configura-se o *Label Distribution Protocol* (LDP), que é o responsável por gerar e distribuir *labels* entre os roteadores. Cada roteador gera *labels* para seus prefixos e então faz o anúncio para seus *neighbors*. Quando o OSPF e LDP estão em execução, pode-se configurar de forma automática para que os prefixos que estão na RIB (*Routing Information Base* recebam um *label*. Abaixo, os comandos executados no roteador R1 para configuração do LDP. Foi seguido o mesmo processo para os demais roteadores constituintes do *MPLS Core*.

```
R1(config)#mpls label protocol ldp
R1(config)#mpls ldp router-id loopback 0
R1(config)#router ospf 1
R1(config-router)mpls ldp autoconfig
```

Desta forma, devido ao comando *mpls ldp autoconfig*, novos roteadores que entrarem na topologia sendo MPLS *Core* com as configurações supracitadas, já terão suas adjacências formadas com suas respectivas *labels*. Na configuração do *Core*, também foi habilitado o mldp através do comando *mpls mldp logging notification*.

3.2.4 Configuração da *Multicast VPN*

Para a configuração da *Multicast VPN*, selecionou-se o *Profile 1* proposto pela Cisco, que funciona como um túnel MDT-mLDP. Para que o tráfego *multicast* seja possível, é necessário que exista comunicação *unicast* entre os roteadores CE na mesma VRF. Logo, foi selecionado o protocolo de roteamento iBGP para formar um túnel entre os dois roteadores de borda do provedor (PEs). Foi escolhido arbitrariamente o ASN de número 100 e fechada uma adjacência iBGP através das interfaces de *Loopback* de ambos roteadores. Com isso, foi configurado um túnel VPNv4 entre os roteadores PEs, que será responsável por importar as rotas das VRFs das *Exchanges* e divulgar via iBGP. É importante ressaltar que, com a criação da VPNv4, para os roteadores P esse processo é transparente, ou seja, eles não transportam nenhuma rota de cliente.

A configuração de iBGP aplicada no roteador R21 está determinada na lista a seguir. Para o roteador R31 seguiu-se a mesma lógica, sendo o *neighbor* a interface 21.21.21.21.

```
R21(config)#router bgp 100
R21(config)#neighbor 31.31.31.31 remote-as 100
R21(config)#neighbor 31.31.31.31 update-source Loopback0
```

```
R21(config)#address-family ipv4
R21(config-router-af)neighbor 31.31.31.31 activate
R21(config-router-af) neighbor 31.31.31.31 next-hop-self

R21(config)#address-family vpnv4
R21(config-router-af)#neighbor 31.31.31.31 activate
R21(config-router-af)#neighbor 31.31.31.31 send-community both
```

O comando *update-source Loopback0* é responsável por formar a adjacência BGP entre os roteadores. É feito dessa forma para alterar a origem dos pacotes enviados para a *Loopback*, assim não é preciso formar duas adjacências BGP entre os roteadores PE e, caso falhe alguma interface física, a sessão BGP permanece no estado *up*. O comando *send-community both* faz a propagação das rotas entre os roteadores.

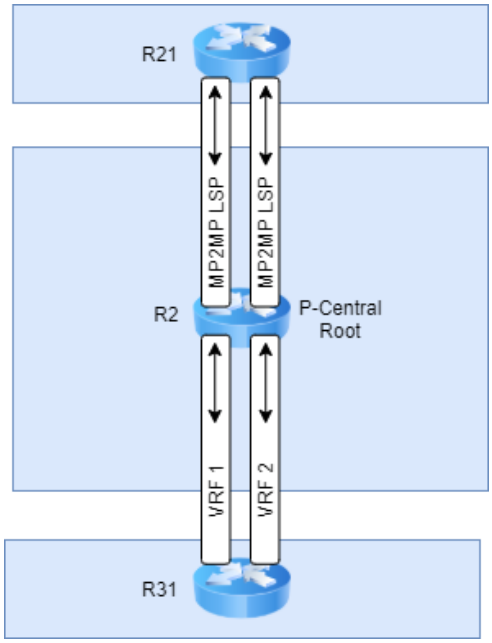
Para separar o tráfego entre as Bolsas de Valores, foi utilizado o recurso de VRF. Criou-se, então, uma VRF chamada de *Exchange_FeedA* para o *stream* da Bolsa de Valores 1 e *Exchange2_FeedA* para a Bolsa de Valores 2. Para que seja usada uma rede sobreposta ou privada, o *route distinguiser* configurado é único para cada VRF, seguindo o padrão "ASN:índice". O *route-target* também segue o mesmo padrão e serve para todas as rotas devem ser importadas e exportadas para dentro da VRF. A configuração de das VRFs no R21 estão listadas abaixo.

```
R21(config)#vrf definition Exchange1_FeedA
R21(config)#rd 100:1
R21(config)#address-family ipv4
R21(config-vrf-af)#route-target export 100:1
R21(config-vrf-af)#route-target import 100:1

R21(config)#vrf definition Exchange2_FeedA
R21(config)#rd 100:2
R21(config)#address-family ipv4
R21(config-vrf-af)#route-target export 100:2
R21(config-vrf-af)#route-target import 100:2
```

A configuração do MDT-mLDP é feito dentro da *adress-family ipv4* de VRF para que seja criada internamente a árvore *multicast*. Para a construção da árvore MDT *default* foi selecionado como *root* da topologia o roteador R2 formando um único MP2MP LSP, conforme ilustrado na Figura . Dentro de cada VRF, foi configurada também a vpn id, que é o valor responsável por indicar a mesma VPN em diferentes PE, ou seja, cada PE

com a mesma vpn id se juntará a mesma árvore MP2MP. Além disso, a vpn id é usada no contexto do mLDP para montar o *Opaque Value* transmitido no mLDP FEC.

Figura 26 – Cenário do *MDT default*



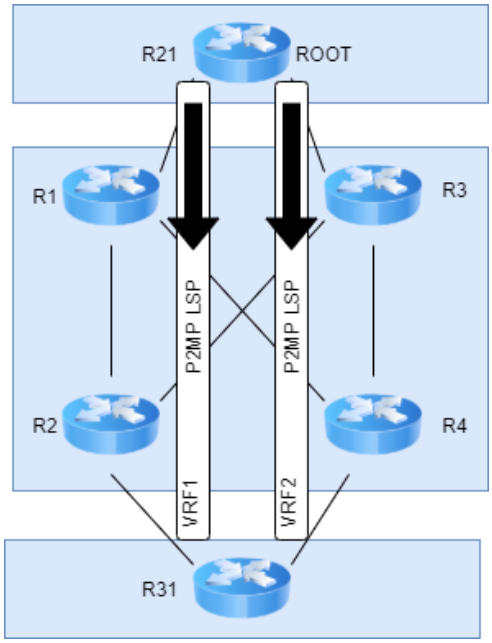
Fonte: O autor

Além disso, como a arquitetura para *Market Data* trata-se de um *stream multicast*, foi configurado o MDT *data* para que, caso o *stream* gerado seja superior a uma determinada largura de banda, é criada uma nova árvore *multicast* com um LSP Point-to-Multipoint, pois a única fonte na árvore *multicast* é proveniente da Bolsa de Valores, jamais das Corretoras. Ainda, o MDT *data* possibilita não onerar os roteadores que fazem parte da mesma VRF, porém não estão interessados em receber todos os canais *multicast*. A Figura 27 demonstra a topologia da MDT *data*. Nota-se a diferença do *root* e do sentido possível do tráfego em comparação à Figura 26.

Diferente da configuração de MDT *default*, não foi configurado o MDT *data* no roteador R31. Quando utiliza-se o MDT *data*, o roteador PE em que é configurado anuncia-se como *root* da topologia e é responsável pelo *switchover* do tráfego após atingir o limiar de 300kbps. Empiricamente, também foi escolhida a quantidade de MDT *data* possíveis para cada VRF. Neste caso, foram selecionadas 10, ou seja, a topologia suporta a adição de até 10 canais *multicast* em cada VRF. Abaixo, encontra-se a configuração realizada no roteador R21 para a formação da MDT *default* e *data*.

```
R21(config)#vrf definition Exchange1_FeedA
R21(config)#vpn id 100:1
```

Figura 27 – Cenário do *MDT data*



Fonte: O autor

```
R21(config)#address-family ipv4
R21(config-vrf-af)#mdt default mpls mldp 2.2.2.2
R21(config-vrf-af)#mdt data mpls mldp 10
R21(config-vrf-af)#mdt data threshold 300


R21(config)#vrf definition Exchange2_FeedA
R21(config)#vpn id 100:2
R21(config)#address-family ipv4
R21(config-vrf-af)#mdt default mpls mldp 2.2.2.2
R21(config-vrf-af)#mdt data mpls mldp 10
R21(config-vrf-af)#mdt data threshold 300
```

Neste momento, é criada automaticamente a interface virtual Lspvif0 e Lspvif1 nos roteadores R21 e R31, formando a adjacência PIM.

3.2.5 Configuração do *PIM SM e RP*

A última configuração realizada no Feed A foi a dos roteadores CEs e suas adjacências com o os PEs. Para isso, foi escolhido o protocolo de roteamento eBGP. Como nos PEs está já está rodando o iBGP, foi adicionada uma *address-family* para cada VRF dentro da sessão BGP. Nos CEs, foi configurado o BGP apenas dentro da *address-family*

de IPv4, visto que eles não possuem "conhecimento" sobre as VRFs. Na Tabela 4 estão os dados para a configuração do eBGP.

Tabela 4 – Relação de *Peers* BGP e ASN

Roteador	Designação	*peer*	ASN
R41	Bolsa 1 *Feed A*	10.21.41.41	65001
R42	Bolsa 2 *Feed A*	10.21.42.42	65002
R101	Corretora 1 Roteador 1	10.31.101.101	65101
R121	Corretora 1 Roteador 2	10.31.121.121	65101

A configuração do eBGP nos roteadores R21 e R31 está demonstrada na lista abaixo.

```
R21(config)#router bgp 100
R21(config)#address-family ipv4 vrf Exchange1_FeedA
R21(config-vrf-af)#neighbor 10.21.41.41 remote-as 65001
R21(config-vrf-af)#neighbor 10.21.41.41 activate

R21(config)#router bgp 100
R21(config)#address-family ipv4 vrf Exchange2_FeedA
R21(config-vrf-af)#neighbor 10.21.42.42 remote-as 65002
R21(config-vrf-af)#neighbor 10.21.42.42 activate

R31(config)#router bgp 100
R31(config)#address-family ipv4 vrf Exchange1_FeedA
R31(config-vrf-af)#neighbor 10.31.101.101 remote-as 65501
R31(config-vrf-af)#neighbor 10.31.101.101 activate

R31(config)#router bgp 100
R31(config)#address-family ipv4 vrf Exchange2_FeedA
R31(config-vrf-af)#neighbor 10.31.121.121 remote-as 65501
R31(config-vrf-af)#neighbor 10.31.121.121 activate
```

Os roteadores CEs (R41, R42, R101, R121), conforme citado anteriormente, seguiram a configuração padrão de BGP. Após, foram adicionadas nas interfaces de borda dos PEs, as respectivas VRFs que cada CE faz parte.

Apesar do parâmetro *route distinguiser* descrito na subseção 3.2.4 tornar as rotas únicas, foi necessário igualmente configurar um NAT *Network Address Translation*nos roteadores CE das Bolsas de Valores e da Corretora. Essa necessidade se fez presente pois, pensando em escalar a topologia para mais Corretoras, elas compartilhariam a mesma VRF e não poderiam possuir os mesmos *ranges* de IPs internos. O NAT traduz os endereços

IPs internos para os IPs da rede de trânsito entre CE e PE. O NAT configurado no R101 está demonstrado abaixo, sendo seguido o mesmo processo para os demais CEs.

```
R101(config)#interface ethernet 0/0
R101(config-if)#ip nat outside

R101(config-if)#interface ethernet 0/2
R101(config)#ip nat inside

R101(config)#ip nat inside source list NAT interface Ethernet0/0 overload
R101(config)#ip access list standard NAT
R101(config-std-nacl)# permit 172.16.1.0 0.0.0.255
R101(config-std-nacl)# permit 172.16.2.0 0.0.0.255
```

O PIM foi habilitado nas interfaces dos roteadores PEs que fazem conexão com os CEs. Ainda, habilitou-se em todas as interfaces dos CEs, incluindo as *Loopbacks*. As setas em preto na Figura ilustram as interfaces com o PIM habilitado. Para habilitar o protocolo, basta entrar no modo de configuração das interfaces e executar o comando *ip pim sparse-mode*. O protocolo IGMP versão 3 também foi habilitado entre as interfaces de conexão dos CEs com os servidores, tanto os *senders* quanto os *receivers*.

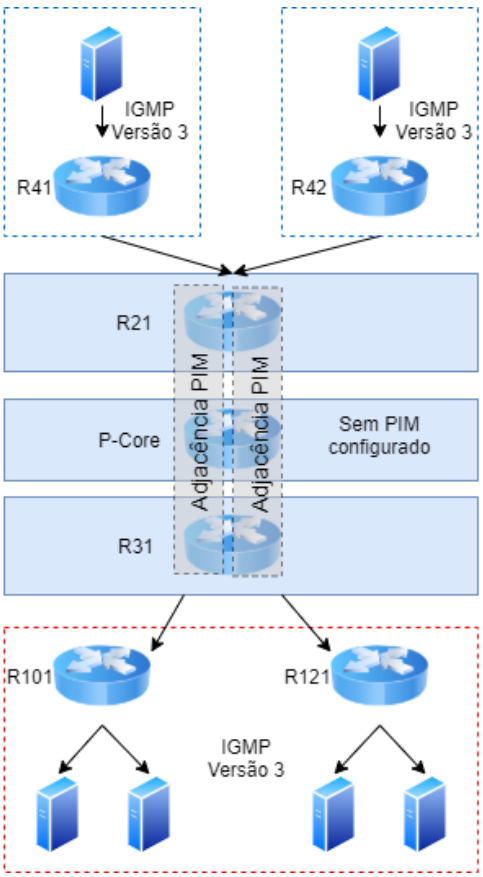
Por fim, foi configurado o RP nos roteadores R41 e R42. Para isso, foi utilizado o BSR (*Multicast PIM Bootstrap*), que serve para encontrar automaticamente o RP em uma rede *multicast*. A configuração do RP foi executada através dos comandos abaixo. O *Candidate BSR* coleta informações de todos os RPs disponíveis e anuncia na rede, enquanto o *Candidate RP* anuncia a si próprio com o desejo de tornar-se RP da rede.

```
R41(config)#ip pim bsr-candidate Loopback0 0
R41(config)#ip pim rp-candidate Loopback0

R42(config)#ip pim bsr-candidate Loopback0 0
R42(config)#ip pim rp-candidate Loopback0
```

Para o teste de transmissão e recebimento do *stream multicast* instalou-se o VLC nos servidores *Exchange1_FeedA*, *Exchange2_FeedA*, S1 e S2. O canal *multicast* transmitido pelo *Exchange1_FeedA* é o 239.1.1.10 e pelo *Exchange2_FeedB* é o 239.2.1.10, cada um transmitindo um vídeo distinto.

Inicialmente, através do VLC do servidor *Exchange1_FeedA*, foi testado para transmitir um *stream* no endereço udp://239.1.1.10:1234. O servidor S1 foi configurado com o mesmo endereço e não foi possível abrir o *stream multicast*. Verificou-se que o

Figura 28 – Interfaces com *PIM Sparse Mode* habilitadas



Fonte: O autor

método UDP é legado e não é o mais indicado para esse tipo de *stream*. Logo, alterando para o método RTP/UDP, configurando o endereço rtp://239.1.1.10:5004 e adicionando o campo *ttl=15s* na transmissão, o servidor S1 conseguiu abrir o *stream multicast*. O mesmo ocorreu para o servidor S2 no endereço rtp://239.2.1.0:5004 com a fonte sendo o servidor *Exchange2_FeedA*. Com o auxílio do *software wireshark* foram capturados os pacotes das interfaces dos servidores.

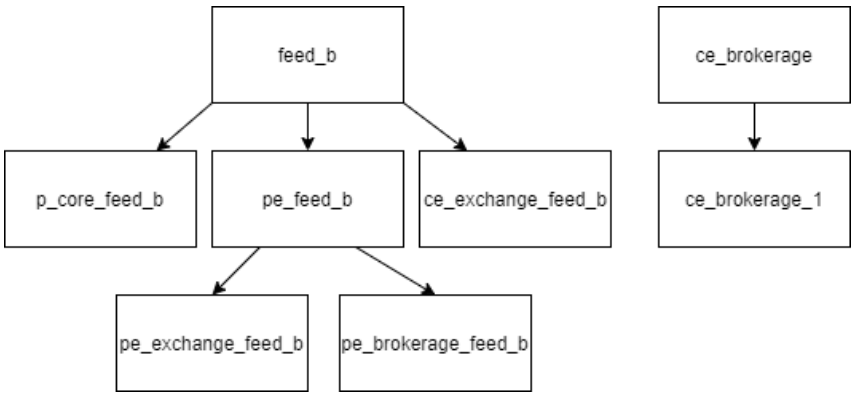
3.2.6 Preparação do servidor Ansible

Com o *Feed A* estruturado e configurado manualmente, pode-se passar para a instalação do servidor do Ansible, que será responsável pelo provisionamento das configurações do *Feed B* de forma automática. Primeiramente, foi instalado o Ansible no servidor central, indicado na topologia presente na Figura 24, que possui sistema operacional *Ubuntu* . A instalação é bastante simples e executada através dos comandos listados abaixo. Para verificar a instalação, foi executado o comando *ansible –version*.

```
$ sudo apt update
$ sudo apt install software-properties-common
$ sudo apt-add-repository --yes --update ppa:ansible/ansible
$ sudo apt install ansible

#Validação da instalação no servidor Ansible

rlourenco@Linux-Desktop:~/ansible$ ansible --version
ansible 2.9.20
config file = /home/rlourenco/ansible/ansible.cfg
configured module search path = [u'/home/rlourenco/.ansible/plugins/modules
    ', u'/usr/share/ansible/plugins/modules']
ansible python module location = /usr/lib/python2.7/dist-packages/ansible
executable location = /usr/bin/ansible
python version = 2.7.17 (default, Feb 27 2021, 15:10:58) [GCC 7.5.0]
```

Em seguida, foi criado o diretório *ansible* na pasta *home* do usuário *rlourenco*. Esse diretório possui o arquivo de configuração do Ansible, o arquivo de *hosts*, a pasta de *playbooks*, a pasta *group_vars* e a pasta *host_vars*. A estrutura do arquivo *hosts*, com a lógica dos grupos e grupos "filhos" desenvolvida, está na Figura 29.

Figura 29 – Estrutura dos grupos no arquivo *hosts*



Fonte: O autor

O arquivo *hosts* completo com os respectivos roteadores pode ser encontrado no Apêndice F.

Para que o servidor *Ansible* pudesse conectar-se com os roteadores, foi necessário fazer a ligação da interface *ethernet* 0/0 de todos os roteadores no *switch* SW1 e configurar o SSH em cada roteador do *Feed B*. A configuração de SSH em cada roteador foi a

única aplicada manualmente e está descrita na lista abaixo. O usuário para acesso SSH é *rlourenco* e a senha *cisco*.

```
en
conf t
int e0/0
ip address 172.168.1.x 255.255.255.0
no shut
exit
username rlourenco password cisco
username rlourenco privilege 15
line vty 0 4
transport input all
login local
exit
ip domain-name rlourenco.com
crypto key generate rsa
1024
do wr
end
```

Os IPs dos servidores foram adicionados nos arquivos YAML de cada roteador dentro da pasta *host_vars*. A variável para a alocação do IP foi a *ansible_host*, que é uma variável nativa do Ansible. Após concluídas as configurações, foi possível acessar os roteadores do *Feed B* através do SSH.

Para finalizar a preparação do ambiente, instalou-se o *plugin cisco.ios*. A *collection* Ansible Cisco IOS inclui uma variedade de módulos responsáveis por automatizar as configurações dos equipamentos de rede Cisco IOS. A *collection* é instalada através do Ansible Galaxy executando o comando.

```
ansible-galaxy collection install cisco.ios
```

Assim, o módulo *ios_config* e a conexão SSH a partir do servidor Ansible para todos roteadores mostrou-se funcional.

3.2.7 Construção e execução dos *playbooks*

Para a construção dos *playbooks*, optou-se por separar um *playbook* para cada bloco de roteadores. Os *playbooks* desenvolvidos foram os seguintes:

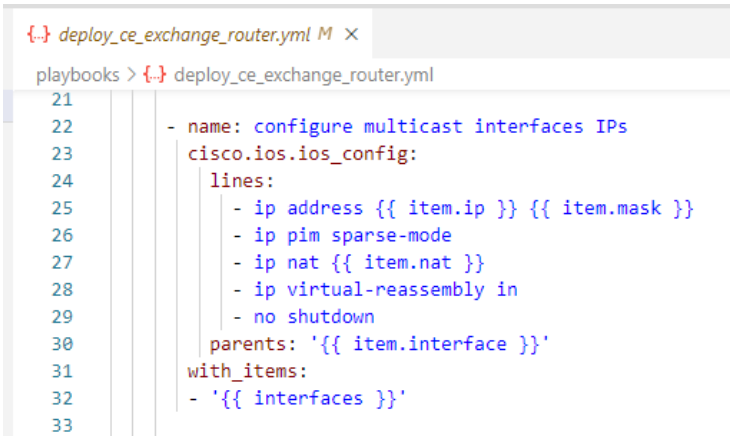
- deploy_ce_brokerage_router.yml
- deploy_ce_exchange_router.yml
- deploy_p_core_router.yml
- deploy_pe_brokerage_router.yml
- deploy_pe_exchange_router.yml

Como pode-se identificar na extensão dos nomes, os *playbooks* são arquivos YAML. O módulo *cisco.ios* possui diversos parâmetros possíveis para uso nas *tasks*. Basicamente, para o desenvolvimento dos *playbooks* no presente trabalho, foram usados três parâmetros principais: *lines*, *parents* e *with_items*. Os parâmetros são declarados como listas YAML.

No parâmetro *lines* estão os comandos a serem executados no roteador e o *parents* representa a sessão ou hierarquia em que os comandos devem ser executados. O parâmetro *with_items* é nativo do Ansible e é utilizado para fazer um *loop*, a fim de repetir uma *task* diversas vezes.

A ilustração do uso dos parâmetros pode ser encontrada na Figura 30. Essa *task* é uma das tarefas do *playbook deploy_ce_exchange_router.yml* responsável por adicionar as configurações nas interfaces dos roteadores CE das Bolsas de Valores que possuem o *PIM Spase Mode* habilitado.

Figura 30 – *Task* para configuração das interfaces *multicast*



Fonte: O autor

Os valores que estão dispostos entre chaves, por exemplo, '{{ *item.interface* }}' são as variáveis que estão declaras nos arquivos de *group_vars* e *host_vars*. Cada roteador possui um arquivo YAML com as variáveis declaradas internamente. Na Figura 31 estão

Figura 31 – Arquivo R81.yml da pasta *host_vars*

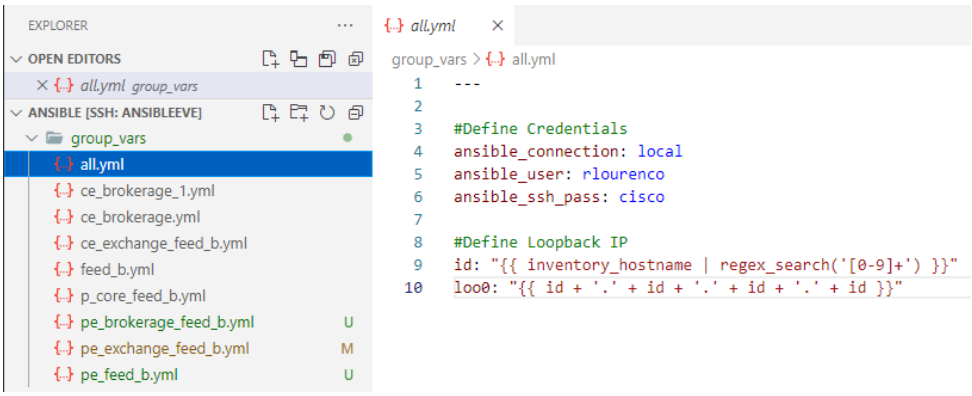
Fonte: O autor

demonstradas as variáveis que precisam ser inseridas no roteador R81 para que o *playbook deploy_ce_exchange_router.yml* possa ser executado sem que ocorram falhas.

Assim, ao executar a *task* ilustrada na Figura 30, devido ao parâmetro *with_itens*, serão configuradas todas as interfaces que forem inseridas dentro do *host_vars*.

Todos os *playbooks* desenvolvidos podem ser encontrados na sessão de Apêndices deste trabalho. As configurações dos roteadores presentes no *Feed A* serviram como base para o desenvolvimento dos *playbooks*. A definição do usuário e senha SSH e o IP da interface de *Loopback* foram definidos no arquivo all.yml dentro de *group_vars* conforme a Figura 32. Essas variáveis são globais e são utilizadas para todos os roteadores.

Figura 32 – Variáveis do arquivo *all.yml*



Fonte: O autor

Por fim, depois de finalizados os *playbooks*, os mesmos foram executados nos roteadores do *Feed B* da topologia. O comando para a execução dos *playbooks* é dado

por *ansible-playbook playbooks/(nome do playbook).yml –extra-vars="target=(grupos)"*. O *playbook* é executado de forma paralela quando a variável *target* é um grupo de *hosts*, ou seja, vai gerar a configuração em todos roteadores simultaneamente. Na Figura está ilustrada o comando para o *deploy* das configurações nos roteadores P e uma fração da saída.

Figura 33 – Execução do *playbook deploy_p_core_router.yml*

```
rlourenco@Linux-Desktop:~/ansible$ ansible-playbook playbooks/deploy_p_core_router.yml --extra-vars="target=p_core_feed_b"

PLAY [p_core_feed_b] ***********************************************************************************************************************************

TASK [inicial config] **********************************************************************************************************************************
[WARNING]: To ensure idempotency and correct diff the input configuration lines should be similar to how they appear if present in the
running configuration on device
[WARNING]: [u'connection local support for this module is deprecated and will be removed in version 2.14, use connection
ansible.netcommon.network_cli']
changed: [R13]
changed: [R12]
changed: [R14]
changed: [R11]

TASK [configure mpls] **********************************************************************************************************************************
changed: [R11]
changed: [R13]
changed: [R12]
changed: [R14]

TASK [configure interfaces ip and enable ospf] *********************************************************************************************************
changed: [R11] => (item={u'interface': u'interface Loopback0', u'ip': u'11.11.11.11', u'mask': u'255.255.255.255'})
changed: [R13] => (item={u'interface': u'interface Loopback0', u'ip': u'13.13.13.13', u'mask': u'255.255.255.255'})
changed: [R12] => (item={u'interface': u'interface Loopback0', u'ip': u'12.12.12.12', u'mask': u'255.255.255.255'})
changed: [R14] => (item={u'interface': u'interface Loopback0', u'ip': u'14.14.14.14', u'mask': u'255.255.255.255'})
changed: [R12] => (item={u'interface': u'interface Ethernet0/1', u'ip': u'10.12.14.12', u'mask': u'255.255.255.0'})
changed: [R11] => (item={u'interface': u'interface Ethernet0/1', u'ip': u'10.11.13.11', u'mask': u'255.255.255.0'})
changed: [R13] => (item={u'interface': u'interface Ethernet0/1', u'ip': u'10.11.13.13', u'mask': u'255.255.255.0'})
changed: [R14] => (item={u'interface': u'interface Ethernet0/1', u'ip': u'10.12.14.14', u'mask': u'255.255.255.0'})
changed: [R12] => (item={u'interface': u'interface Ethernet0/2', u'ip': u'10.11.12.12', u'mask': u'255.255.255.0'})
changed: [R11] => (item={u'interface': u'interface Ethernet0/2', u'ip': u'10.11.12.11', u'mask': u'255.255.255.0'})
changed: [R13] => (item={u'interface': u'interface Ethernet0/2', u'ip': u'10.13.14.13', u'mask': u'255.255.255.0'})
changed: [R14] => (item={u'interface': u'interface Ethernet0/2', u'ip': u'10.13.14.14', u'mask': u'255.255.255.0'})
changed: [R12] => (item={u'interface': u'interface Ethernet0/3', u'ip': u'10.12.13.12', u'mask': u'255.255.255.0'})
changed: [R11] => (item={u'interface': u'interface Ethernet0/3', u'ip': u'10.11.14.11', u'mask': u'255.255.255.0'})
changed: [R13] => (item={u'interface': u'interface Ethernet0/3', u'ip': u'10.12.13.13', u'mask': u'255.255.255.0'})
```

Fonte: O autor

O texto em amarelo demonstra que o Ansible alterou as configurações para cada roteador do grupo *p_core_feed_b* conforme a ordem das *tasks* dos *playbooks*. Além disso, pode-se perceber que cada *task* é executada ao mesmo tempo em cada roteador. O *stream multicast* nos servidores da corretora não foi recebido após a primeira execução dos *playbooks*. Após um *troubleshooting*, percebeu-se que os roteadores R101 e R121 estavam considerando apenas os roteadores R41 e R42 como RP da topologia. Assim, foi reconfigurado nos roteadores R41 e R42 para serem o RP apenas dos grupos *multicast* com range 239.1.1.0 0.0.0.255 e 239.1.2.0 0.0.0.255, respectivamente. Ainda, foi adicionado essa configuração no *playbook deploy_ce_brokerage_router.yml* e os ranges 239.2.1.0 0.0.0.255 e 239.2.2.0 0.0.0.255 na variável *multicast_range* dentro dos arquivos de variáveis do R101 e R121.

Portanto, foi possível receber o *Feed A* da Bolsa de Valores 1 e o *Feed B* da Bolsa de Valores 2 no servidor S1 e o *Feed B* da Bolsa de Valores 1 o *Feed A* da Bolsa de Valores 2 no S2. Foi feita a coleta dos pacotes nas interfaces *ens3* dos servidores para verificação do recebimento dos canais *multicast*.

4 RESULTADOS E DISCUSSÕES

4.1 Análise das configurações do *Feed A*

A primeira análise realizada corresponde às configurações manuais dos roteadores realizas no *Feed A*. Conforme descrito na subseção 3.2.5, foi possível receber o *stream multicast* em ambos servidores receptores situados na Corretora.

Como a solução implementada na rede MPLS foi uma MDT-mLDP, validou-se que não há o protocolo *PIM Sparse Mode* em execução nos roteadores *Core*, sendo passado o tráfego *multicast* todo pelo túnel. Abaixo, pode-se verificar com o resultado do comando que não há nenhum vizinho PIM no roteador R2.

```
   R2#show ip pim neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, DR - Designated Router, N - Default DR Priority,
      P - Proxy Capable, S - State Refresh Capable, G - GenID Capable,
      L - DR Load-balancing Capable
Neighbor Interface Uptime/Expires Ver DR
Address Prio/Mode
```

O túnel PIM, conforme proposto pela *Multicast VPN*, foi fechado entre os roteadores R21 e R31 dentro de cada VRF. A saída abaixo no roteador R21 demonstra os vizinhos PIM. Pode-se perceber que é fechado um túnel para cada VRF através das interfaces *Lspvif0* e *Lspvif1*, visto que elas formam uma vizinhança com a interface de *Loopback0* do roteador R31. A outra adjacência PIM de cada VRF é formada com o IP de borda dos roteadores CEs.

```
   R21#show ip pim vrf Exchange1_FeedA neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, DR - Designated Router, N - Default DR Priority,
      P - Proxy Capable, S - State Refresh Capable, G - GenID Capable,
      L - DR Load-balancing Capable
Neighbor Interface Uptime/Expires Ver DR
Address Prio/Mode
10.21.41.41 Ethernet0/2 04:23:41/00:01:36 v2 1 / DR S P G
31.31.31.31 Lspvif0 06:04:38/00:01:39 v2 1 / DR S P G
```

```
   R21#show ip pim vrf Exchange2_FeedA neighbor
PIM Neighbor Table
Mode: B - Bidir Capable, DR - Designated Router, N - Default DR Priority,
      P - Proxy Capable, S - State Refresh Capable, G - GenID Capable,
      L - DR Load-balancing Capable
Neighbor Interface Uptime/Expires Ver DR
Address Prio/Mode
10.21.42.42 Ethernet0/3 06:07:39/00:01:39 v2 1 / DR S P G
31.31.31.31 Lspvif1 06:04:45/00:01:39 v2 1 / DR S P G
```

As MDTs dentro da mLDP VPN podem ser vistas na lista abaixo. As duas MDT *defaults* criadas são as MP2MP, sendo o *root* da topologia o roteador 2.2.2.2. Como o *stream* do vídeo ultrapassou a taxa de *threshold* determinada de 300kbps, foram criadas as MDT *datas* corretamente. O *root* da MDT *data* é o roteador R21, sendo o comportamento padrão esperado devido ao auto anúncio do mesmo.

```
   R21#show mpls mldp database sum

LSM ID Type Root Decoded Opaque Value Client Cnt.
5 P2MP 21.21.21.21 [mdt 100:1 1] 2
6 P2MP 21.21.21.21 [mdt 100:2 1] 2
1 MP2MP 2.2.2.2 [mdt 100:1 0] 1
3 MP2MP 2.2.2.2 [mdt 100:2 0] 1
```

Foram avaliadas as tabela de roteamento *multicast* e o *Redezvous Point* configurados nos roteadores R41 e R42. Como o *PIM Sparse Mode* adota a estratégia de que nenhum *host* deseja receber o conteúdo *multicast* até que haja uma manifestação explícita de interesse, pode-se verificar que houve um *join* dos servidores S1 e S2 nos endereços *multicast* 239.1.1.10 e 239.2.1.10, assim como os servidores *Exchange1_FeedA* e *Exchange2_FeedA* responderam nos respectivos canais formando a árvore *multicast*.

```
   R41#show ip mroute sum
   (*, 239.1.1.10), 02:25:12/00:03:14, RP 41.41.41.41, OIF count: 1, flags:
      SF
   (192.168.1.10, 239.1.1.10), 02:25:12/00:02:17, OIF count: 1, flags: FT

   R42#show ip mroute sum
   (*, 239.2.1.10), 02:23:46/00:02:35, RP 42.42.42.42, OIF count: 1, flags:
      SF
   (192.168.1.10, 239.2.1.10), 02:23:46/00:03:26, OIF count: 1, flags: FT
```

Na lista acima está o resultado abreviado da saída do comando *show ip mroute*. No texto destacado em amarelo, pode-se identificar que os grupos *multicast* estão ativos e que os dois remetentes de grupos diferentes possuem o mesmo endereço IP privado. Ainda na lista acima, pode-se verificar qual é o RP de cada uma das árvores, indicando o funcionamento do *Multicast PIM Bootstrap*. O resultado do comando *show ip mroute count* executado duas vezes no roteador R41 está indicado na lista abaixo. Os comandos foram executados com 30s de diferença, demonstrando que há tráfego *multicast* passando no canal.

```
R41#show ip mroute count
Group: 239.1.1.10, Source count: 2, Packets forwarded: 261650, Packets
    received: 268136
Source: 192.168.1.10/32, Forwarding: 261650/25/1346/276, Other:
    268136/0/6486


R42#show ip mroute count
Group: 239.1.1.10, Source count: 2, Packets forwarded: 262484, Packets
    received: 268970
Source: 192.168.1.10/32, Forwarding: 262484/27/1346/299, Other:
    268970/0/6486
```

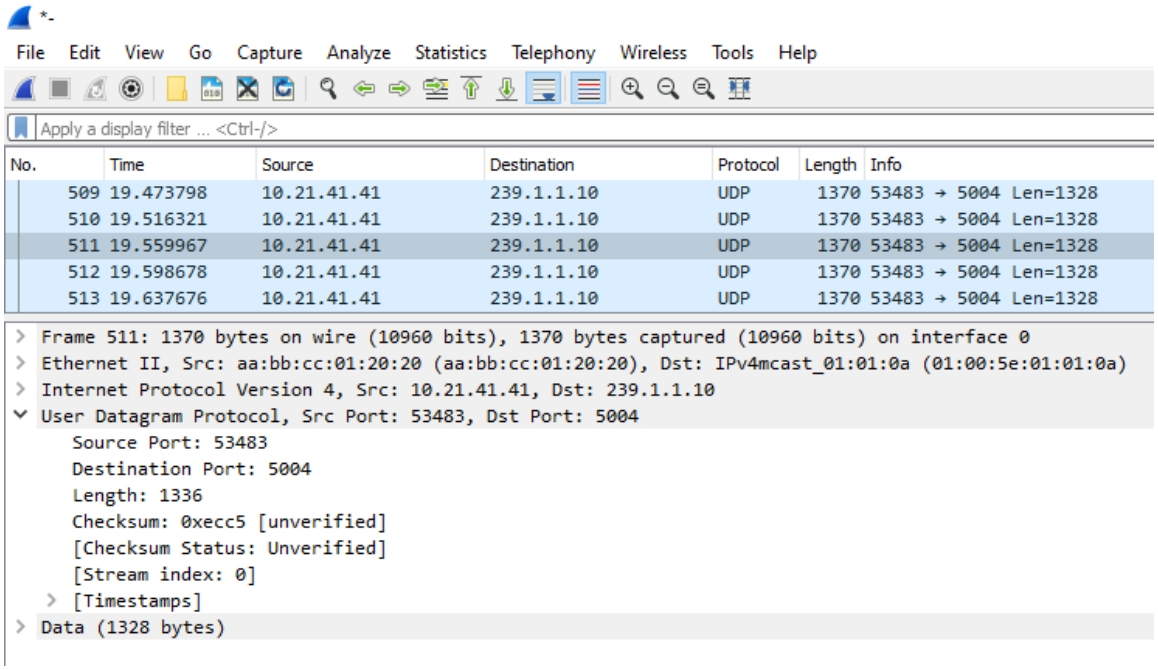
Por fim, o resultado da captura de pacotes das interfaces dos servidores com o *wireshark*. Na Figura 34 está o resultado da captura da interface ens3 do servidor S1. Pode-se identificar que o protocolo de transporte é o UDP e que o *source* é o IP da rede de trânsito do R41, demonstrando a funcionalidade do NAT. Em *Destination*, o canal *multicast* 239.1.1.10 configurado no *stream*. O mesmo segue para a captura dos pacotes no servidor S2 demonstrado na Figura 35.

Percebeu-se que o vídeo nos receptores chegou com um atraso de aproximadamente 3s. Esse *delay* pode ser explicado pelos equipamentos serem virtualizados sob uma máquina virtual, o que acarreta em uma queda considerável no desempenho dos mesmos. Um exemplo é a saída de um comando *ping* para a interface de *Loopback* do R41 com origem do servidor S1. O tempo de resposta variou de 1,62ms até 19,2ms, sendo considerado um atraso alto visto que não há distância entre os *links*.

4.2 Testes com os *playbooks* executados no *Feed B*

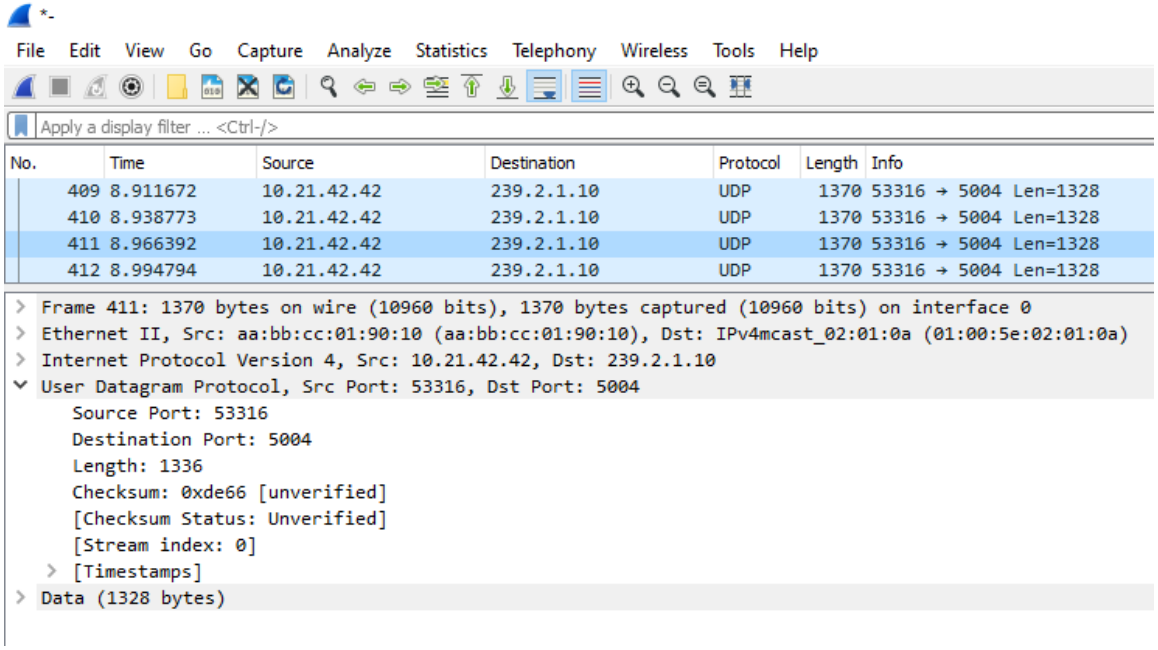
A segunda parte da análise foi dividida entre a verificação da recepção do *multicast* nos servidores S1 e S2 e a avaliação das configurações geradas pelo Ansible. Conforme

Figura 34 – Captura de pacotes com o *wireshark* no S1



Fonte: O autor

Figura 35 – Captura de pacotes com o *wireshark* no S2



Fonte: O autor

descrito no final da seção 3.2.7, foi necessário definir os grupos *multicast* para cada RP. Na forma como estava configurado previamente, os roteadores do *Feed A* anunciavam-se como RPs para todos os grupos *multicast* da rede, sendo assim, os roteadores R101 e R121 conheciam apenas os roteadores R41 e R42 como RP. Logo, as mensagens de *join* dos servidores S1 e S2 estavam indo para o RP que não conhecia grupo do *stream* em que eles estavam interessados. Com a alteração no *AUTO-RP-DISCOVERY*, o mapeamento do RP pode ser vista na lista abaixo.

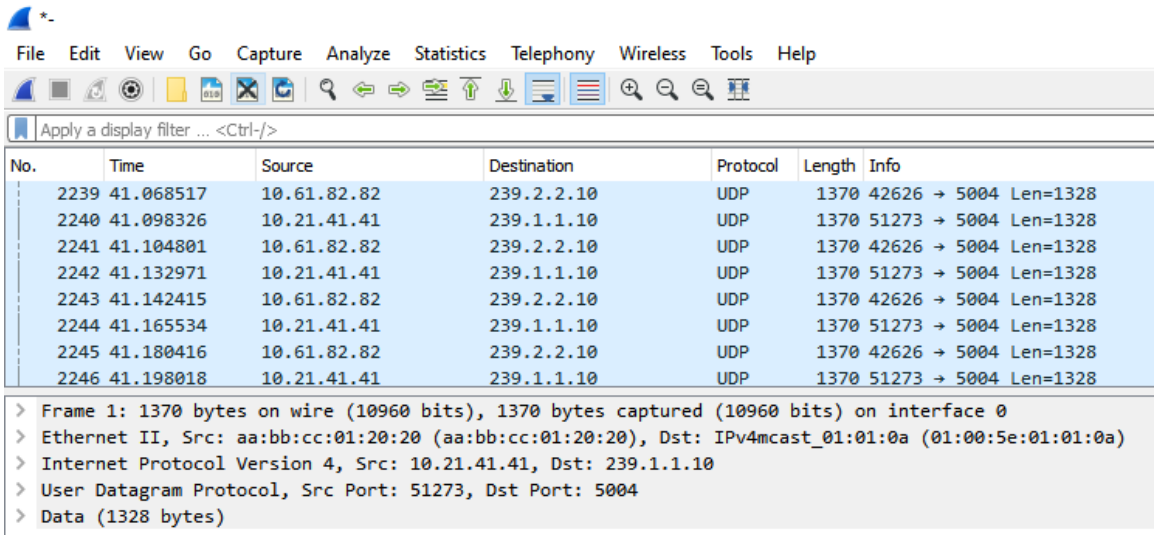
```
    R101#show ip pim rp mapping
PIM Group-to-RP Mappings

Group(s) 239.1.1.0/24
  RP 41.41.41.41 (?), v2
    Info source: 41.41.41.41 (?), via bootstrap, priority 0, holdtime 150
         Uptime: 01:59:48, expires: 00:02:11
Group(s) 239.2.2.0/24
  RP 82.82.82.82 (?), v2
    Info source: 82.82.82.82 (?), via bootstrap, priority 0, holdtime 150
         Uptime: 01:49:57, expires: 00:01:53


    R121#show ip pim rp mapping
PIM Group-to-RP Mappings

Group(s) 239.1.2.0/24
  RP 81.81.81.81 (?), v2
    Info source: 81.81.81.81 (?), via bootstrap, priority 0, holdtime 150
         Uptime: 01:15:06, expires: 00:02:19
Group(s) 239.2.1.0/24
  RP 42.42.42.42 (?), v2
    Info source: 42.42.42.42 (?), via bootstrap, priority 0, holdtime 150
         Uptime: 01:28:30, expires: 00:02:03
```

Portanto, poderiam ser adicionadas novas Bolsas de Valores na arquitetura com diferentes grupos *multicast* sem impacto aos RPs em produção. Um detalhe incoerente encontrado foi o fato dos novos RPs configurados no *Feed B* não terem tornado-se primário topologia. De acordo com (CISCO, 2002), quando forem configurados múltiplos *Rendezvous Points* e estes atenderem aos mesmos grupos *multicast*, o RP com maior endereço IP será escolhido como primário. Este mecanismo suporta a criação de uma infraestrutura

redundante, pois o roteador com o menor endereço IP atuará como backup nos eventos de falha do equipamento principal.

A captura das interfaces *ens3* dos servidores S1 e S2 estão expostos nas Figura 36 e 37, respectivamente. Os servidores receberam o *stream* nos quatro grupos *multicast* determinados e com o IP de origem de NAT dos roteadores CEs da Bolsa de Valores. Além disso, os *Feeds A e B*, tanto para a *Exchange1* quanto para a *Exchange2* chegaram em roteadores diferentes na Corretora, garantindo a redundância exigida pela rede *Market Data*.

Figura 36 – Captura de pacotes com o *wireshark* no S1



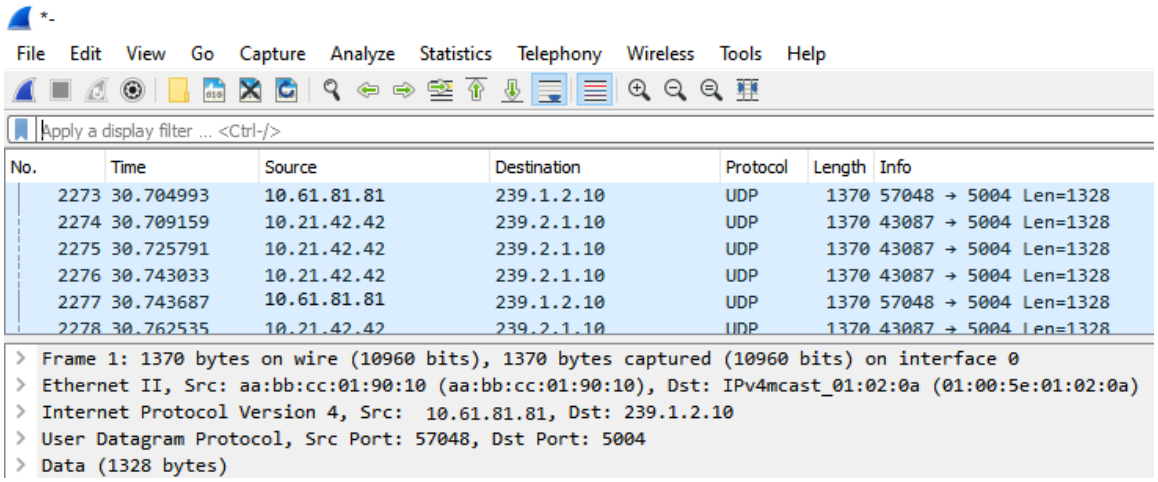
Fonte: O autor

Para validar os resultados das configurações com o Ansible nos roteadores do *Feed B*, foi desenvolvido um *playbook* que faz a diferença entre as configurações da *startup_config* e a *running_config*. A *startup_config* é a configuração utilizada pelo roteador para fazer o *boot* do sistema operacional. Quando é utilizado o comando *wr* no Cisco IOS, as configurações da *running_config* são passadas para a *startup_config*, ou seja, ele faz o salvamento das configurações. O *playbook* para a diff usa o mesmo módulo *cisco.ios.ios_config* e está demonstrado na Figura 38.

Dessa forma, conforme pode-se verificar na Figura 39, o *playbook* foi executado no grupo *all*, abrangendo todos os roteadores. A saída da execução do *playbook* demonstra as configurações que foram adicionadas e removidas pelo Ansible e está descrita de forma completa no Apêndice G.

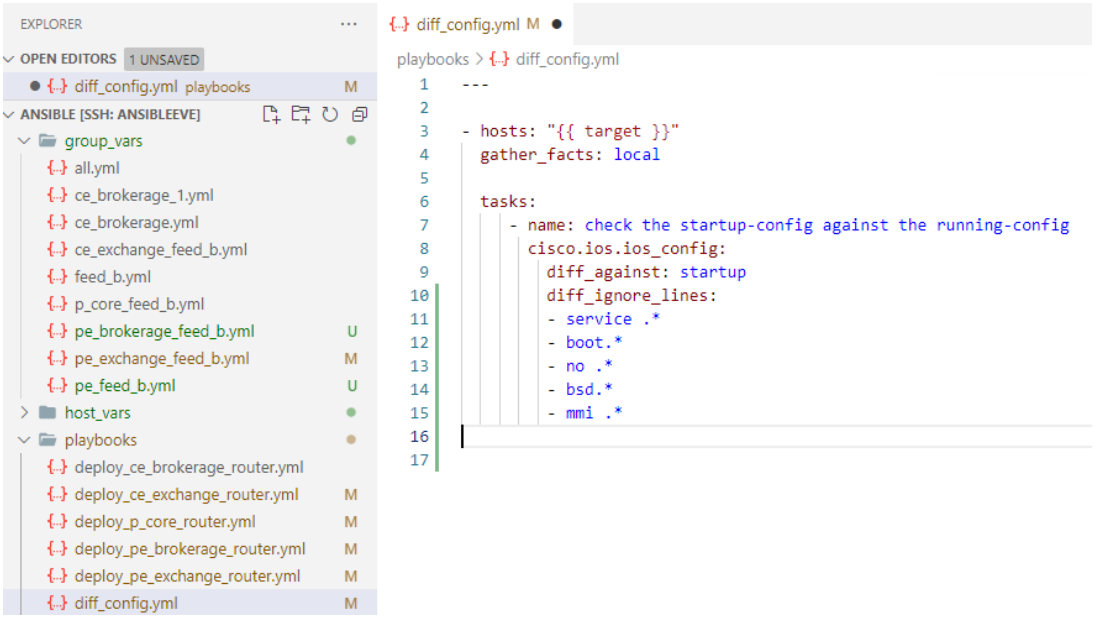
Foi possível observar que o Ansible é capaz de aplicar corretamente as configurações nos roteadores através dos *playbooks* com o módulo *cisco.ios*, tendo em vista que as

Figura 37 – Captura de pacotes com o *wireshark* no S2



Fonte: O autor

Figura 38 – *Playbook* para diferença de configurações



Fonte: O autor

configurações ficaram de acordo com o esperado. Além disso, foi medido o tempo de execução de cada *playbook* apresentado na Tabela 5.

Pelos tempos encontrados, percebe-se que os *playbooks* com maior duração para finalizar a execução são aqueles com maior número de *tasks*, apesar de terem como *target* somente um roteador. O *playbook* de *deploy_p_core_router.yml* foi executado em 4 roteadores e, por ter um número menor de *tasks* para realizar, finalizou sua execução mais

Figura 39 – Execução do *playbook* de diferença de configuração em todos roteadores

```
rlourenco@Linux-Desktop:~/ansible$ ansible-playbook --diff playbooks/diff_config.yml --extra-vars="target=all"

PLAY [all] *********************************************************************************************************************************************

TASK [check the startup-config against the running-config] *********************************************************************************************
--- before
+++ after
@@ -1,12 +1,28 @@
-hostname R
+hostname R13
+interface Loopback0
+ ip address 13.13.13.13 255.255.255.255
+ ip ospf 1 area 0
  ip address 172.168.1.13 255.255.255.0
 interface Ethernet0/1
+ ip address 10.11.13.13 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/2
+ ip address 10.13.14.13 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/3
+ ip address 10.12.13.13 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/0
+ ip address 10.13.61.13 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router ospf 1
+ mpls ldp sync
+ mpls ldp autoconfig
+ router-id 13.13.13.13
+mpls ldp router-id Loopback0
  login local
  transport input all
```

Fonte: O autor

Tabela 5 – Tempo de execução dos *playbooks*

Playbook	Tempo (s)
deploy_ce_brokerage_router.yml	21
deploy_ce_exchange_router.yml	17
deploy_p_core_router.yml	23
deploy_pe_brokerage_router.yml	38
deploy_pe_exchange_router.yml	36

rapidamente. Esse resultado demonstra a capacidade do Ansible em executar as tarefas paralelamente em diversos *hosts*.

5 CONCLUSÕES

Esse trabalho teve como objetivo conceber e implementar o modelo de uma arquitetura de rede *Market Data* com o auxílio da ferramenta de automação de redes Ansible. Para isso, foi realizado um estudo visando caracterizar os principais componentes da arquitetura e o desenvolvimento dos *playbooks* de configurações para os roteadores Cisco IOS.

Inicialmente, foi criada uma máquina virtual na plataforma *Google Cloud* para a instalação do emulador de redes EVE-NG. Dentro do emulador, foi possível adicionar os roteadores para formar a topologia de rede para *Market Data* proposta. A topologia abrange a simulação do *Feed A* e *Feed B* de duas Bolsas de Valores, duas redes MPLS com *Core* separados e uma Corretora recebendo o *stream multicast* dos *Feeds*.

As configurações nos roteadores do *Feed A* foram executadas manualmente, a fim de compreender as etapas e os componentes que envolvem a transmissão e recepção de um *stream multicast* em uma rede *Market Data*. Para o tráfego *multicast* no *backbone* da provedora de serviços, foi configurada uma *Multicast VPN* do tipo MDT-mLDP. Nos roteadores de borda, representados pelo roteadores das Bolsas de Valores e da Corretora, foi configurado o protocolo *PIM Sparse Mode* com o RP auto anunciado. O *stream multicast* foi um sinal de vídeo gerado a partir da ferramenta VLC.

Para a realização das configurações no *Feed B*, visto que são similares às executadas no *Feed A*, foi instalado um servidor central com a ferramenta de automação de redes Ansible. Foram escritos os *playbooks* para todos os roteadores do *Feed B*, sendo utilizado o módulo *cisco.ios*. Após a execução dos *playbooks* de configuração, foi possível receber a transmissão dos quatro *streams multicast* gerados pelas Bolsas de Valores nos dois servidores dispostos na Corretora.

Na validação da comunicação *multicast* do *Feed A* incialmente, percebeu-se que a formação da *MDT default* funcionou de acordo com o esperado, enquanto a *MDT data* ficou oscilando em alguns períodos, sendo constantemente deletada e criada novamente pelos roteadores. O comportamento ficou dessa forma, mesmo reduzindo o limiar para a comutação, não sendo possível identificar a causa da falha. Após aplicar as configurações através do Ansible no *Feed B*, foi necessário reconfigurar os roteadores com RP no *Feed A*, pois os RPs configurados inicialmente abrangiam todo o *range multicast*.

No que tange o uso do Ansible, percebeu-se que é uma opção adequada para administradores que precisam automatizar configurações em roteadores Cisco IOS. O Ansible, mesmo instalado em um *hardware* com baixo desempenho, mostrou-se capaz de rodar um *playbook* de cinco *tasks* em quatro roteadores paralelamente em aproximadamente

23 s no ambiente do emulador. Referente ao emulador, o funcionamento mostrou-se adequado para construção de laboratórios e validação de configurações em redes. Entretanto, foram percebidos alguns atrasos no recebimento do *multicast*, bem como uma alta latência em respostas de simples comandos *ping*.

A linguagem declarativa YAML usada para a construção dos *playbooks* mostrou-se bastante intuitiva e simples de usar, sendo possível escrever um *playbook* com poucas linhas. O módulo *cisco.ios* também atendeu as necessidades exigidas. Porém, pode-se verificar que, em alguns casos, mesmo que uma configuração já estivesse presente no roteador, a saída no terminal do Ansible demonstrou que houve uma mudança na configuração.

Dentro do escopo desse trabalho considera-se, por fim, que a construção do modelo de uma arquitetura de rede para *Market Data* foi desenvolvido com sucesso, sendo possível utilizar a automação do Ansible para gerar as configurações em roteadores Cisco IOS.

Novos trabalhos podem ser realizados visando melhorar a automação da arquitetura de rede proposta, tais como:

- O desenvolvimento de *roles* para uso nos *playbooks*;
- Adicionar novos roteadores na topologia e executar os *playbooks* desenvolvidos, a fim de validar a infraestrutura como código;
- Adicionar roteadores de outros fabricantes na topologia e reescrever os *playbooks* com diferentes módulos;
- Simular casos de falha em roteadores e a reconfiguração através dos *playbooks*;

Além disso, trabalhos futuros podem ser desenvolvidos fazendo a utilização de outro protocolo PIM e outra solução para *Multicast VPN*, a fim de comparar o funcionamento da comunicação *multicast*.

REFERÊNCIAS BIBLIOGRÁFICAS

AIRES, D. M. *MPLS: MULTIPROTOCOL LABEL SWITCHING. RFC3031.* [S.l.], 2004. Disponível em: <http://www.cricte2004.eletrica.ufpr.br/edu/anterior/cd03/trab2/daniel/Trabalho%20de%20MPLS%20-%20RFC3031.htm>. Citado na página 35.

B3. Manual de procedimentos operacionais de negociação da b3. p. 143, 2019. Citado 2 vezes nas páginas 17 e 32.

CABRAL, M.; KRONBAUER, A.; MARTINS, J. Avaliação de Protocolos Multicast. In: . [S.l.: s.n.], 1998. Citado na página 34.

CAMOES, R. J. d. S.; SILVA, J. A. d. DESENVOLVIMENTO E OPERAÇÕES UTILIZANDO ANSIBLE COMO PRINCIPAL FERRAMENTA DE IMPLANTAÇÃO: ESTUDO DE CASO NO TRIBUNAL DE JUSTIÇA DO DISTRITO FEDERAL (TJDFT). *TECNOLOGIAS EM PROJEÇÃO*, v. 9, n. 2, p. 36–52, 2018. ISSN 2178-6267. Number: 2. Disponível em: <http://revista.faculdadeprojecao.edu.br/index.php/Projecao4/article/view/1140>. Citado 2 vezes nas páginas 20 e 21.

CARDOSO, A. F. B. *Automatização de instanciação de serviços de rede numa rede de operador*. Dissertação (Mestrado) — Instituto Superior de Engenharia de Lisboa, jul. 2019. Accepted: 2019-08-30T20:35:10Z. Disponível em: <https://repositorio.ipl.pt/handle/10400.21/10456>. Citado na página 16.

CHANG, C. et al. Automation with Ansible Student Workbook (Role). 2016. Citado 5 vezes nas páginas 25, 26, 27, 29 e 30.

CISCO. *Configuring a Rendezvous Point*. 2002. Disponível em: <https://www.cisco.com/c/en/us/td/docs/ios/solutions_docs/ip_multicast/White_papers/rps.html>. Citado na página 69.

CISCO. *Market Data Network Architecture (MDNA) Overview*. 2008. Disponível em: <https://www.cisco.com/c/dam/en_us/solutions/industries/docs/finance/md-arch-ext.pdf>. Citado 5 vezes nas páginas 17, 32, 33, 40 e 49.

CISCO. *IP Multicast: LSM Configuration Guide - MLDP-Based MVPN*. 2018. Disponível em: <https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/ipmulti_lsm/configuration/xe-16/imc-lsm-xe-16-book/imc-mldp-based-mvpn.html>. Citado 2 vezes nas páginas 37 e 39.

CISCO. *O que é automação de rede?* 2019. Disponível em: <https://www.cisco.com/c/pt_br/solutions/automation/network-automation.html>. Citado na página 16.

CISCO. *Visão geral dos perfis mVPN*. 2020. Disponível em: <https://www.cisco.com/c/pt_br/support/docs/ip/multicast/216493-overview-of-the-mvpn-profiles.html>. Citado 2 vezes nas páginas 37 e 38.

COMER, D. E. *Redes de Computadores e Internet - 6.ed.* [S.l.]: Bookman Editora, 2016. Google-Books-ID: 1nwdDAAAQBAJ. ISBN 978-85-8260-373-4. Citado na página 16.

DUVALL, P. *Automação de infraestrutura*. 2012. Disponível em: <http://www.ibm.com/developerworks/br/library/a-devops2/index.html>. Citado na página 23.

DZERKALS, U. *EVE-NG Professional Cookbook*. 2020. Disponível em: <https://www.eve-ng.net/index.php/documentation/professional-cookbook/>. Citado 2 vezes nas páginas 41 e 46.

EDELMAN, J. *Network Automation with Ansible*. [S.l.]: O'Reilly Media, Inc., 2016. ISBN 978-1-4919-3784-6. Citado na página 28.

FARRUCA, N. M. G. *Wireshark para sistemas distribuídos*. Dissertação (Mestrado) — FCT - UNL, 2009. Accepted: 2009-11-24T10:59:42Z. Disponível em: <https://run.unl.pt/handle/10362/2288>. Citado 2 vezes nas páginas 44 e 45.

FEAMSTER, N.; REXFORD, J.; ZEGURA, E. *The Road to SDN - An intellectual history of programmable networks*. 2013. Disponível em: <https://queue.acm.org/detail.cfm?id=2560327>. Citado na página 16.

FONSECA, R.; SIMOES, A. Alternativas ao XML: YAML e JSON. In: . [s.n.], 2007. ISBN 978-972-99166-4-9. Accepted: 2007-03-14T11:05:08Z. Disponível em: <http://repositorium.sdum.uminho.pt/>. Citado na página 28.

FRANCA, B. W. d. F.; JUNIOR, E. J. d. S.; BRITTO, R. F. Análise de tráfego e simulação de redes multicast. nov. 2012. Accepted: 2014-06-03T23:40:22Z Publisher: Universidade Tecnológica Federal do Paraná. Disponível em: <http://repositorio.roca.utfpr.edu.br:8080/jspui/handle/1/2061>. Citado na página 34.

GEDIA, D.; PERIGO, L. A Centralized Network Management Application for Academia and Small Business Networks. ago. 2018. Citado na página 19.

GIESEN, G. F.; OLIVEIRA, F. C. d. Abordagem SDN: uma mudança de paradigma na arquitetura de rede tradicional. 2017. ISSN 2358-1913. Citado 2 vezes nas páginas 19 e 20.

HEAP, M. *Ansible: From Beginner to Pro*. [S.l.]: Apress, 2016. Google-Books-ID: mZdbjwEACAAJ. ISBN 978-1-4842-1660-6. Citado na página 22.

HOCHSTEIN, L.; MOSER, R. *Ansible: Up and Running: Automating Configuration Management and Deployment the Easy Way*. [S.l.]: "O'Reilly Media, Inc.", 2017. Google-Books-ID: h5YtDwAAQBAJ. ISBN 978-1-4919-7977-8. Citado 2 vezes nas páginas 29 e 31.

HüTTERMANN, M. *DevOps for Developers*. [S.l.]: Apress, 2012. Google-Books-ID: JfUAkB8AA7EC. ISBN 978-1-4302-4570-4. Citado 2 vezes nas páginas 16 e 17.

KIM, H.; FEAMSTER, N. Improving network management with software defined networking. *IEEE Communications Magazine*, v. 51, n. 2, p. 114–119, fev. 2013. ISSN 1558-1896. Conference Name: IEEE Communications Magazine. Citado na página 16.

LINHARES, F. G. d. Avaliação de desempenho de VPNs sobre redes MPLS-Linux. 2010. Accepted: 2011-03-30T05:59:52Z. Disponível em: <https://lume.ufrgs.br/handle/10183/28311>. Citado na página 35.

MARKETS&MARKETS. *Network Automation Market*. [S.l.], 2021. Citado na página 17.

MELO, V. A. Automoção e provisionamento de uma rede campus utilizando Open Networking. jun. 2018. Accepted: 2018-07-19T12:08:41Z. Disponível em: <http://repositorio.uniceub.br/jspui/handle/235/12373>. Citado na página 16.

MENZEN, D. C. Estudo e implementação de soluções para automação de dispositivos de rede. p. 54, 2015. Citado 3 vezes nas páginas 16, 17 e 20.

MORAES, I. M.; DUARTE, O. M. B. D. Uma Comparação entre Protocolos de Roteamento Multicast para Distribuição de TV na Internet. set. 2004. Disponível em: <https://www.gta.ufrj.br/ftp/gta/TechReports/MoDu04.pdf>. Citado na página 33.

NUNES, B. A. A. et al. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys Tutorials*, v. 16, n. 3, p. 1617–1634, 2014. ISSN 1553-877X. Conference Name: IEEE Communications Surveys Tutorials. Citado na página 16.

PEREIRA, T. B. Uma estrategia de roteamento OSPF adaptativo baseado em estimação de banda. 2004. Accepted: 2018-08-04T02:56:17Z Publisher: [s.n.]. Disponível em: <http://repositorio.unicamp.br/jspui/handle/REPOSIP/259284>. Citado na página 36.

PILLA, D. *Automação Ansible, do CLI para uma solução mais Enterprise*. 2020. Disponível em: <https://zallpy.com/ar10022020.html>. Citado na página 26.

RAZA, A. *Puppet vs. Chef vs. Ansible vs. SaltStack*. 2016. Section: Agent Management. Disponível em: <https://jetpatch.com/blog/agent-management/puppet-vs-chef-vs-ansible-vs-saltstack/>. Citado na página 23.

REDHAT. *O que é automação da infraestrutura de TI?* 2018. Disponível em: <https://www.redhat.com/pt-br/topics/automation/whats-it-automation>. Citado na página 21.

SATO, D. *DevOps na prática: Entrega de software confiável e automatizada*. [S.l.]: Editora Casa do Código, 2014. Google-Books-ID: Cm2CCwAAQBAJ. ISBN 978-85-66250-66-4. Citado na página 21.

SCARPATI, J. *What is network automation?* 2017. Disponível em: <https://searchnetworking.techtarget.com/definition/network-automation>. Citado 2 vezes nas páginas 20 e 21.

SOUSA, M. et al. Simuladores e Emuladores de Rede para o Projeto e Solução de Problemas em Ambientes de Produção. *REVISTA DE TECNOLOGIA DA INFORMAÇÃO E COMUNICAÇÃO*, VOL. 6, p. 16, out. 2016. Citado 2 vezes nas páginas 41 e 42.

SYNERGY, R. G. *Switch & Router Revenues Set a New Record; Cisco Market Share Still Over 50% | Synergy Research Group*. 2019. Disponível em: <https://www.srgresearch.com/articles/switch-router-revenues-set-new-record-cisco-market-share-still-over-50>. Citado na página 16.

VENKATESH, P. *Network Automation: Road to Programmable Networks*. 2018. Disponível em: <https://onug.net/blog/network-automation-road-to-programmable-networks/>. Citado 2 vezes nas páginas 19 e 20.

WESTPHAL, R. Estudo e implementação de MPLS/BGP IPv6 VPNs em GNU/Linux. 2011. Accepted: 2012-02-09T01:20:03Z. Disponível em: <https://lume.ufrgs.br/handle/10183/37170>. Citado 2 vezes nas páginas 35 e 36.

YIGAL, A. *Chef vs. Puppet: Methodologies, Concepts, and Support*. 2017. Disponível em: <https://logz.io/blog/chef-vs-puppet/>. Citado 2 vezes nas páginas 23 e 24.

YUNGA, A. Implementación del provisionamiento automático de configuraciones (Network Automation) en infraestructuras multivendor con Ansible. dez. 2018. Accepted: 2019-06-25T16:09:30Z Publisher: Escuela Superior Politécnica de Chimborazo. Disponível em: <http://dspace.espoch.edu.ec/handle/123456789/10935>. Citado 3 vezes nas páginas 16, 20 e 22.

Apêndices

APÊNDICE A – ARQUIVO HOSTS

```
[feed_b]


[feed_b:children]
ce_exchange_feed_b
pe_feed_b
p_core_feed_b

[pe_feed_b]

[pe_feed_b:children]
pe_exchange_feed_b
pe_brokerage_feed_b

[pe_exchange_feed_b]
R61

[pe_brokerage_feed_b]
R71

[p_core_feed_b]
R[11:14]


[ce_exchange_feed_b]
R[81:82]

[ce_brokerage]

[ce_brokerage:children]
ce_brokerage_1

[ce_brokerage_1]
R101
R121
```

APÊNDICE B – DEPLOY_CE_BROKERAGE_ROUTER

```
---

- hosts: "{{ target }}"
  gather_facts: local

  tasks:
     - name: inicial config
       cisco.ios.ios_config:
          lines:
             - hostname {{ inventory_hostname }}
             - no service config
             - ip multicast-routing

     - name: configure Loopback0
       cisco.ios.ios_config:
         lines:
           - ip address {{ loo0 }} 255.255.255.255
           - ip pim sparse-mode
         parents: interface Loopback 0


     - name: configure multicast interfaces IPs
       cisco.ios.ios_config:
         lines:
           - ip address {{ item.ip }} {{ item.mask }}
           - ip pim sparse-mode
           - ip nat {{ item.nat }}
           - ip virtual-reassembly in
           - no shutdown
         parents: '{{ item.interface }}'
       with_items:
       - '{{ interfaces }}'

     - name: configure ebgp routing
```

```
    cisco.ios.ios_config:
      lines:
        - neighbor {{ ebgp_peer }} remote-as 200
      parents: router bgp {{ asn }}

  - name: configure ebgp address-family ipv4
    cisco.ios.ios_config:
      lines:
        - neighbor {{ ebgp_peer }} activate
        - network {{ loo0 }} mask 255.255.255.255
        - network {{ ebgp_network }} mask 255.255.255.0
      parents: address-family ipv4
      before: router bgp {{ asn }}

  - name: configure and NAT
    cisco.ios.ios_config:
      lines:
        - ip nat inside source list NAT interface Ethernet0/1 overload

  - name: configure NAT access list
    cisco.ios.ios_config:
      lines:
        - permit {{ nat_inside }}
      parents: ip access-list standard NAT
```

APÊNDICE C – DEPLOY_CE_EXCHANGE_ROUTER

```
---

- hosts: "{{ target }}"
  gather_facts: local

  tasks:
     - name: inicial config
       cisco.ios.ios_config:
          lines:
             - hostname {{ inventory_hostname }}
             - no service config
             - ip multicast-routing

     - name: configure Loopback0
       cisco.ios.ios_config:
         lines:
           - ip address {{ loo0 }} 255.255.255.255
           - ip pim sparse-mode
         parents: interface Loopback 0


     - name: configure multicast interfaces IPs
       cisco.ios.ios_config:
         lines:
           - ip address {{ item.ip }} {{ item.mask }}
           - ip pim sparse-mode
           - ip nat {{ item.nat }}
           - ip virtual-reassembly in
           - no shutdown
         parents: '{{ item.interface }}'
       with_items:
       - '{{ interfaces }}'

     - name: configure ebgp routing
```

```
    cisco.ios.ios_config:
      lines:
        - neighbor {{ ebgp_peer }} remote-as 200
      parents: router bgp {{ asn }}

  - name: configure ebgp address-family ipv4
    cisco.ios.ios_config:
      lines:
        - neighbor {{ ebgp_peer }} activate
        - network {{ loo0 }} mask 255.255.255.255
        - network {{ ebgp_network }} mask 255.255.255.0
      parents: address-family ipv4
      before: router bgp {{ asn }}

  - name: configure rendezvous point and NAT
    cisco.ios.ios_config:
      lines:
        - ip pim rp-candidate Loopback0 group-list {{ rp_group }}
        - ip nat inside source list NAT interface Ethernet0/1 overload

  - name: configure NAT access list
    cisco.ios.ios_config:
      lines:
        - permit {{ nat_inside }}
      parents: ip access-list standard NAT

  - name: configure RP multicast channel
    cisco.ios.ios_config:
      lines:
        - access-list {{ rp_group }} permit {{ rp_multicast_channel }}
```

APÊNDICE D – DEPLOY_P_CORE_ROUTER

```
---

- hosts: "{{ target }}"
  gather_facts: local

  tasks:
     - name: inicial config
       cisco.ios.ios_config:
          lines:
             - hostname {{ inventory_hostname }}
             - no service config

     - name: configure mpls
       cisco.ios.ios_config:
          lines:
             - no mpls label range {{ mpls_label }}
             - no mpls label protocol ldp
             - no mpls mldp logging notifications

     - name: configure interfaces ip and enable ospf
       cisco.ios.ios_config:
         lines:
           - ip address {{ item.ip }} {{ item.mask }}
           - ip ospf 1 area 0
           - no shutdown
         parents: '{{ item.interface }}'
       with_items:
       - '{{ interfaces }}'

     - name: configure routing ospf
       cisco.ios.ios_config:
         lines:
           - mpls ldp sync
           - mpls ldp autoconfig
```

```
        - router-id {{ loo0 }}
      parents: router ospf 1

  - name: configure mpls router-id
    cisco.ios.ios_config:
      lines:
        - mpls ldp router-id Loopback0
```

APÊNDICE E – DEPLOY_PE_BROKERAGE_ROUTER

```
---
- hosts: "{{ target }}"
  gather_facts: local

  tasks:
     - name: inicial config
       cisco.ios.ios_config:
          lines:
             - hostname {{ inventory_hostname }}
             - no service config

     - name: configure vrf
       cisco.ios.ios_config:
         lines:
           - rd {{ asn }}:{{ item.vrf_id }}
           - vpn id {{ asn }}:{{ item.vrf_id }}
         parents: vrf definition {{ item.vrf_name }}
       with_items:
       - '{{ vrfs }}'

     - name: configure vrf address-family
       cisco.ios.ios_config:
         lines:
           - mdt default mpls mldp {{ root }}
           - route-target both {{ asn }}:{{ item.vrf_id }}
         parents: address-family ipv4
         before: vrf definition {{ item.vrf_name }}
       with_items:
       - '{{ vrfs }}'

     - name: configure multicast in vrf
       cisco.ios.ios_config:
         lines:
            - ip multicast-routing vrf {{ item.vrf_name }}
```

```
    with_items:
      - '{{ vrfs }}'

  - name: configure mpls
    cisco.ios.ios_config:
       lines:
          - mpls label range {{ mpls_label }}
          - mpls label protocol ldp
          - mpls mldp logging notifications

  - name: configure mpls interfaces IPs
    cisco.ios.ios_config:
      lines:
        - ip address {{ item.ip }} {{ item.mask }}
        - ip ospf 1 area 0
        - no shutdown
      parents: '{{ item.interface }}'
    with_items:
    - '{{ interfaces_mpls }}'

  - name: configure multicast interfaces IPs
    cisco.ios.ios_config:
      lines:
        - vrf forwarding {{ item.vrf }}
        - ip address {{ item.ip }} {{ item.mask }}
        - ip pim sparse-mode
        - no shutdown
      parents: '{{ item.interface }}'
    with_items:
    - '{{ interfaces_vrf }}'

  - name: configure ospf routing
    cisco.ios.ios_config:
      lines:
        - mpls ldp sync
        - mpls ldp autoconfig
        - router-id {{ loo0 }}
      parents: router ospf 1

  - name: configure ibgp routing
    cisco.ios.ios_config:
      lines:
```

```
        - no bgp default ipv4-unicast
        - neighbor {{ item.peer }} remote-as {{ asn }}
        - neighbor {{ item.peer }} update-source Loopback0
      parents: router bgp {{ asn }}
    with_items:
    - '{{ ibgp_peers }}'

- name: configure ibgp address-family ipv4
    cisco.ios.ios_config:
      lines:
        - neighbor {{ item.peer }} activate
        - neighbor {{ item.peer }} next-hop-self
      parents: address-family ipv4
      before: router bgp {{ asn }}
    with_items:
    - '{{ ibgp_peers }}'

- name: configure ibgp address-family vpnv4
    cisco.ios.ios_config:
      lines:
        - neighbor {{ item.peer }} activate
        - neighbor {{ item.peer }} send-community both
      parents: address-family vpnv4
      before: router bgp {{ asn }}
    with_items:
    - '{{ ibgp_peers }}'

- name: configure ebgp address-family
    cisco.ios.ios_config:
      lines:
        - neighbor {{ item.peer }} remote-as {{ item.asn }}
        - neighbor {{ item.peer }} activate
      parents: address-family ipv4 vrf {{ item.vrf }}
      before: router bgp {{ asn }}
    with_items:
    - '{{ ebgp_peers }}'

- name: configure mpls router-id
    cisco.ios.ios_config:
      lines:
        - mpls ldp router-id Loopback0
```

APÊNDICE F – DEPLOY_PE_EXCHANGE_ROUTER

```
---
- hosts: "{{ target }}"
  gather_facts: local
  tasks:
     - name: inicial config
       cisco.ios.ios_config:
          lines:
             - hostname {{ inventory_hostname }}
             - no service config

     - name: configure vrf
       cisco.ios.ios_config:
         lines:
           - rd {{ asn }}:{{ item.vrf_id }}
           - vpn id {{ asn }}:{{ item.vrf_id }}
         parents: vrf definition {{ item.vrf_name }}
       with_items:
       - '{{ vrfs }}'

     - name: configure vrf address-family
       cisco.ios.ios_config:
         lines:
           - mdt default mpls mldp {{ root }}
           - mdt data mpls mldp 100
           - mdt data threshold 300
           - route-target both {{ asn }}:{{ item.vrf_id }}
         parents: address-family ipv4
         before: vrf definition {{ item.vrf_name }}
       with_items:
       - '{{ vrfs }}'

     - name: configure multicast in vrf
       cisco.ios.ios_config:
         lines:
```

```
        - ip multicast-routing vrf {{ item.vrf_name }}
    with_items:
      - '{{ vrfs }}'

  - name: configure mpls
    cisco.ios.ios_config:
       lines:
          - mpls label range {{ mpls_label }}
          - mpls label protocol ldp
          - mpls mldp logging notifications

  - name: configure mpls interfaces IPs
    cisco.ios.ios_config:
      lines:
        - ip address {{ item.ip }} {{ item.mask }}
        - ip ospf 1 area 0
        - no shutdown
      parents: '{{ item.interface }}'
    with_items:
    - '{{ interfaces_mpls }}'

  - name: configure multicast interfaces IPs
    cisco.ios.ios_config:
      lines:
        - vrf forwarding {{ item.vrf }}
        - ip address {{ item.ip }} {{ item.mask }}
        - ip pim sparse-mode
        - no shutdown
      parents: '{{ item.interface }}'
    with_items:
    - '{{ interfaces_vrf }}'

  - name: configure ospf routing
    cisco.ios.ios_config:
      lines:
        - mpls ldp sync
        - mpls ldp autoconfig
        - router-id {{ loo0 }}
      parents: router ospf 1

  - name: configure ibgp routing
    cisco.ios.ios_config:
```

```
      lines:
        - no bgp default ipv4-unicast
        - neighbor {{ item.peer }} remote-as {{ asn }}
        - neighbor {{ item.peer }} update-source Loopback0
      parents: router bgp {{ asn }}
    with_items:
    - '{{ ibgp_peers }}'

  - name: configure ibgp address-family ipv4
    cisco.ios.ios_config:
      lines:
        - neighbor {{ item.peer }} activate
        - neighbor {{ item.peer }} next-hop-self
      parents: address-family ipv4
      before: router bgp {{ asn }}
    with_items:
    - '{{ ibgp_peers }}'

  - name: configure ibgp address-family vpnv4
    cisco.ios.ios_config:
      lines:
        - neighbor {{ item.peer }} activate
        - neighbor {{ item.peer }} send-community both
      parents: address-family vpnv4
      before: router bgp {{ asn }}
    with_items:
    - '{{ ibgp_peers }}'

  - name: configure ebgp address-family
    cisco.ios.ios_config:
      lines:
        - neighbor {{ item.peer }} remote-as {{ item.asn }}
        - neighbor {{ item.peer }} activate
      parents: address-family ipv4 vrf {{ item.vrf }}
      before: router bgp {{ asn }}
    with_items:
    - '{{ ebgp_peers }}'
  - name: configure mpls router-id
    cisco.ios.ios_config:
      lines:
        - mpls ldp router-id Loopback0
```

APÊNDICE G – SAÍDA DO PLAYBOOK DE DIFF

```
rlourenco@Linux-Desktop:~/ansible$ ansible-playbook --diff playbooks/diff_
    config.yml --extra-vars="target=all"

PLAY [all]
    ********************************************************************

TASK [check the startup-config against the running-config]
    ********************************************************************
--- before
+++ after
@@ -1,12 +1,28 @@
-hostname R
+hostname R13
+interface Loopback0
+ ip address 13.13.13.13 255.255.255.255
+ ip ospf 1 area 0
  ip address 172.168.1.13 255.255.255.0
 interface Ethernet0/1
+ ip address 10.11.13.13 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/2
+ ip address 10.13.14.13 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/3
+ ip address 10.12.13.13 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/0
+ ip address 10.13.61.13 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router ospf 1
+ mpls ldp sync
```

```
+ mpls ldp autoconfig
+ router-id 13.13.13.13
+mpls ldp router-id Loopback0
  login local
  transport input all
 end
\ No newline at end of file

[WARNING]: [u'connection local support for this module is deprecated and will
    be removed in version 2.14, use connection
ansible.netcommon.network_cli']
changed: [R13]
--- before
+++ after
@@ -1,12 +1,37 @@
-hostname R
+hostname R81
+ip multicast-routing
+interface Loopback0
+ ip address 81.81.81.81 255.255.255.255
+ ip pim sparse-mode
  ip address 172.168.1.81 255.255.255.0
 interface Ethernet0/1
+ ip address 10.61.81.81 255.255.255.0
+ ip pim sparse-mode
+ ip nat outside
+ ip virtual-reassembly in
 interface Ethernet0/2
+ ip address 192.168.2.1 255.255.255.0
+ ip pim sparse-mode
+ ip nat inside
+ ip virtual-reassembly in
 interface Ethernet0/3
 interface Ethernet1/0
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router bgp 65001
+ bgp log-neighbor-changes
+ neighbor 10.61.81.61 remote-as 200
+ address-family ipv4
+ network 10.61.81.0 mask 255.255.255.0
```

```
+ network 81.81.81.81 mask 255.255.255.255
+ neighbor 10.61.81.61 activate
+ exit-address-family
+ip pim rp-candidate Loopback0 group-list 3
+ip nat inside source list NAT interface Ethernet0/1 overload
+ip access-list standard NAT
+ permit 192.168.2.0 0.0.0.255
+access-list 3 permit 239.1.2.0 0.0.0.255
  login local
  transport input all
 end
\ No newline at end of file

changed: [R81]
--- before
+++ after
@@ -1,12 +1,28 @@
-hostname R
+hostname R14
+interface Loopback0
+ ip address 14.14.14.14 255.255.255.255
+ ip ospf 1 area 0
  ip address 172.168.1.14 255.255.255.0
 interface Ethernet0/1
+ ip address 10.12.14.14 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/2
+ ip address 10.13.14.14 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/3
+ ip address 10.11.14.14 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/0
+ ip address 10.14.71.14 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router ospf 1
+ mpls ldp sync
+ mpls ldp autoconfig
+ router-id 14.14.14.14
```

```
+mpls ldp router-id Loopback0
  login local
  transport input all
 end
\ No newline at end of file

changed: [R14]
--- before
+++ after
@@ -1,12 +1,28 @@
-hostname R
+hostname R11
+interface Loopback0
+ ip address 11.11.11.11 255.255.255.255
+ ip ospf 1 area 0
  ip address 172.168.1.11 255.255.255.0
 interface Ethernet0/1
+ ip address 10.11.13.11 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/2
+ ip address 10.11.12.11 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/3
+ ip address 10.11.14.11 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/0
+ ip address 10.11.61.11 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router ospf 1
+ mpls ldp sync
+ mpls ldp autoconfig
+ router-id 11.11.11.11
+mpls ldp router-id Loopback0
  login local
  transport input all
 end
\ No newline at end of file

changed: [R11]
```

```
--- before
+++ after
@@ -1,12 +1,28 @@
-hostname R
+hostname R12
+interface Loopback0
+ ip address 12.12.12.12 255.255.255.255
+ ip ospf 1 area 0
  ip address 172.168.1.12 255.255.255.0
 interface Ethernet0/1
+ ip address 10.12.14.12 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/2
+ ip address 10.11.12.12 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/3
+ ip address 10.12.13.12 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/0
+ ip address 10.12.71.12 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router ospf 1
+ mpls ldp sync
+ mpls ldp autoconfig
+ router-id 12.12.12.12
+mpls ldp router-id Loopback0
  login local
  transport input all
 end
\ No newline at end of file

changed: [R12]
--- before
+++ after
@@ -10,6 +10,10 @@
  ip nat outside
  ip virtual-reassembly in
 interface Ethernet0/2
+ ip address 10.71.101.101 255.255.255.0
```

```
+ ip pim sparse-mode
+ ip nat outside
+ ip virtual-reassembly in
 interface Ethernet0/3
  ip address 172.16.1.1 255.255.255.0
  ip pim sparse-mode
@@ -23,10 +27,13 @@
 router bgp 65101
  bgp log-neighbor-changes
  neighbor 10.31.101.31 remote-as 100
+ neighbor 10.71.101.71 remote-as 200
  address-family ipv4
   network 10.31.101.0 mask 255.255.255.0
+ network 10.71.101.0 mask 255.255.255.0
   network 101.101.101.101 mask 255.255.255.255
   neighbor 10.31.101.31 activate
+ neighbor 10.71.101.71 activate
  exit-address-family
 ip nat inside source list NAT interface Ethernet0/1 overload
 ip access-list standard NAT

changed: [R101]
--- before
+++ after
@@ -1,12 +1,37 @@
-hostname R
+hostname R82
+ip multicast-routing
+interface Loopback0
+ ip address 82.82.82.82 255.255.255.255
+ ip pim sparse-mode
  ip address 172.168.1.82 255.255.255.0
 interface Ethernet0/1
+ ip address 10.61.82.82 255.255.255.0
+ ip pim sparse-mode
+ ip nat outside
+ ip virtual-reassembly in
 interface Ethernet0/2
+ ip address 192.168.2.1 255.255.255.0
+ ip pim sparse-mode
+ ip nat inside
+ ip virtual-reassembly in
```

```
 interface Ethernet0/3
 interface Ethernet1/0
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router bgp 65002
+ bgp log-neighbor-changes
+ neighbor 10.61.82.61 remote-as 200
+ address-family ipv4
+ network 10.61.82.0 mask 255.255.255.0
+ network 82.82.82.82 mask 255.255.255.255
+ neighbor 10.61.82.61 activate
+ exit-address-family
+ip pim rp-candidate Loopback0 group-list 4
+ip nat inside source list NAT interface Ethernet0/1 overload
+ip access-list standard NAT
+ permit 192.168.2.0 0.0.0.255
+access-list 4 permit 239.2.2.0 0.0.0.255
  login local
  transport input all
 end
\ No newline at end of file

changed: [R82]
--- before
+++ after
@@ -1,12 +1,75 @@
-hostname R
+hostname R61
+vrf definition Exchange1_FeedB
+ rd 200:1
+ vpn id 200:1
+ address-family ipv4
+ mdt default mpls mldp 12.12.12.12
+ mdt data mpls mldp 100
+ mdt data threshold 300
+ route-target export 200:1
+ route-target import 200:1
+ exit-address-family
+vrf definition Exchange2_FeedB
+ rd 200:2
+ vpn id 200:2
```

```
+ address-family ipv4
+ mdt default mpls mldp 12.12.12.12
+ mdt data mpls mldp 100
+ mdt data threshold 300
+ route-target export 200:2
+ route-target import 200:2
+ exit-address-family
+ip multicast-routing vrf Exchange1_FeedB
+ip multicast-routing vrf Exchange2_FeedB
+mpls label range 6100 6199
+mpls label protocol ldp
+mpls mldp logging notifications
+interface Loopback0
+ ip address 61.61.61.61 255.255.255.255
+ ip ospf 1 area 0
  ip address 172.168.1.61 255.255.255.0
 interface Ethernet0/1
+ ip address 10.11.61.61 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/2
+ ip address 10.13.61.61 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/3
+ vrf forwarding Exchange1_FeedB
+ ip address 10.61.81.61 255.255.255.0
+ ip pim sparse-mode
 interface Ethernet1/0
+ vrf forwarding Exchange2_FeedB
+ ip address 10.61.82.61 255.255.255.0
+ ip pim sparse-mode
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router ospf 1
+ mpls ldp sync
+ mpls ldp autoconfig
+ router-id 61.61.61.61
+router bgp 200
+ bgp log-neighbor-changes
+ neighbor 71.71.71.71 remote-as 200
+ neighbor 71.71.71.71 update-source Loopback0
+ address-family ipv4
```

```
+ neighbor 71.71.71.71 activate
+ neighbor 71.71.71.71 next-hop-self
+ exit-address-family
+ address-family vpnv4
+ neighbor 71.71.71.71 activate
+ neighbor 71.71.71.71 send-community both
+ exit-address-family
+ address-family ipv4 vrf Exchange1_FeedB
+ neighbor 10.61.81.81 remote-as 65001
+ neighbor 10.61.81.81 activate
+ exit-address-family
+ address-family ipv4 vrf Exchange2_FeedB
+ neighbor 10.61.82.82 remote-as 65002
+ neighbor 10.61.82.82 activate
+ exit-address-family
+mpls ldp router-id Loopback0
  login local
  transport input all
 end
\ No newline at end of file

changed: [R61]
ok: [R121]
--- before
+++ after
@@ -1,12 +1,71 @@
-hostname R
+hostname R71
+vrf definition Exchange1_FeedB
+ rd 200:1
+ vpn id 200:1
+ address-family ipv4
+ mdt default mpls mldp 12.12.12.12
+ route-target export 200:1
+ route-target import 200:1
+ exit-address-family
+vrf definition Exchange2_FeedB
+ rd 200:2
+ vpn id 200:2
+ address-family ipv4
+ mdt default mpls mldp 12.12.12.12
+ route-target export 200:2
```

```
+ route-target import 200:2
+ exit-address-family
+ip multicast-routing vrf Exchange1_FeedB
+ip multicast-routing vrf Exchange2_FeedB
+mpls label range 7100 7199
+mpls label protocol ldp
+mpls mldp logging notifications
+interface Loopback0
+ ip address 71.71.71.71 255.255.255.255
+ ip ospf 1 area 0
  ip address 172.168.1.71 255.255.255.0
 interface Ethernet0/1
+ ip address 10.12.71.71 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/2
+ ip address 10.14.71.71 255.255.255.0
+ ip ospf 1 area 0
 interface Ethernet0/3
+ vrf forwarding Exchange2_FeedB
+ ip address 10.71.101.71 255.255.255.0
+ ip pim sparse-mode
 interface Ethernet1/0
+ vrf forwarding Exchange1_FeedB
+ ip address 10.71.121.71 255.255.255.0
+ ip pim sparse-mode
 interface Ethernet1/1
 interface Ethernet1/2
 interface Ethernet1/3
+router ospf 1
+ mpls ldp sync
+ mpls ldp autoconfig
+ router-id 71.71.71.71
+router bgp 200
+ bgp log-neighbor-changes
+ neighbor 61.61.61.61 remote-as 200
+ neighbor 61.61.61.61 update-source Loopback0
+ address-family ipv4
+ neighbor 61.61.61.61 activate
+ neighbor 61.61.61.61 next-hop-self
+ exit-address-family
+ address-family vpnv4
+ neighbor 61.61.61.61 activate
```

```
+ neighbor 61.61.61.61 send-community both
+ exit-address-family
+ address-family ipv4 vrf Exchange1_FeedB
+ neighbor 10.71.121.121 remote-as 65101
+ neighbor 10.71.121.121 activate
+ exit-address-family
+ address-family ipv4 vrf Exchange2_FeedB
+ neighbor 10.71.101.101 remote-as 65101
+ neighbor 10.71.101.101 activate
+ exit-address-family
+mpls ldp router-id Loopback0
  login local
  transport input all
 end
\ No newline at end of file

changed: [R71]

PLAY RECAP
    ********************************************************************
R101 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R11 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R12 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R121 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R13 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R14 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R61 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R71 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R81 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
R82 : ok=1 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```