Instituto Federal da Paraíba

Campus João Pessoa

Programa de Pós-Graduação em Tecnologia da Informação

Nível Mestrado Profissional

AÉCIO DOS SANTOS PIRES

SOLUÇÃO INTEGRADA PARA CONFIGURAÇÃO AUTOMATIZADA DE ATIVOS DE REDE

DISSERTAÇÃO DE MESTRADO

JOÃO PESSOA

2021

Aécio dos Santos Pires

SOLUÇÃO INTEGRADA PARA CONFIGURAÇÃO AUTOMATIZADA DE ATIVOS DE REDE

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Tecnologia da Informação, pelo Programa de Pós-Graduação em Tecnologia da Informação do Instituto Federal da Paraíba – IFPB.

Orientador: Prof. Dr. Paulo Ditarso Maciel Jr.

Coorientador: Prof. Dr. Diego E. Rosa Pessoa

João Pessoa

2021

Dados Internacionais de Catalogação na Publicação – CIP
Biblioteca Nilo Peçanha – IFPB, *campus* João Pessoa

P667s Pires, Aécio dos Santos.
Solução integrada para configuração automatizada de ativos de rede / Aécio dos Santos Pires. – 2021.
79 f. : il.

Dissertação (Mestrado em Tecnologia da Informação) – Instituto Federal da Paraíba – IFPB / Programa de Pós-Graduação em Tecnologia da Informação.
Orientador: Prof. Dr. Paulo Ditarso Maciel Jr.
Coorientador: Prof. Dr. Diego E. Rosa Pessoa.

1. Tecnologia da Informação. 2. Ativos de rede. 3. Infraestrutura como código. 4. Automação. I. Título.

CDU 004

Bibliotecária responsável Josinete Nóbrega de Araújo – CRB15/116

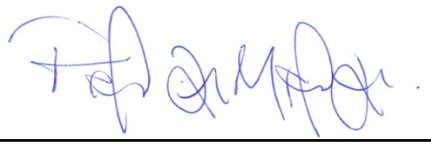
Aécio dos Santos Pires

SOLUÇÃO INTEGRADA PARA CONFIGURAÇÃO AUTOMATIZADA DE ATIVOS DE REDE

Dissertação apresentada como requisito par- cial para obtenção do título de Mestre em Tecnologia da Informação, pelo Programa de Pós-Graduação em Tecnologia da Informação do Instituto Federal da Paraíba – IFPB.

Aprovado em 26 de Fevereiro de 2021.

BANCA EXAMINADORA:



Prof. Dr. Paulo Ditarso Maciel Jr.
Orientador e Avaliador Interno

Prof. Dr. Diego E. Rosa Pessoa
Coorientador e Avaliador Interno

Prof. Dr. Dênio Mariz Timoteo de Sousa – IFPB
Avaliador Interno

Prof. Dr. Thiago Gouveia da Silva – IFPB
Avaliador Interno

Prof. Dr. Augusto José Venâncio Neto – UFRN
Avaliador Externo

Visto e permitida a impressão
João Pessoa

Prof. Dr. Francisco Petronio A. de Medeiros
Coordenador PPGTI

Este trabalho é dedicado a todos os profissionais
da área de Tecnologia da Informação, especialmente aos que
gostam de compartilhar o que aprendem.

AGRADECIMENTOS

Agradeço a Deus pela vida, misericórdia, saúde, força, bençãos e oportunidades que me permitiram adquirir conhecimento para chegar até este momento.

Obrigado Janaína Militão[1] pela enorme paciência, compreensão e apoio em cada momento.

Agradeço aos professores Lafayette Batista[2], Luciana Oliveira[3] e Denio Mariz[4] pelo compartilhamento do conhecimento dentro e fora da sala de aula.

Obrigado professor Maxwell Amaral[5] pelo apoio durante a fase de elaboração da patente e registro de *software*.

Agradeço, especialmente, aos professores Paulo Ditarso[6] e Diego Pessoa[7] pela paciência e orientação durante o ano de 2020. Vocês melhoraram muito a qualidade da pesquisa e sem esse apoio ela teria sido interrompida e, consequentemente, não teria chegado até este momento.

Obrigado Francisco Petrônio[8] pela descomunal energia, atenção, paciência e agilidade nas atividades de coordenação do mestrado, bem como pelo compartilhamento do conhecimento.

Agradeço a Luciano Alves[9], Tatiane Rosa[10] e Cezar Serafini[11], ex-colegas da Unirede, pela ajuda na identificação do problema alvo desta pesquisa.

Thank you Mircea Ulinic[12] for contributing to the development of the SaltStack, Salt SProxy and Napalm. Thanks for your attention in answering my questions throughout this research.

Thank you Tomas Kirnak[13] for your attention in answering my questions and sharing access to Unimus usage licenses. It was essential to carry out one of the tests during the research.

Por fim, obrigado aos funcionários do IFPB, colegas e amigos que, direta ou indiretamente, providenciaram a ajuda necessária durante a execução das atividades do mestrado.

---

1 Janaína Militão: https://www.linkedin.com/in/janaina-militao
2 Lafayette Batista: http://lattes.cnpq.br/2144574905285987
3 Luciana Oliveira: http://lattes.cnpq.br/4044015211881197
4 Denio Mariz: http://lattes.cnpq.br/0353331129592565
5 Maxwell Amaral: http://lattes.cnpq.br/7725418498659982
6 Paulo Ditarso: http://lattes.cnpq.br/1101383196957378
7 Diego Pessoa: http://lattes.cnpq.br/4442363726982518
8 Francisco Petrônio: http://lattes.cnpq.br/9716270626654261
9 Luciano Alves: https://br.linkedin.com/in/luciano-alves-unirede
10 Tatiane Rosa: https://br.linkedin.com/in/tatianerosa
11 Cezar Serafini: https://br.linkedin.com/in/cezar-olavo-russo-serafini-911b191
12 Mircea Ulinic: https://www.linkedin.com/in/mirceaulinic
13 Tomas Kirnak: https://www.linkedin.com/in/tomas-kirnak-9b015957/

RESUMO

O gerenciamento manual de ativos de rede está suscetível a problemas incorridos pelas pessoas que administram tais recursos. Além disso, proporciona um ambiente de trabalho onde certas particularidades podem acontecer, como: erros de configuração, falta de padronização, grande quantidade de trabalho repetitivo e pouca ou nenhuma rastreabilidade de alterações ao longo do tempo. Com o intuito de mitigar esses problemas, a abordagem *Infrastructure as Code* (IaC) automatiza o processo de configuração de recursos como sistemas operacionais de servidores, serviços de rede, contêineres e aplicações. Neste contexto, ferramentas de gerenciamento de configuração permitem que a infraestrutura de rede seja tratada como *software* e possibilitam a padronização e reversão da configuração, bem como a redução do trabalho manual. Com isso, a gerência de recursos de rede pode se beneficiar de técnicas de desenvolvimento como o versionamento e reuso de código e a aplicação de testes unitários. Como exemplos de tais ferramentas de gerenciamento de configuração pode-se citar: *Puppet*, *SaltStack*, *Ansible* e *Chef*. Apesar de serem utilizadas geralmente no contexto de provisionamento de sistemas operacionais e serviços, essas ferramentas também podem ser usadas em ambientes produtivos para substituir procedimentos manuais nas atividades de gerenciamento da configuração de ativos de rede (*switches* e roteadores). Contudo, a possibilidade de coexistência de ativos heterogêneos, nos quais os diferentes fabricantes implementam comandos e sintaxes divergentes para realizar atividades semelhantes, limita o campo de atuação dessas ferramentas. O objetivo desta pesquisa é propor uma solução integrada a partir de ferramentas de *softwares* diferentes e que utilizam a abordagem IaC, com o intuito de realizar a automação da configuração de ativos de rede, levando em consideração diferentes modelos e fabricantes. Além disso, busca-se investigar a solução proposta em termos de sua eficiência realizando uma avaliação quantitativa. Para tal, são realizados experimentos a partir de casos de uso elaborados com a finalidade de demonstrar características importantes da solução proposta.

Palavras-chaves: ativos de rede, automação, versionamento, infraestrutura como código.

ABSTRACT

The manual management of network assets is susceptible to problems incurred by the people who manage such resources. In addition, it provides a work environment where certain peculiarities can happen, such as: configuration errors, lack of standardization, a large amount of repetitive work, and little or no traceability of changes over time. In order to mitigate theses problems the approach Infrastructure as Code (IaC) automate the process of configuring resources such as operating systems, network services, containers, and applications. In this context, configuration management tools allow the network infrastructure to be treated as software and enable the standardization and reversal of the configuration, as well as the reduction of manual work. The management of network resources can benefit from development techniques likewise versioning, reuse of code, and application of unit testing. Examples of such configuration management tools include Puppet, SaltStack, Ansible, and Chef. Although they are generally used in the context of provisioning operating systems and services, these tools can also be used in production environments to replace manual procedures in the activities of managing the configuration of network assets (switches and routers). However, the possibility of heterogeneous assets coexistence, in which different manufacturers implement divergent commands and syntaxes to carry out similar activities, limits the scope of such tools. The objective of this research is to propose an integrated solution using different software tools that use the IaC approach in order to automate the configuration of network assets, taking into account different models and manufacturers. In addition, we seek to investigate the proposed solution in terms of its efficiency by performing a quantitative assessment. To this end, experiments are carried out with use cases designed to demonstrate important characteristics of the proposed solution.

Key-words: network devices, automation, versioning, infrastructure as code.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AE	Análise de Escalabilidade
AP	Análise de Perfilamento
ACL	*Access Control List*
ACM	*Association for Computing Machinery*
AES	*Advanced Encryption Standard*
API	*Application Programming Interface*
AWS	*Amazon Web Services*
CA	*Certification Authority*
CE	Critério de Exclusão
CI	*Continuos Integration*
CD	*Continuos Delivery*
CD	*Continuos Deploy*
CLI	*Command-line Interface*
CPU	*Central Process Unit*
CVS	*Concurrent Versions System*
CWE	*Common Weakness Enumeration*
DNS	*Domain Name System*
DSL	*Domain-specific Language*
EC2	*Elastic Compute Cloud*
FP	Faceta de Pesquisa
FT	Faceta Tecnológica
GB	*Gigabyte*
GCM	*Galois Counter Mode*
GCP	*Google Cloud Platform*
GNS3	*Graphical Network Simulator 3*
GNU	*GNU's Not Unix*

GPG	*GNU Privacy Guard*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hyper Text Transfer Protocol Secure*
IaC	*Infrastructure as Code*
IEEE	*Institute of Electrical and Electronics Engineers*
IFPB	Instituto Federal da Paraíba
INPI	Instituto Nacional da Propriedade Industrial
IOF	Imposto sobre Operações Financeiras
IoT	*Internet of Things*
IP	*Internet Protocol*
IIQ	Intervalo Interquartil
JSON	*JavaScript Object Notation*
KMS	*Key Management Service*
LAN	*Local Area Network*
LXD	*Linux Container Daemon*
MAN	*Metropolitan Area Network*
MIC	Módulo de Integração Contínua
MGC	Módulo de Gerência de Configuração
MN	Módulo de Notificações
MV	Módulo de Versionamento
NaC	*Network as Code*
NAM	*Network Automation Manager*
NCaaS	*Network Configuration as a Service*
NCM	*Network Configuration Manager*
NAPALM	*Network Automation and Programmability Abstraction Layer with Multivendor support*
NETCONF	*Network Configuration Protocol*
NTP	*Network Time Protocol*
PDF	*Portable Document Format*
PGP	*Pretty Good Privacy*

PoC	*Proof of Concept*
PPGTI	Programa de Pós-Graduação em Tecnologia da Informação
RAM	*Random Access Memory*
REST	*Representational state transfer*
SDN	*Software Defined Networking*
SGBD	*Sistemas de Gerenciamento de Banco de Dados*
SMTP	*Simple Mail Transfer Protocol*
SOPS	*Secrets Operations*
SSH	*Secure Shell*
SVN	*Apache Subversion*
TCP	*Transmission Control Protocol*
TI	Tecnologia da Informação
TLS	*Transport Layer Security*
VPN	*Virtual Private Network*
WAN	*Wide Area Network*
XML	*Extensible Markup Language*
YAML	*YAML Ain't Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta uma visão geral e os desafios inerentes à gerência de configuração de ativos de rede heterogêneos, quando realizada de forma manual. Também é mostrada a necessidade de utilizar a abordagem de infraestrutura como código para automatizar a gerência de ativos em larga escala. Apesar da existência de várias ferramentas que implementam a infraestrutura como código, nem todas oferecem o suporte a ativos de rede de diferentes modelos e fabricantes. Neste trabalho é proposta uma solução integrada a partir de *softwares* existentes para gerenciar a configuração de ativos de rede heterogêneos em larga escala, utilizando a abordagem de infraestrutura como código. A primeira seção apresenta uma visão geral sobre a gerência de configuração de ativos de rede, os desafios enfrentados pelos times de administradores de rede, a solução proposta para auxiliar nesses desafios, bem como as principais contribuições deste trabalho. Nas seções seguintes são apresentados os objetivos do trabalho e, por fim, a estrutura deste documento.

1.1 Motivação e Descrição do Problema

A Tecnologia da Informação (TI) é considerada atualmente uma ferramenta com um papel importante para impulsionar o sucesso corporativo. Um provisionamento bem planejado e uma gerência eficiente dos recursos de TI podem contribuir para alcançar vantagens competitivas em termos de negócios (CHRISTOPHER, 2019). Portanto, a relevância das ações do departamento de TI cresce com o aumento das decisões estratégicas providas pelo mesmo, tornando o gerenciamento eficiente uma tarefa fundamental. Isso se reflete cada vez mais no desempenho de uma empresa, tendo em vista que "[...] há um alinhamento da TI com a missão da organização, objetivos estratégicos e resultados esperados" (MOURA; SAUVÉ; BARTOLINI, 2008).

Como consequência, as operações de TI estão cada vez mais imbuídas de técnicas ágeis e sucintas que buscam encurtar o ciclo de desenvolvimento de um *software* ou unificar seus procedimentos com as atividades de desenvolvimento. Essa tendência de usar recursos da engenharia de *software* para reduzir o espaço, tempo e esforço entre atividades de desenvolvimento e operações de *software*, bem como a distância técnica e organizacional entre os dois tipos de equipes responsáveis é conhecida como cultura *DevOps* (ARTA et al., 2017). Como parte de seus princípios, muitas práticas envolvem a reutilização de padrões e ferramentas de *software*, tais como o versionamento e a revisão de código.

Uma parte importante da gerência de TI envolve a configuração de ativos de rede tais como: *switches*, roteadores, *firewalls* e pontos de acesso. Propriedades como a conectividade, o desempenho, a eficiência de custo, a confiabilidade e a segurança de uma rede dependem do comportamento dos ativos definidos por suas respectivas configurações. Estas são geradas

inicialmente quando a rede é construída e são atualizadas frequentemente devido à ocorrência de incidentes ou à incorporação de novos requisitos de aplicativos e serviços (LIU et al., 2018). O gerenciamento do ciclo de vida destas configurações é notoriamente desafiador para os administradores de rede, pois envolve atividades que vão desde a geração e atualização de configurações, passando pelo diagnóstico de problemas e transição de estados. Além disso, essas tarefas são muitas vezes realizadas de forma manual, por meio da execução de comandos diretamente nos ativos.

De acordo com um estudo encomendado pela Cisco em 2016 (MCKINSEY, 2016 apud SHAH; DUBARIA, 2019), cerca de 95% das alterações na configuração de ativos de rede eram realizadas manualmente, 70% das violações de políticas foram ocasionadas por erros humanos e 75% dos custos de operação eram gastos com alterações nas configurações e solução de problemas (do inglês *troubleshooting*). Além disso, cerca de US$ 60 bilhões foram gastos com mão de obra e aquisição de ferramentas de operações de rede naquele ano. Outro ponto interessante é que o tempo médio para resolver um problema de rede após a notificação foi de 1 a 5 horas. Por fim, ainda segundo o estudo, 74% das operadoras relataram alterações na rede que impactaram significativamente em seus negócios, das quais 97% admitiram que isso ocorreu a partir de erros humanos que causaram interrupções na rede e 22% das interrupções não planejadas foram causadas por tais erros.

Uma pesquisa publicada em 2020 por GARROS e CAEN (2019) compara dados coletados em 2016 e 2019 e conta com a participação de 293 profissionais de 6 continentes. Os resultados da pesquisa mostram que 70% dos participantes em 2019 utilizaram alguma ferramenta para automatizar a gerência de configuração e esse número era 64% em 2016. A pesquisa mostra ainda que 67% dos entrevistados em 2019 utilizaram alguma ferramenta para implantar a configuração automatizada em ativos e em 2016 esse número foi de 59%. Quando perguntados se era permitido fazer alterações manuais na configuração dos ativos em adição ao processo automatizado, 85,5% dos entrevistados em 2019 disseram que sim (dado não coletado em 2016).

A partir dos resultados destas pesquisas, é possível inferir que o problema da gerência de configuração de ativos continua relevante, representando um dos fatores de impacto para o negócio de várias empresas e organizações ao redor do mundo. Essas pesquisas evidenciam a importância do trabalho proposto nesta pesquisa. Alterações em configurações de redes devem ser realizadas de maneira planejada, principalmente quando a infraestrutura é larga e heterogênea. Quando executadas de forma manual, as atividades de atualização e/ou reversão da configurações de rede ficam sujeitas à erros humanos, trazendo riscos de indisponibilidade da rede e dificuldades da manutenção do rastreamento dos efeitos destas configurações em ativos diferentes.

Outro fator complicador está associado à linguagem de configuração de um ativo, cuja sintaxe de comandos varia entre os diferentes modelos e fornecedores. Logo, os administradores precisam ter experiência na manipulação de comandos em diferentes modelos de ativos e na aplicação de alterações que podem afetar a rede inteira. Uma alternativa viável é o uso de ferramentas

de automação de configuração, muito embora essa abordagem possa significar uma mudança no paradigma de trabalho da equipe que gerencia a infraestrutura de rede, pois a utilização dessas ferramentas envolve a manipulação da configuração como código e a utilização de técnicas e abordagens de programação, tais como: reuso de código, versionamento, parametrização, lógica e algoritmo, linguagem de programação, integração contínua, entre outras que há alguns anos eram mais relacionadas ao processo de desenvolvimento de *software*.

Dentro do contexto da cultura DevOps, a gerência de Infraestrutura como Código (IaC, do inglês *Infrastructure as Code*) é uma abordagem que permite a automação de uma infraestrutura utilizando práticas de desenvolvimento de *software*, enfatizando rotinas consistentes e repetitivas para orquestração, provisionamento e configuração (MORRIS, 2016). Por exemplo, em vez de configurar manualmente uma máquina virtual, na qual um sistema deve ser implantado com as versões corretas de todas as dependências de *software*, bastaria especificar os requisitos uma vez, em formato de código, após o qual as ferramentas aplicariam automaticamente essa especificação para gerar a imagem adequada. Além da automação, o fato de o ambiente ter sido especificado em formato de código significa explicitamente que o mesmo poderá ser implantado em qualquer lugar, minimizando a possibilidade de inconsistências (JIANG; ADAMS, 2015).

Contudo, a decisão de usar ferramentas que implementam IaC não representa muitas vezes uma transição simples e direta. Adicionalmente, pode haver resistência da equipe responsável quando é necessário se adaptar em um curto prazo. Como exemplo de uma dificuldade inerente, a curva de aprendizado da linguagem de programação de uma determinada ferramenta nem sempre é suave. É importante enfatizar que cada ferramenta geralmente utiliza uma Linguagem de Domínio Específico (DSL, do inglês *Domain Specific Language*) própria e, considerando infraestruturas de redes com ativos heterogêneos em que são necessárias várias delas, o aprendizado de mais de uma linguagem é inevitável. Portanto, a heterogeneidade do ambiente que contém elementos de diferentes fornecedores, versões de objetos e/ou sistemas operacionais, implica em vários dados, métodos e sintaxes de configuração distintas (ISMAIL; HAMZA; MOHANED, 2018). Além disso, poucas ferramentas conseguem gerenciar ativos de rede heterogêneos.

Levando em consideração as dificuldades expostas, nota-se que uma solução eficiente para o gerenciamento de configurações automatizadas de ativos de rede é altamente desejável. Para tal, esta pesquisa propõe a integração de ferramentas existentes para a gerência de ativos de redes, utilizando uma abordagem de Infraestrutura como Código. A solução proposta centraliza a automação das atividades de configuração e permite realizar o *backup* de todo o processo. Além disso, a mesma busca abstrair a diversidade de sintaxes para diferentes ativos, bem como possibilita o gerenciamento de uma grande quantidade de dispositivos. Em resumo, a solução proposta pode ser vista como um *pipeline* escrito como código, consiste em um conjunto de ferramentas, fluxos e processos automatizados que integra todos os componentes de *software* envolvidos (INDIA, 2019). Faz parte desta pesquisa a concepção da arquitetura proposta e dos componentes envolvidos, bem como a implementação da solução integrada e do

fluxo de execução das etapas do *pipeline*. Além disso, propõe-se uma investigação baseada em experimentação para validar a solução por meio de um ambiente de testes controlado.

Como uma solução baseada em *software*, duas propriedades importantes podem ser utilizadas como critérios de sucesso, são elas: ***eficiência*** e ***eficácia***. Por mais que possam significar diferentes aspectos dependendo da situação, essas duas propriedades são definidas no contexto deste trabalho como descrito a seguir. A eficiência da solução proposta será caracterizada como a capacidade de executar as tarefas com desempenho apropriado, em termos de heterogeneidade (diversidade de ativos) e escalabilidade (número de ativos) do ambiente. Por sua vez, a eficácia será caracterizada pela qualidade do *software* produzido, ou seja, a "corretude" do mesmo em relação ao que se propõe a fazer. Logo, a solução será considerada bem sucedida se o código produzido tiver pouca ou nenhuma falha e o mesmo for capaz de automatizar eficientemente a configuração de uma grande quantidade de ativos variados.

1.2 Objetivos

A seguir são apresentados os objetivos geral e específicos, que norteiam a pesquisa.

1.2.1 Objetivo Geral

O objetivo desta pesquisa é melhorar os procedimentos de gerência de configuração de ativos de rede heterogêneos a partir de uma solução integrada de *software* utilizando a abordagem Infraestrutura como Código, o controle de versão dos arquivos de credenciais e configuração dos ativos. Levando em consideração a necessidade de utilizar diferentes ferramentas para atender a esse objetivo foi elaborada a seguinte questão de pesquisa:

> *"Como integrar, de maneira eficiente e eficaz, diferentes softwares para realizar a automação da configuração de uma infraestrutura de ativos de rede?"*

Esta questão norteia toda a pesquisa desde a identificação de trabalhos relacionados à validação da solução proposta.

1.2.2 Objetivos Específicos

- Identificar trabalhos relacionados à gerência da configuração de ativos de rede que utilizem a abordagem de Infraestrutura como Código;
- Propor uma arquitetura de *software* integrado para gerenciar de maneira eficiente e eficaz a configuração de ativos de rede de diferentes modelos e fabricantes;
- Implementar uma prova de conceito da arquitetura proposta, utilizando *switches* e roteadores virtuais e simulados de diferentes modelos e fabricantes;

- Avaliar a utilização da solução proposta através de uma metodologia de experimentação;
- Reportar os resultados obtidos na pesquisa não apenas na dissertação, mas também através de artigos científicos.

1.3 Estrutura do Documento

Esta dissertação compreende mais cinco capítulos subsequentes a este, que estão organizados da maneira descrita a seguir.

O **Capítulo 2** apresenta uma fundamentação teórica sobre Infraestrutura como Código, versionamento, cultura DevOps e outros temas que estão diretamente relacionados a solução proposta nesta pesquisa. Apresenta-se também uma seção de conceitualização das ferramentas utilizadas ao longo do trabalho para facilitar o entendimento da proposta.

O **Capítulo 3** apresenta os detalhes do protocolo de revisão sistemática utilizado na busca por trabalhos acadêmicos relacionados a esta pesquisa. Esses trabalhos são descritos e categorizados como *faceta de pesquisa* ou *faceta tecnológica*.

O **Capítulo 4** descreve a solução proposta para gerenciar a configuração de ativos de rede de diferentes modelos e fabricantes, utilizando para isto a abordagem de infraestrutura como código. Também são apresentadas a descrição da arquitetura, o fluxo de execução do *pipeline*, os pré-requisitos de funcionamento e as sugestões de uso da solução.

O **Capítulo 5** apresenta as propostas de análises de desempenho e perfilamento para avaliação da solução, a metodologia utilizada e a definição das métricas a serem analisadas. Os resultados obtidos também são apresentados neste capítulo.

O **Capítulo 6** expõe as considerações finais acerca do trabalho realizado até o momento. Além disso, apresenta algumas propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Gerenciar a configuração de ativos de rede heterogêneos utilizando a abordagem de Infraestrutura como Código é um desafio para o time de administradores de rede e envolve uma mudança no paradigma. Neste capítulo, os conceitos básicos e os benefícios inerentes a Infraestrutura como Código, cultura DevOps e NetDevOps são brevemente introduzidos. Em seguida, são apresentadas as ferramentas utilizadas na solução proposta, bem como outras ferramentas alternativas citadas por alguns trabalhos relacionados, que serão apresentados no capítulo seguinte. Essas ferramentas estão agrupadas de acordo com as principais características comuns e são apresentadas neste capítulo com o objetivo de auxiliar o entendimento do leitor e facilitar a localização das definições para os casos em que houverem dúvidas sobre a atuação das mesmas ao serem citadas nos capítulos subsequentes.

2.1 Infraestrutura como Código

De acordo com Rahman, Mahdavi-Hezaveh e Williams (2019), o conceito de IaC é atribuído por vários profissionais a Chad Fowler, após uma publicação em seu blog em 2013 falando sobre infraestrutura imutável[1]. O termo "*as Code*" corresponde à aplicação de práticas tradicionais da Engenharia de *Software* como a revisão e o versionamento de código em *scripts*, arquivos de configurações e outros tipos de arquivos que possam ser utilizados para gerenciar a infraestrutura de TI. Outros profissionais da área como Martin Fowler[2], Greg Orzell[3], James Carr[4], Kief Morris[5] e Ben Butler-Cole[6] também contribuíram com ideias e trabalhos significativos sobre Infraestrutura como Código e imutável[7].

A infraestrutura imutável, também citada por Chad Fowler, é implementada via IaC para fornecer estabilidade, eficiência e fidelidade aos seus aplicativos por meio da automação e do uso de padrões de programação bem sucedidos. A ideia básica é que a mesma seja criada e operada usando o conceito de programação de imutabilidade; ou seja, uma vez que algum recurso seja criado, ele não é mais alterado. Em vez disso, o recurso deve ser substituído por outro que contenha as alterações necessárias para garantir o comportamento adequado.

Para provisionar uma infraestrutura como código, os administradores geralmente utilizam uma sintaxe específica para determinar as configurações dos recursos da infraestrutura de maneira

[1] Chad Fowler: http://chadfowler.com/2013/06/23/immutable-deployments.html
[2] Martin Fowler: https://martinfowler.com/bliki/PhoenixServer.html
[3] Greg Orzell: https://netflixtechblog.com/building-with-legos-d68368fe4ce
[4] James Carr: https://dzone.com/articles/immutable-servers-packer-and
[5] Kief Morris: http://kief.com/immutable-server.html
[6] Ben Butler-Cole: https://www.thoughtworks.com/insights/blog/rethinking-building-cloud-part-4-immutable-servers
[7] Blog O'Reilly: https://www.oreilly.com/radar/an-introduction-to-immutable-infrastructure/

semelhante ao que ocorre com o desenvolvimento do código fonte de um *software*. De forma genérica, essa sintaxe é denominada de *Domain-specific language* (DSL) e varia de acordo com a ferramenta utilizada para descrever determinados recursos da infraestrutura.

Segundo Morris (2016), os principais benefícios ao utilizar IaC para implantar, gerenciar e atualizar a infraestrutura de TI em uma organização são:

1. **Sistemas fáceis e rápidos de serem reproduzidos**. Os administradores (ou até desenvolvedores) podem provisionar e configurar toda a infraestrutura desde o início, simplesmente escrevendo alguns *scripts* e definindo configurações em arquivos. Cada elemento da infraestrutura pode ser reproduzido repetidamente de forma confiável e com o mínimo de esforço. Novos serviços e aplicativos podem ser implantados na infraestrutura facilmente, o que torna o processo de implantação de *software* mais ágil e eficiente.

2. **Sistemas descartáveis**. Os recursos das novas infraestruturas dinâmicas são facilmente criados, destruídos, atualizados, redimensionados e realocados no sistema. Ainda assim, o *software* implantado pode ser executado mesmo quando o servidor em que ele é processado for excluído, movido ou redimensionado. Melhorias e correções para a infraestrutura se tornaram mais fáceis de serem aplicadas. Isso é crucial em infraestruturas de nuvem em grande escala onde o sistema não pode depender do *hardware* subjacente.

3. **Consistência de configuração**. Erros humanos sempre causaram problemas para a consistência da configuração, mesmo quando os procedimentos padrão são seguidos. A dependência do esforço humano torna a configuração suscetível a desvios inesperados (p.ex., erros de digitação), que acabam dificultando e atrasando um eventual processo de auditoria. A implementação de IaC padroniza totalmente a configuração da infraestrutura, deixando pouco espaço para erros humanos.

4. **Sistemas e processos auto-documentados**. As equipes de TI mantêm esforços para documentar os sistemas de forma útil e precisa. As atualizações e melhorias na infraestrutura são implementadas em ritmo acelerado, tornando desafiador que a documentação esteja sempre atualizada. Devido a isso, o processo de documentação pode necessitar de detalhes mais precisos inerentes à configuração do ambiente. Portanto, a documentação gerada manualmente pode acabar não representando de maneira fidedigna o que realmente está aplicado na infraestrutura. IaC consegue resolver este problema incluindo e explicando todas as etapas para executar um processo nos arquivos de definição e as ferramentas que realmente realizam o procedimento. Uma pequena documentação adicional é necessária e deve estar próxima (fisicamente e significativamente) ao código que ela explica para ajudar as pessoas a adquirirem uma boa compreensão dos procedimentos subjacentes.

5. **Versionamento da infraestrutura**. Como toda a infraestrutura é codificada em arquivos de definição, isso possibilita utilizar técnicas de controle de versão para acompanhar as

alterações confirmadas e reverter as alterações para uma versão anterior estável quando ocorre um erro.

6. **Sistemas e processos continuamente testados**. O teste automatizado é uma das práticas mais importantes que foram adicionadas ao desenvolvimento de infraestrutura com a introdução de IaC. Um erro de configuração, mesmo quando pequeno, pode rapidamente causar danos significativos. Escrever testes automatizados para gerenciar uma infraestrutura é um desafio, mas a implementação correta proporciona uma infraestrutura organizada, simples e funcional. O teste é realizado simultaneamente com o desenvolvimento da infraestrutura, detectando erros de forma mais rápida e possibilitando a correção antes da configuração ser aplicada nos ambientes produtivos.

2.2 DevOps

Segundo Davis e Daniels (2018), o termo DevOps é derivado do inglês, junção dos termos *Development* e *Operations*, e foi introduzido em 2008 na conferência Agile em Toronto-CA, por Andrew Shafer e Patrick Debois. DevOps é definido como uma cultura na qual a comunicação constante entre os times de desenvolvimento e operações aplicam um conjunto de práticas para reduzir o tempo de implantação de novas funcionalidades no ambiente de produção. O objetivo é diminuir a taxa de falhas das aplicações e aumentar a qualidade dos produtos e serviços oferecidos.

Há uma tendência de presumir que DevOps seja apenas a automação dos processos de desenvolvimento e implantação de *software*, mas o que Andrew Shafer e Patrick Debois propõem, especificamente, é uma alteração no paradigma de desenvolvimento do *software* e também da operação da infraestrutura. Embora pressuponha alto grau de automação na execução das tarefas do ciclo de vida, para que funcione efetivamente, é necessário que haja uma integração das equipes de desenvolvimento e operações, de forma a haver uma contínua retroalimentação de conhecimento e colaboração entre as duas equipes.

2.3 NetDevOps

O termo NetDevOps é derivado do inglês, em que há a junção dos termos *Network*, *Development* e *Operations*. Não há um consenso sobre o responsável pela criação do termo, mas ele é apresentado por Shah e Dubaria (2019) como sendo a aplicação dos princípios da cultura DevOps à engenharia e operações de rede, com o intuito de prover uma infraestrutura de rede controlada por uma implantação consistente de versões de configuração, com provisionamento paralelo e automatizado.

NetDevOps também vai além da automação de configuração e processos. Esse termo compreende a cultura, os métodos técnicos, as estratégias e as melhores práticas da cultura

DevOps, utilizadas para realizar o gerenciamento de configuração de ativos de rede. Ao praticar a cultura NetDevOps, o time de administradores precisa armazenar a configuração dos ativos de rede em um sistema de controle de versão, tal como o Git[8]. Dessa forma, o time passa a tratar o sistema de controle de versão como o ponto de controle e de origem única das mudanças aplicadas nos ambientes produtivos.

2.4 NaC - *Network as Code*

O termo NaC (do inglês *Network as Code*) é apresentado por PRESTON (2018) como sendo a aplicação da abordagem IaC ao domínio específico de Rede de Computadores, incluindo *data centers* tradicionais, redes locais, metropolitanas e ambientes de nuvem. A implementação bem sucedida do NaC faz parte da adoção mais ampla da cultura NetDevOps em uma organização público/privada. O uso do NaC envolve mudanças significativas no paradigma de projetar e operar uma rede de computadores, desde a colaboração entre as pessoas à seleção de de ferramentas utilizadas para gerenciar as configurações dos ativos de rede.

PRESTON (2018) propõem três princípios para o uso do NaC:

- Armazenamento das configurações dos ativos de rede em um repositório de controle de versão (por exemplo, Git) junto com informações sobre quem fez as alterações, quando e por quê;
- O sistema de controle de versão é a única fonte da verdade a cerca das configurações - cada ambiente pode ser construído usando as mesmas definições de configuração com variações mínimas;
- Implantação das configurações com APIs programáticas, se possível padronizadas, para facilitar a interoperabilidade, interconectividade, transferibilidade e consistência em larga escala.

2.5 *Pipeline*

O *pipeline* no desenvolvimento de *software* consiste em um conjunto de ferramentas, fluxos e processos automatizados, permitindo que as equipes utilizem tecnologias para construir e implantar *software* (INDIA, 2019). O *pipeline* pode ser utilizado para automatizar o processo de compilação, implantação, teste, entrega e/ou homologação de um ou mais *softwares* (HUMBLE; FARLEY, 2010). Na prática, o *Pipeline* é escrito como código e reúne uma sequência de instruções que podem ser executadas por ferramentas de integração, entrega e implantação contínua.

India (2019) resume o ciclo de vida de um *pipeline* como:

[8] Git: https://git-scm.com

- Escrita do código;
- Execução de testes do código;
- Implantação da aplicação/configuração no servidor do ambiente de testes;
- Testes no funcionamento da aplicação;
- Correção de problemas;
- Implantação no servidor no ambiente de produção;
- Monitoramento da aplicação;
- Repetição do processo para aplicar melhorias continuamente.

Um *pipeline* bem estruturado fortalece a capacidade da organização de entregar mudanças e melhorias mais rapidamente com lançamentos mais frequentes, enfatizando a melhoria contínua e maior satisfação do cliente.

Nesta pesquisa foi desenvolvido um *pipeline* para realizar a integração de diferentes *softwares* com o objetivo de proporcionar a gerência de configuração automatizada dos ativos de rede heterogêneos utilizando a Infraestrutura como Código.

2.6 Ferramentas de Infraestrutura como Código

Existem várias ferramentas que utilizam a abordagem IaC, cada uma com sua própria DSL e sintaxe particular, tal como ocorre com linguagens de programação utilizadas para o desenvolvimento de *software*. Além disso, cada ferramenta possui um conjunto de funcionalidades específicas, concebido com o objetivo de suprir as necessidades de um cenário específico (TORBERNTSSON; RYDIN, 2014).

Esta seção busca apresentar e classificar as definições de ferramentas que se destacam no contexto de Infraestrutura como Código e que são referenciadas ao longo deste trabalho. Algumas destas estão incluídas na solução proposta por esta pesquisa, certas ferramentas são destacadas em trabalhos relacionados e algumas são citadas para enfatizar o entendimento de um conceito específico. Com isso, pretende-se familiarizar o leitor com tais ferramentas e facilitar o entendimento dos capítulos subsequentes. Para melhor apresentação das mesmas, as soluções são classificadas como ferramentas de:

- gerência de configuração;
- virtualização;
- serviços de versionamento;

- integração, entrega e implantação contínua;
- edição de arquivos criptografados;
- gerenciamento de banco de dados.

Importante ressaltar que todas as ferramentas abordadas nesta seção são de código aberto, de licença livre de uso (completa ou parcialmente) e compatíveis com sistemas *Unix-like*. Além disso, salienta-se que boa parte das informações apresentadas a seguir foram obtidas a partir dos sites oficiais das ferramentas.

2.6.1 Ferramentas de Gerência de Configuração

Dentre as soluções que se destacam no critério de gerenciamento de configurações, citam-se as seguintes: **Ansible**[9], **Chef**[10], **Puppet**[11], **SaltStack**[12], **Salt SProxy**[13] e **Napalm**[14].

O Ansible é uma ferramenta amplamente utilizada para o provisionamento e gerência de configuração de servidores, além da configuração de ativos de rede, e foi desenvolvida a partir da integração de diversas tecnologias como Python, PowerShell, Shell Bash e Ruby. Para declarar a configuração, Ansible implementa uma DSL própria e aplicada no formato de *playbook*.

O Chef também é uma ferramenta de gerência de configuração bastante difundida, que foi desenvolvida em Ruby e Erlang. Mais especificamente, utiliza uma DSL definida originalmente em Ruby, para escrever as chamadas "receitas" (*recipes*) de gerenciamento de aplicações, servidores e utilitários. Receitas podem ser agrupadas em *cookbooks* e descrevem recursos como pacotes, serviços e arquivos, além de seus respectivos estados.

O Puppet pode ser definido como uma ferramenta que auxilia na gerência e automação da configuração de servidores. Com ele, é possível definir o estado desejado dos sistemas por meio de um código escrito em uma DSL específica chamada de *Puppet manifest*. O desenvolvimento do Puppet a partir das linguagens C++, Clojure e Ruby, além de uma camada de abstração, possibilita descrever a configuração dos recursos em alto-nível, como usuários, serviços e pacotes; sem a necessidade de comandos específicos do sistema operacional.

O SaltStack é uma ferramenta de automação da configuração de uma infraestrutura de rede e operações de segurança. Desenvolvido em Python, o Saltstack também possui uma DSL própria assim como as ferramentas citadas anteriormente. Possui como características ser uma ferramenta de linha de comando, ter comunicação baseada em SSH e a utilização de arquivos no formato YAML (do inglês *YAML Ain't Markup Language*) como entrada e saída de configurações.

[9] Ansible: https://www.ansible.com
[10] Chef: https://www.chef.io
[11] Puppet: https://puppet.com
[12] SaltStack: https://www.saltstack.com
[13] Salt SProxy: https://salt-sproxy.readthedocs.io
[14] Napalm: https://napalm-automation.net

Funciona em uma arquitetura *cliente-servidor*, cuja comunicação com o servidor é realizada através dos chamados *Proxy Minions*, que são módulos para controlar remotamente os serviços através de uma API REST[15].

O Salt SProxy não é necessariamente uma ferramenta independente, mas sim um *plug-in* para o SaltStack, cujo objetivo é automatizar o gerenciamento e a configuração de ativos de rede em larga escala. Uma diferença fundamental é que o Salt SProxy não utiliza componentes Proxy Minion. Com isso, pode usufruir dos benefícios de escalabilidade, flexibilidade e extensibilidade do SaltStack, sem a necessidade de gerenciar diversos "Minions" para os respectivos serviços.

O Napalm (do acrônimo em inglês, *Network Automation and Programmability Abstraction Layer with Multivendor support*) é uma biblioteca escrita em Python que implementa um conjunto de funções para interagir com diferentes sistemas operacionais específicos de ativos de rede, através de uma API unificada. Possui diversos métodos de conexão com os ativos, o que possibilita a manipulação dos arquivos de configuração e a recuperação dos dados. Sua flexibilidade permite a integração com outras ferramentas, como Ansible e SaltStack.

2.6.2 Ferramentas de Virtualização

Ainda mais abrangentes do que as ferramentas de gerência de configuração, diversas soluções de virtualização estão disponíveis no mercado e no meio acadêmico, variando desde simples mecanismos para prover virtualização leve através de contêineres, até orquestradores complexos de infraestruturas virtualizadas. Portanto, para exemplificar essa variedade em termos de ferramentas, apresentam-se os seguintes exemplos: **OpenStack**[16], **Vagrant**[17] e **Docker**[18].

O OpenStack é uma plataforma de computação em nuvem em que é possível controlar grandes conjuntos (*pools*) de recursos em um centro de dados. Por exemplo, esta ferramenta possibilita o gerenciamento de componentes virtuais (máquinas virtuais ou contêineres), bem como os recursos de computação, armazenamento e rede utilizados pelos mesmos. Através de diversos módulos adicionais disponíveis, várias funcionalidades alternativas podem ser incorporadas para orquestrar os componentes virtuais e recursos físicos.

Com um conjunto de funcionalidades mais simples, o Vagrant é um utilitário de linha de comando para gerência do ciclo de vida de componentes virtuais. A partir dele, é possível automatizar a criação de componentes virtuais personalizados, inclusive com o auxílio de ferramentas de configuração descritas anteriormente, como Ansible, Puppet ou Chef. Ressalta-se que o Vagrant não é o responsável por executar a virtualização, mas sim coordenar as ações de criar, personalizar e remover componentes virtuais. A virtualização é realizada por hipervisores como VMware ou VirtualBox, chamados de *providers* no jargão do Vagrant.

[15] API REST: https://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm
[16] OpenStack: https://www.openstack.org
[17] Vagrant: https://www.vagrantup.com
[18] Docker: https://www.docker.com

O Docker é uma solução de virtualização leve realizada através de contêineres, que permite o isolamento e empacotamento de aplicações para entrega rápida de *software* em diferentes ambientes. Além da implementação do conceito, a solução Docker provê várias funcionalidades que facilitam a gerência da aplicação. A execução de tais aplicações neste ambiente virtualizado é considerada leve, pois exige menos recursos computacionais do que máquinas virtuais tradicionais.

2.6.3 Ferramentas de Controle de Versão

Da década de 70 até os dias atuais, diversos sistemas de versionamento de código têm sido utilizados no desenvolvimento de sistemas (RUPARELIA, 2010), com destaque para os amplamente consolidados **CVS**, **SVN** e **Git** (OTTE, 2009). Nos últimos ano, o Git se tornou o sistema de controle de versão mais utilizado em projetos de *software* (ZOLKIPLI; NGAH; DERAMAN, 2018). Caracteriza-se por ser um sistema de controle de versão distribuído, de código aberto e gratuito, projetado para lidar com projetos de pequeno a grande porte, com agilidade e eficiência. Dada a sua popularidade, diversas plataformas oferecem serviços de versionamento baseados no Git, como por exemplo: **Gitlab**[19], **BitBucket**[20], **Github**[21] e **Gogs**[22]. Diferentemente das três primeiras, que oferecem serviço *on-line* de hospedagem do código, o Gogs é um serviço Git auto-hospedado para versionamento de arquivos. Portanto, pode ser implantado tanto localmente quanto em um servidor remoto, de acordo com a necessidade da equipe de administração.

2.6.4 Ferramentas de Integração, Entrega e Implantação Contínua

Dentro das premissas do DevOps, práticas importantes no desenvolvimento moderno de *software* são constituídas pela integração, entrega e implantação contínua (do termo em inglês, *Continuous Integration and Continuous Delivery* - CI/CD). Como não poderia ser diferente, ferramentas que implementam tais práticas auxiliam o desenvolvimento e a entrega de sistemas. Como exemplos de sistemas que proveem CI/CD, podem ser citados o **Gitlab**[10] e o **Jenkins**[23]. O primeiro oferece, além dos serviços de versionamento citados anteriormente, funcionalidades de edição de arquivos e CI/CD. Da mesma forma, o Jenkins se destaca por oferecer um serviço de automação de código, que pode ser utilizado para automatizar tarefas relacionadas à criação, teste, entrega e implantação de *software*.

[19] Gitlab: https://docs.gitlab.com
[20] BitBucket: https://bitbucket.org
[21] Github: https://github.com
[22] Gogs: https://gogs.io
[23] Jenkins: https://www.jenkins.io

2.6.5 Ferramentas de Edição de Arquivos Criptografados

Para garantir o desenvolvimento seguro do código de IaC e uma operação confiável dos arquivos de configuração, diversas ferramentas para criptografar arquivos podem ser utilizadas, como por exemplo: **Git-crypt**[24], **BlackBox**[25], **Transcrypt**[26] e **Sops**[27].

Git-crypt é uma ferramenta de linha de comando que permite criptografia e descriptografia transparentes de arquivos em um repositório Git. BlackBox também é uma ferramenta de linha de comando que proporciona a criptografia e descriptografia de arquivos específicos, utilizando-se do *Gnu Privacy Guard* (GPG)[28]. Transcrypt é um *script* para configurar a criptografia transparente de arquivos confidenciais armazenados em um repositório Git. Por fim, o Sops é um editor de arquivos criptografados que suporta vários formatos de arquivo e criptografa-os utilizando chaves simétricas ou assimétricas, além de dar suporte a vários serviços de nuvem que oferecem o armazenamento e gerenciamento dessas chaves criptográficas.

2.6.6 Ferramentas de Gerenciamento de Banco de Dados

Por último, mas de igual importância em projetos de desenvolvimento de *software*, os Sistemas de Gerenciamento de Banco de Dados (SGBD) proporcionam não apenas o acesso à informação quando necessário, mas também a gerência das estruturas de *software* em uso. Dentre uma grande variedade de alternativas existentes, o **MySQL**[29] e o **PostgreSQL**[30] são algumas das soluções mais populares que oferecem opção de licença gratuita[31]. Ambos se caracterizam como sistemas de gerenciamento de banco de dados relacional de objetos, com código aberto. Além disso, as duas soluções possuem mais de 30 anos de desenvolvimento ativo e ganharam forte reputação de confiabilidade, robustez de recursos e desempenho.

Algumas das ferramentas apresentadas neste capítulo são utilizadas na solução proposta, mas outras são citadas apenas em alguns dos trabalhos relacionados com esta pesquisa, a serem apresentados em seguida. Essas ferramentas foram agrupadas com o objetivo de auxiliar o entendimento dos próximos capítulos, além de facilitar a busca posterior pelas definições e características de tais ferramentas, nos casos de eventuais dúvidas sobre as mesmas.

[24] Git-crypt: https://github.com/AGWA/git-crypt
[25] BlackBox: https://github.com/StackExchange/blackbox
[26] Transcrypt: https://github.com/elasticdog/transcrypt
[27] Sops: https://github.com/mozilla/sops
[28] GPG: https://gnupg.org
[29] MySQL: https://www.mysql.com/
[30] PostgreSQL: https://www.postgresql.org
[31] Tendências de uso sobre bancos de dados: https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databases-single-vs-multiple-database-use/

3 ESTADO DA ARTE

Este capítulo apresenta os trabalhos relacionados à gerencia de configuração de ativos de rede heterogêneos. Com o propósito de investigar o estado da arte, um protocolo de revisão sistemática da literatura foi elaborado para identificar os trabalhos que se relacionam com esta pesquisa. Neste contexto, primeiramente é resgatada a principal questão da pesquisa e em seguida é apresentado o protocolo de busca utilizado, bem como os resultados obtidos durante a realização da revisão sistemática. A primeira seção apresenta o protocolo de busca proposto, contendo o termo de busca, as bases de conhecimento utilizadas e os critérios de exclusão empregados para filtrar os resultados. Na seção seguinte, são apresentados os trabalhos selecionados a esta pesquisa encontrados durante a revisão sistemática. Os trabalhos são descritos, relacionados com esta pesquisa e categorizados de acordo com *facetas de pesquisa* e/ou *facetas tecnológicas*.

3.1 Protocolo de Revisão Sistemática

> *"Como integrar, de maneira eficiente e eficaz, diferentes softwares para realizar a automação da configuração de uma infraestrutura de ativos de rede?"*

Esta é a principal questão de pesquisa que norteia a revisão sistemática delineada nesta seção. A resposta para esta pergunta contribui para entender que abordagens ou soluções podem ser utilizadas com o intuito de automatizar a configuração de ativos de rede, como elas podem ser integradas e o que esperar do resultado produzido. Para atingir este objetivo, define-se um protocolo de busca que está delimitado pela área de conhecimento da Ciência da Computação, com foco na subárea de Redes de Computadores. Especificamente nesta subárea, a gerência de configuração de ativos de redes é o tema principal de busca. A elaboração do protocolo proposto foi baseada nos trabalhos de Petersen et al. (2008) e Almeida, Maciel Jr. e Verdi (2020).

Para realizar a busca por trabalhos relacionados na literatura, foram utilizadas como bases de conhecimento a **ACM Digital Library**[1], a **Springer**[2], a **Science Direct**[3] e o **IEEE Xplore**[4]. Embora existam outras fontes de trabalhos acadêmicos, considera-se que as bases escolhidas agrupam os principais trabalhos na área de gerência de redes, além de também indexarem trabalhos de outros repositórios. Estas bases serviram como fontes de coleta a partir do termo de busca descrito a seguir, cada qual com suas particularidades em relação ao motor de busca, refletidas em alguns critérios de exclusão definidos adiante.

[1] ACM: https://dl.acm.org
[2] Springer: https://www.springer.com
[3] Science Direct: https://www.sciencedirect.com
[4] IEEE Xplore: https://ieeexplore.ieee.org/Xplore

```
((network OR remote)
 AND
 (device OR router OR switch OR asset)
 AND
 (configuration OR setup)
 AND
 (automatic OR automation)
 AND
 ("infrastructure as code" OR SDN OR "Software-defined network"))
```

Foram definidos neste protocolo 15 critérios de exclusão de trabalhos, dos quais os 7 primeiros são comuns a todas as bases do conhecimento e os 8 restantes são relacionados à uma determinada base. A seguir, são listados primeiramente os critérios de exclusão comuns, que foram definidos para, de maneira geral, remover estudos que não estão relacionados com o objetivo da pesquisa.

- CE1: artigos com conteúdo irrelevante para os objetivos da pesquisa, como por exemplo: redes sem fio, IoT, segurança de aplicações, análise de tráfego, *cloud computing*, dispositivos móveis, etc;
- CE2: artigos com ano de publicação inferior a 2015;
- CE3: publicações que não estiverem no idioma inglês;
- CE4: trabalhos que não foram publicados em conferências ou periódicos;
- CE5: publicações duplicadas;
- CE6: estudos que não são artigos completos (p.ex. resumos, demonstrações e cartazes);
- CE7: trabalhos que tratam de *Software Defined Network*, mas são diretamente relacionados a novas propostas de protocolos, monitoramento, segurança, análise de tráfego ou proposição de arquitetura, não relacionadas com configuração de ativos de rede;
- CE8: excluídas as publicações que não estiverem no formato PDF.

Os critérios de exclusão específicos, conforme os filtros disponíveis no mecanismo de busca de cada base de conhecimento utilizada nesta pesquisa, foram:

IEEE Xplore

- CE9: excluídas as publicações que não pertencem aos tópicos: *software defined networking*, *protocols*, *computer network management*;

ACM Digital Library

- CE10: excluídas as publicações que não são do tipo: *Research Articles*;

Springer

- CE11: excluídas as publicações que não são do tipo: *Article* e *Conference Paper*;
- CE12: excluídas as publicações que não pertencem a área: *Computer Science*;
- CE13: excluídas as publicações que não pertencem a subárea: *Computer Communication Networks*;

Science Direct

- CE14: excluídas as publicações que não são do tipo: *Research Articles*;
- CE15: excluídas as publicações que não pertencem a subárea: *Computer Networks*.

A busca foi realizada em 10 de maio de 2020 e 1.079 documentos foram indexados a partir de alguns dos critérios de exclusão listados anteriormente. A Tabela 1 mostra a quantidade de publicações encontradas nos resultados da busca em cada base, bem como os critérios de exclusão aplicados e os resultados obtidos após a aplicação dos mesmos.

Tabela 1 – Quantidade de publicações contidas nos resultados da busca.

Critérios de exclusão	**Quantidade de publicações por base**			
	IEEE Xplore	***ACM***	***Springer***	***Science Direct***
Resultados após aplicação dos critérios de exclusão: CE2-CE4 e CE9-CE15	39	419	258	363
Total de artigos pré-selecionados				1.079
Resultados após aplicar os critérios: CE1 e CE5-CE8	2	6	2	–
Publicações selecionadas				10

Conforme mostra a Tabela 1, o termo de busca aplicado nas bases de conhecimento citadas anteriormente retornou um total de 1.079 resultados, quando aplicados os critérios de exclusão CE2-CE4 e CE9-CE15. A partir deste total, foram analisados os dados provenientes do título, resumo e palavras-chaves para aplicação dos critérios de exclusão CE1 e CE5-CE8.

Após a aplicação desses critérios, apenas 10 publicações foram selecionadas para a leitura completa do conteúdo e incluídas para uma análise aprofundada.

Antes de uma investigação detalhada, foi realizado um estudo classificatório dos 10 trabalhos selecionados, com o objetivo de comparar o posicionamento da pesquisa proposta neste trabalho em relação aos mesmos. Seguindo uma metodologia similar à descrita por Almeida, Maciel Jr. e Verdi (2020), para cada artigo foi definido o seu posicionamento de acordo com dois tipos de facetas: *facetas de pesquisa* (FP) e *facetas tecnológicas* (FT).

As facetas de pesquisa foram definidas com base no processo de classificação proposto por Wieringa et al. (2006) e são descritas a seguir:

- **FP1 -** ***Pesquisa de aferição***: este tipo de trabalho mostra como uma determinada técnica é implementada na prática (implementação da solução) e quais são as consequências da implementação em termos de benefícios e desvantagens (avaliação da solução);
- **FP2 -** ***Proposta de solução***: a solução pode ser nova ou uma extensão significativa de uma existente, em que as vantagens potenciais e a aplicabilidade da mesma são mostradas por um pequeno exemplo ou uma boa linha de argumentação;
- **FP3 -** ***Pesquisa de validação***: com o intuito de validar proposições, as técnicas investigadas são teóricas e ainda não foram implementadas na prática, como por exemplo, experimentos executados em laboratório;
- **FP4 -** ***Artigos "filosóficos"***: tais trabalhos esboçam uma nova maneira de perceber uma realidade, estruturando o campo de pesquisa na forma de taxonomia ou estrutura conceitual;
- **FP5 -** ***Artigos de experiências pessoais***: relatos de experiência que explicam o processo de como algo foi feito na prática, geralmente retratados sob a perspectiva pessoal do(s) autor(es) do trabalho;
- **FP6 -** ***Artigos de opinião***: esses documentos expressam a opinião de alguém se uma determinada técnica (ou solução) é boa ou ruim, ou como as coisas devem ser feitas; usualmente sem considerar trabalhos relacionados e metodologias de pesquisa.

A Figura 1 apresenta a classificação dos artigos selecionados quanto às facetas de pesquisa. Ressalta-se que um determinado artigo pode estar associado a uma ou mais facetas. Embora o número pequeno de trabalhos diretamente relacionados impossibilite tirar conclusões generalizadas a respeito, acredita-se que existe uma tendência por artigos que propõem uma solução e busquem por validá-la, além dos artigos que relatam a experiência em implantar tais soluções. O trabalho proposto nesta pesquisa caracteriza-se também por essas 3 facetas (FP2, FP3 e FP5), o que indica um alinhamento com as pesquisas realizadas na literatura. Portanto, é possível observar uma maior presença na comunidade acadêmica por trabalhos que apresentam estes aspectos de pesquisa.

As facetas tecnológicas utilizadas para classificar os trabalhos foram divididas nas nove categorias a seguir:

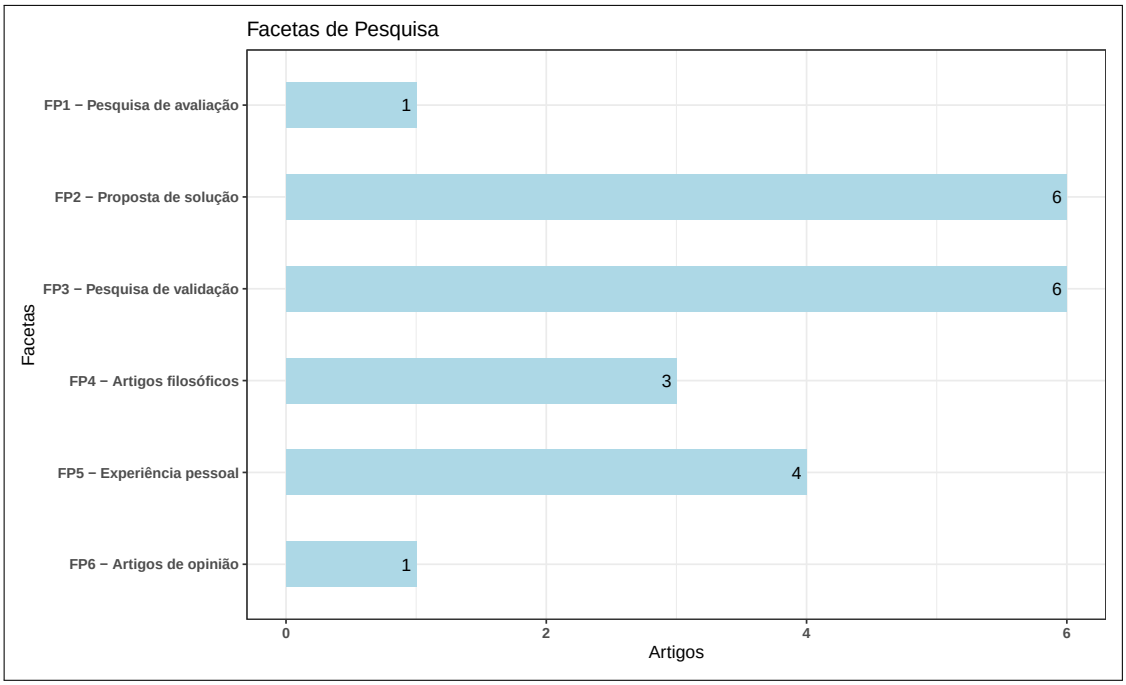


Figura 1 – Classificação dos trabalhos por facetas de pesquisa. Fonte: Elaboração própria.

- **FT1 - *Infraestrutura como Código***: trabalhos que abordam conceitos e/ou ferramentas relacionadas a IaC;
- **FT2 - *Redes Definidas por Software***: artigos que tratam sobre conceitos e aspectos relacionados a SDN;
- **FT3 - *Gerência de configuração***: trabalhos que apresentam ou propõem ferramentas que fazem a gerência de configuração;
- **FT4 - *Código fonte aberto***: artigos que falam sobre soluções de código fonte aberto;
- **FT5 - *DevOps***: trabalhos que falam sobre cultura DevOps, princípios e conceitos relacionados à mesma;
- **FT6 - *Versionamento***: estudos que tratam sobre versionamento de *software* ou de configuração como código;
- **FT7 - *NetDevOps***: artigos que abordam a cultura NetDevOps, princípios e conceitos relacionados;
- **FT8 - *Interoperabilidade***: artigos que abordam a interoperabilidade de soluções e os benefícios de *softwares* que dão suporte a múltiplos fabricantes;
- **FT9 - *Configurações simultâneas***: estudos que apresentam ferramentas que gerenciam a configuração simultânea de diversos ativos.

Estas facetas foram assim definidas porque representam as características que devem estar presentes na solução proposta nesta pesquisa, tanto em relação ao seu posicionamento como paradigma tecnológico (FT1, FT2, FT5 e FT7), quanto a critérios de funcionalidade (FT3, FT4, FT6, FT8 e FT9). A Figura 2 apresenta a classificação dos 10 artigos selecionados quanto às facetas tecnológicas. Novamente, destaca-se que um artigo pode estar associado a uma ou mais facetas. Percebe-se pelos resultados que quase todas as características são encontradas em outros trabalhos da literatura, com exceção de configurações paralelas. Contudo, conclui-se também que, apesar de importantes, as facetas tecnológicas que indicam funcionalidades foram pouco exploradas na literatura, possivelmente indicando uma área promissora de pesquisa.

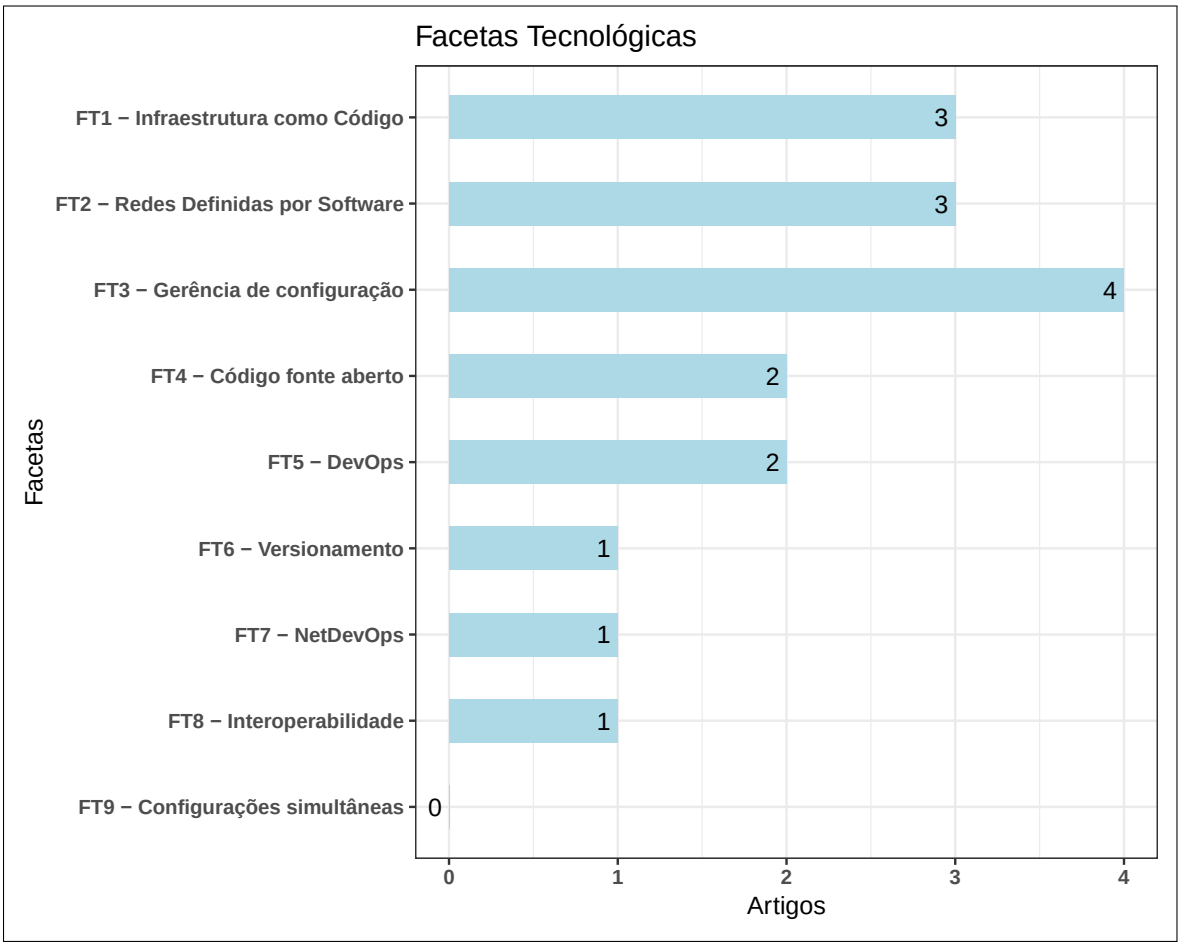


Figura 2 – Classificação dos trabalhos por facetas tecnológicas. Fonte: Elaboração própria.

Ao que tudo indica, a solução proposta é a primeira a apresentar todas as características e funcionalidades representadas pelas facetas tecnológicas, dentro do contexto da gerência de infraestrutura de rede apresentado neste trabalho.

3.2 Trabalhos Relacionados

Os trabalhos apresentados a seguir são agrupados de acordo com as facetas tecnológicas FT1, FT2, FT5, FT7 e FT8, apresentadas na seção anterior. As facetas FT3, FT4 e FT6 são

características ortogonais a algumas das soluções apresentadas, além da ausência de trabalhos a partir da faceta FT9.

3.2.1 Infraestrutura como Código

Jiang e Adams (2015) apresentam um estudo empírico analisando o uso da abordagem de IaC em 265 repositórios de código no projeto OpenStack. O estudo analisa a quantidade de trabalho envolvido na manutenção dos arquivos de configuração de uma infraestrutura gerenciada como código, utilizando as ferramentas Chef e Puppet. O estudo analisa ainda o tamanho e a complexidade dos arquivos de especificação da infraestrutura. Os autores descobriram que tais arquivos são grandes e recebem alterações frequentes, o que pode indicar uma potencial introdução de erros. Além disso, os autores concluíram que os arquivos de código de infraestrutura são fortemente acoplados entre os projetos, especialmente arquivos de teste, o que implica que os testadores geralmente precisam alterar as especificações da infraestrutura ao fazer alterações na estrutura de testes. Contudo, o trabalho não trata especificamente sobre a gerência de configuração de *switches* e roteadores, e sim de servidores e máquinas virtuais. A solução proposta nesta pesquisa utiliza as ferramentas Salt SProxy e Napalm para gerenciar a configuração de *switches* e roteadores heterogêneos, sem necessariamente precisar alterar com grande frequência os arquivos de especificação da infraestrutura como código, consequentemente sem apresentar um forte acoplamento com outros projetos de gerência de configuração.

Duplyakin, Haney e Tufo (2015) relataram a experiência no gerenciamento de um *cluster* Linux com 20 nós, utilizando uma implementação de IaC através da ferramenta Chef. Os autores analisaram a utilização e o custo dos recursos necessários, além de indicarem que o sistema projetado é uma alternativa de baixo custo à aquisição de *hardware* físico adicional para fortalecer o gerenciamento da infraestrutura. Eles também destacaram os recursos de segurança e a capacidade de gerenciamento do protótipo, adequados para implantações maiores e prontas para produção. Entretanto, assim como o trabalho apresentado anteriormente, não foi abordado o uso do Chef para gerenciar a configuração de *switches* e roteadores.

Vilalta et al. (2020) abordam uma visão geral da experiência prática no uso de ferramentas de programação para controlar e monitorar ativos de rede; o que permite a automação, controle e gerenciamento da configuração sem intervenção humana. Foram exploradas linguagens de modelagem de dados, além de protocolos de controle e monitoramento de redes. Contudo, ressalta-se que o uso de tais ferramentas e protocolos geralmente apresenta uma curva de aprendizado com tempo não desprezível, diferentemente do que se propõe a partir desta pesquisa, que utiliza uma solução integrada de fácil utilização. Na abordagem proposta pelos autores, outra característica diferente é a ausência de um sistema de controle de versão para rastrear as mudanças no código.

3.2.2 Redes Definidas por *Software*

Dwaraki et al. (2015) abordam o problema do controle de versões para o gerenciamento de estado de fluxo em redes habilitadas para SDN (do inglês, *Software Defined Network*). O principal objetivo é que o plano de dados subjacente possa fornecer melhor proteção de estado, proveniência, facilidade de programação e suporte para vários aplicativos. Inspirados pelas ferramentas de controle de versão utilizadas para o desenvolvimento de *software*, os autores propõem uma abstração em uma abordagem chamada GitFlow. A ideia central é utilizar um repositório Git para manter a cópia autorizada do estado de configuração do fluxo, rastrear as mudanças e automatizar a resolução de conflitos de configuração, reorganizando o novo estado de fluxo. Os autores não utilizaram uma ferramenta de gerência de configuração, mas como SDN oferece uma API para que os planos de controle e dados das redes sejam programáveis e facilmente evoluíveis, o comportamento da rede no plano de dados pode ser gerenciado por meio de um *software* externo. Esta pesquisa não trata da automação da configuração dos planos de controle e dados do SDN, mas pretende utilizar repositórios git para armazenar, versionar e centralizar as configurações dos *switches* e roteadores.

O artigo publicado por Chen, Wang e Loo (2018) aborda a proliferação de DSLs em redes de computadores. Segundo os autores, apesar dessas linguagens permitirem explorar a capacidade de programação das redes, elas não são amplamente adotadas. O primeiro motivo é o fato destas linguagens exigirem uma curva de aprendizado maior entre operadores que não são programadores treinados. Além disso, ocorre certas vezes que novas aplicações SDN "confiam" na funcionalidade de redes herdadas que não podem ser facilmente migradas ou analisadas. Para enfrentar tais desafios, os autores propuseram o Facon, uma ferramenta que permite gerar automaticamente programas em DSLs arbitrárias, com base em exemplos de entrada/saída. Como estes exemplos se aplicam a qualquer protocolo de rede, essa abordagem pode ser generalizada, permitindo migrar redes herdadas para novas DSLs ou transformar uma DSL em outra. O Facon foi utilizado em ambientes controlados de teste. O trabalho não aborda o uso de ferramentas de gerência de configuração, infraestrutura como código, integração contínua e versionamento de código dos exemplos gerados pelo Facon.

O trabalho publicado por Tian et al. (2019) retrata o uso de uma ferramenta chamada Jinjing na operação da rede WAN da Alibaba[5], através da atualização automática das configurações da *Access Control List* (ACL). O Jinjing permite que os operadores expressem as configurações do controle de tráfego nos roteadores utilizando uma linguagem declarativa, denominada LAI. No *kernel* do Jinjing, os autores desenvolveram um conjunto de novas técnicas de verificação e síntese para garantir rigorosamente a exatidão dos planos de atualização. O trabalho desenvolvido pelos autores é focado na automação da configuração de ACLs para roteadores em rede WAN, mais especificamente na tabela de roteamento do ativo. Em contrapartida, esta pesquisa se propõe a automatizar outras funcionalidades oferecidas por *switches* e roteadores, estejam eles

[5] Alibaba Cloud: https://www.alibabacloud.com

conectados em redes LAN, MAN ou WAN.

3.2.3 DevOps

Morales, Yasar e Volkman (2018) apresentam uma discussão sobre a implementação de práticas da cultura DevOps e da abordagem de Infraestrutura como Código em ambientes altamente regulamentados. Tais ambientes são tipicamente caracterizados por serem espaços físicos fechados, com sistemas de computador com maior segurança e controles de acesso por papéis e usuários, com segregação de funções e a incapacidade de pessoas levarem determinados artefatos para fora das instalações ou ter acesso facilitado às configurações de ativos de rede. A solução proposta nesta pesquisa possibilita o controle de acesso por papéis e usuários e também pode ser utilizada em um ambiente no qual o servidor e os ativos de rede encontram-se em redes virtuais privadas segregadas. A comunicação entre os componentes de *hardware* e *software* envolvidos pode ser criptografada se os ativos de rede derem suporte a protocolos de segurança e a equipe de administradores de rede decidir utilizá-los. Por exemplo, escolher acessar os ativos utilizando o protocolo SSH ao invés do Telnet, ou ainda HTTPS ao invés de HTTP. Todos os arquivos de configuração e *backup* gerados pela solução são armazenados de forma criptografada utilizando um algoritmo de chave simétrica. O acesso à chave simétrica é restrito e definido pela equipe de administradores de rede.

Leite et al. (2020) apresentam uma pesquisa sobre a cultura DevOps investigando e discutindo os desafios a partir da perspectiva de engenheiros, gerentes e pesquisadores. Foi realizada uma revisão sistemática e criado um mapa conceitual da cultura DevOps, correlacionando os conceitos com ferramentas de automação. Algumas das ferramentas apresentadas no trabalho são utilizadas na solução proposta nesta pesquisa, tais como: Git, Jenkins e Docker. Além disso, ao adotar a cultura DevOps no contexto de gerenciamento do ciclo de vida de aplicações, os desafios e os encadeamentos elencados no trabalho são os mesmos abordados nesta pesquisa ao gerenciar o ciclo de vida das configurações dos ativos de rede como código.

3.2.4 NetDevOps

A definição de NetDevOps é apresentada por Shah e Dubaria (2019) como sendo a aplicação dos princípios da cultura DevOps à engenharia e operações de rede, no intuito de prover uma infraestrutura de rede controlada por uma implantação consistente de versões de configuração e que serão implantadas com provisionamento paralelo e automatizado. NetDevOps também compreende a cultura, os métodos técnicos, as estratégias e as melhores práticas da cultura DevOps para o gerenciamento da configuração de ativos de rede. Esse artigo também apresenta o conceito de *Network as Code* (NaC), no qual a configuração dos ativos de rede são armazenadas em um sistema de controle de versão, tais como Git ou SVN. O trabalho sugere o uso de todas as características expostas como importantes em uma solução para automatizar a gerência de redes, porém não apresenta nenhum estudo de caso ou implementação empírica.

3.2.5 Interoperabilidade

Opara-Martins, Sahandi e Tian (2016) descrevem o problema de dependência de fornecedores como uma grande barreira para a adoção da computação em nuvem, devido à falta de padronização. As soluções e os esforços atuais para resolver o problema de dependência do fornecedor são predominantemente orientados à tecnologia. Os autores apresentam uma análise crítica do problema de dependência do fornecedor em uma perspectiva comercial. Foi realizada uma pesquisa baseada em abordagens qualitativas e quantitativas que permitiram identificar os principais fatores de risco que dão origem a situações de dependência de fornecedores. Os resultados da análise exemplificam a importância da interoperabilidade, portabilidade e padrões na computação em nuvem. Várias estratégias são propostas sobre como evitar e mitigar os riscos de bloqueio ao migrar para a computação em nuvem. As estratégias estão relacionadas a contratos, seleção de fornecedores que suportam formatos e protocolos padronizados em relação a estruturas de dados e APIs padrão, desenvolvendo a conscientização sobre pontos comuns e dependências entre soluções baseadas em nuvem. Apesar do trabalho não tratar especificamente da dependência de fornecedores de *switches* e roteadores, esses problemas podem fazer parte do cotidiano de uma equipe de administradores de rede.

Neste capítulo foram apresentados os trabalhos relacionados com esta pesquisa. A leitura desses trabalhos foi fundamental para investigar o estado da arte no contexto do uso de Infraestrutura como Código para configuração de ativos de redes, além de auxiliar o entendimento da questão principal da pesquisa, apresentada no início do capítulo. Os conceitos básicos sobre a Infraestrutura como Código, cultura DevOps e NetDevOps, bem como algumas das ferramentas citadas no capítulo anterior estão presentes nos trabalhos relacionados. O conhecimento adquirido durante a realização da revisão sistemática foi aplicado durante o desenvolvimento da solução proposta, que será apresentada no capítulo seguinte.

4 DESCRIÇÃO DA SOLUÇÃO PROPOSTA

Este capítulo apresenta a descrição da solução integrada de *software* proposta para gerenciar a configuração de ativos de rede heterogêneos. A primeira seção apresenta a descrição da arquitetura utilizada na solução desenvolvida, detalhando cada elemento que compõe a mesma. Também são apresentados os modelos de instalação e o papel de atuação de cada servidor que compreende a solução. Na seção seguinte, são apresentadas as justificativas utilizadas na seleção das ferramentas que integram a solução. A seção subsequente descreve os fluxos de trabalho das atividades realizadas pela solução. Mais adiante são apresentados os pré-requisitos de *hardware* e *software* para o seu correto funcionamento. Por fim, na última seção são apresentadas funcionalidades e sugestões de uso em ambientes comerciais e acadêmicos.

4.1 Arquitetura da Solução

No contexto desta pesquisa, uma solução integrada é a que incorpora todos os componentes de *software* necessários, configurados para interagirem e operarem em conjunto, sem que seja necessário controlá-los individualmente.

Com o intuito de facilitar sua referenciação, a solução proposta nesta pesquisa será denominada de **PipeConf** deste ponto em diante. O PipeConf é uma solução integrada de *software* para proporcionar a automação da configuração de ativos de rede de diferentes modelos e fabricantes, utilizando uma abordagem de Infraestrutura como Código. O PipeConf pode gerenciar a configuração de *switches* e roteadores em uma LAN (*Local Area Network*), MAN (*Metropolitan Area Network*) ou mesmo WAN (*Wide Area Network*), desde que haja conectividade entre os ativos e o(s) servidor(es) que hospeda(m) a solução.

O PipeConf apresenta uma arquitetura modular logicamente dividida em um ***Plano de Controle*** e um ***Plano de Dados***, conforme mostra a Figura 3. Os dois planos podem estar centralizados em um único servidor ou distribuídos em servidores distintos. No Plano de Dados é onde fica o **Módulo de Versionamento** (MV), responsável pelo registro das versões do código produzido e pelo armazenamento das configurações dos ativos de rede. No Plano de Controle, por sua vez, ficam os seguintes componentes:

- **Módulo de Integração Contínua** (MIC), que recebe as credenciais de acesso aos ativos de rede (endereço IP, porta, usuário, senha, etc.) e executa o *pipeline* integrando todas as ferramentas utilizadas no PipeConf;
- **Módulo de Gerência de Configuração** (MGC), que interage com os ativos de rede para aplicar novas configurações ou para obter o *backup* dos mesmos;

- **Módulo de Criptografia** (MC), responsável pela criptografia e decriptografia dos arquivos de credenciais e de configuração dos ativos de rede;
- **Módulo de Notificações** (MN), que envia notificações sobre o resultado das operações aos usuários da solução.

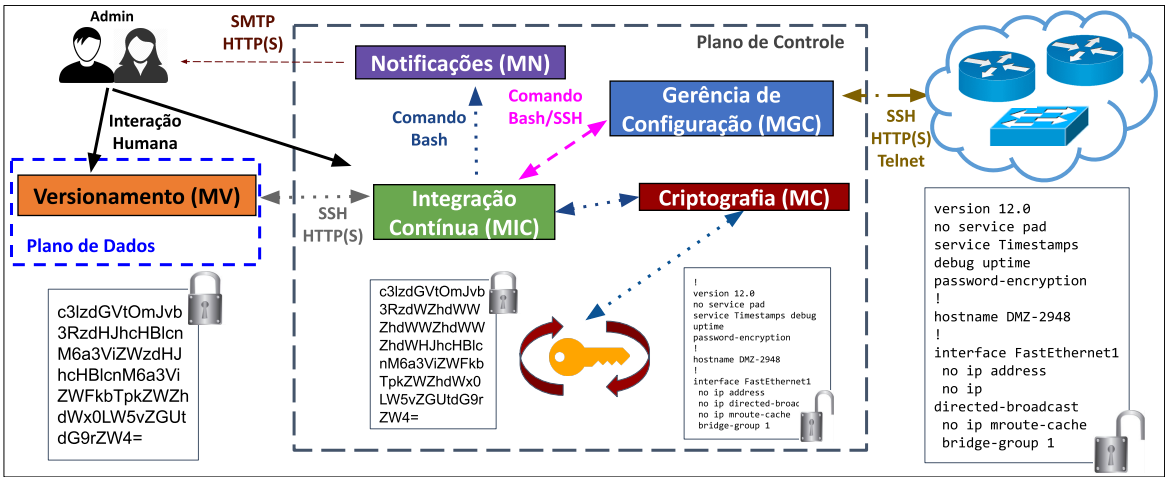


Figura 3 – Arquitetura do PipeConf. Fonte: Elaboração própria.

Pela Figura 3, percebe-se que o MIC recebe os dados para acesso aos ativos a partir do MV. Também é função do MIC enviar o *backup* das configurações dos ativos de rede ao MV, para ser armazenado e versionado. Além disso, o MN recebe do MIC e transmite as notificações aos serviços de envio previamente cadastrados. As notificações podem ser sobre a ocorrência de algum problema na execução de atividades ou sobre o término da execução do *pipeline*. O MC é acionado pelo MIC quando precisa-se criptografar ou descriptografar os arquivos de credenciais e/ou de configuração dos ativos de rede. Por fim, o único módulo da solução que tem acesso direto aos ativos é o MGC, que comunica-se com o MIC para receber as credenciais de acesso aos ativos e as novas configurações a serem aplicadas nos mesmos. O MGC também interage com o MIC para encaminhar os arquivos de *backup* dos ativos.

Os componentes de *software* que constituem o PipeConf, embora já existissem e funcionassem de forma independente uns dos outros, foram utilizados em conjunto para atender as finalidades distintas suportadas na solução. Desta forma, foi possível construir um *pipeline* escrito como código e executado por uma ferramenta de integração, entrega e implantação contínua. O código do *pipeline* é responsável por integrar todas as ferramentas envolvidas resultando em uma solução integrada que proporciona a gerência de configuração automatizada de *switches* e roteadores.

O *pipeline* possibilitou ao PipeConf ser idealizado como uma solução modular, ou seja, que suporta a adição ou remoção de componentes de *software* para atender demandas futuras. Além disso, a abstração da complexidade envolvida na integração de todos os componentes de *software*, cria a impressão, do ponto de vista do usuário, de que trata-se de uma única solução.

O servidor no qual são instalados os módulos MV, MIC, MC e, opcionalmente, MN é denominado **PipeConf-Server**. O servidor no qual é instalado o módulo MGC é denominado **PipeConf-Node**. A Figura 4 exibe os modelos de instalação do PipeConf sugeridos para um ambiente de produção, utilizando o PipeConf-Server e PipeConf-Node. Eles podem ser servidores físicos ou virtuais e podem estar em um único ou múltiplos dispositivos de *hardware*. Para o caso da rede em que o PipeConf tem acesso direto aos ativos, o PipeConf-Server e o PipeConf-Node podem estar no mesmo *hardware*. Caso contrário, seriam necessários dois servidores (físicos ou virtuais) diferentes, um para o PipeConf-Server e outro para o PipeConf-Node, dispostos em redes físicas ou virtuais separadas. Portanto, instalar o PipeConf em dois servidores distintos é indicado para empresas ou organizações que queiram restringir o acesso aos *switches* e roteadores, mantendo-os isolados em redes virtuais privadas diferentes do restante dos ativos.

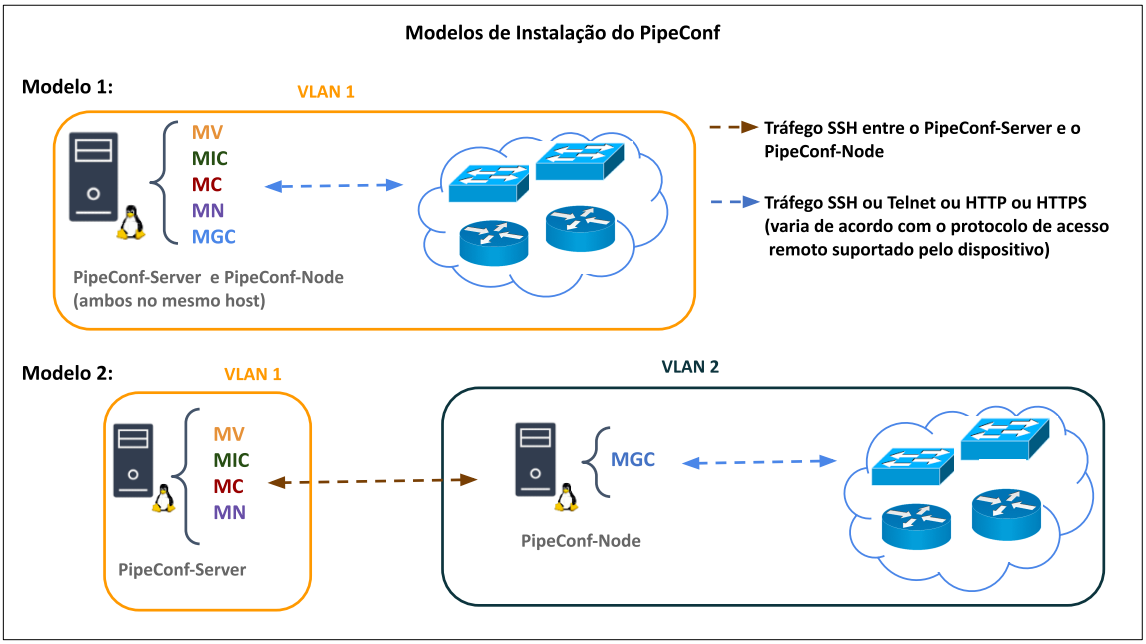


Figura 4 – Modelos de instalação do PipeConf. Fonte: Elaboração própria.

4.2 Justificativa das Ferramentas Selecionadas

A Figura 5 apresenta as ferramentas selecionadas correlacionando com cada módulo da arquitetura. Como critérios de seleção, as ferramentas foram escolhidas de acordo com os recursos mínimos de CPU e memória requeridos, bem como as funcionalidades exigidas em cada módulo.

Como mencionado anteriormente, o PipeConf consiste na integração de várias ferramentas e a seleção ocorreu dentre aquelas descritas na Seção 2.6, são elas: **Salt SProxy** e **Napalm** (para gerência de configuração); **Docker** (para virtualização), **Gogs** (para controle de versão), **Jenkins** (para CI/CD), **Sops** (para edição de arquivos criptografados), **PostgreSQL** (para gerenciamento de banco de dados) e, adicionalmente, um **Servidor SMTP**.

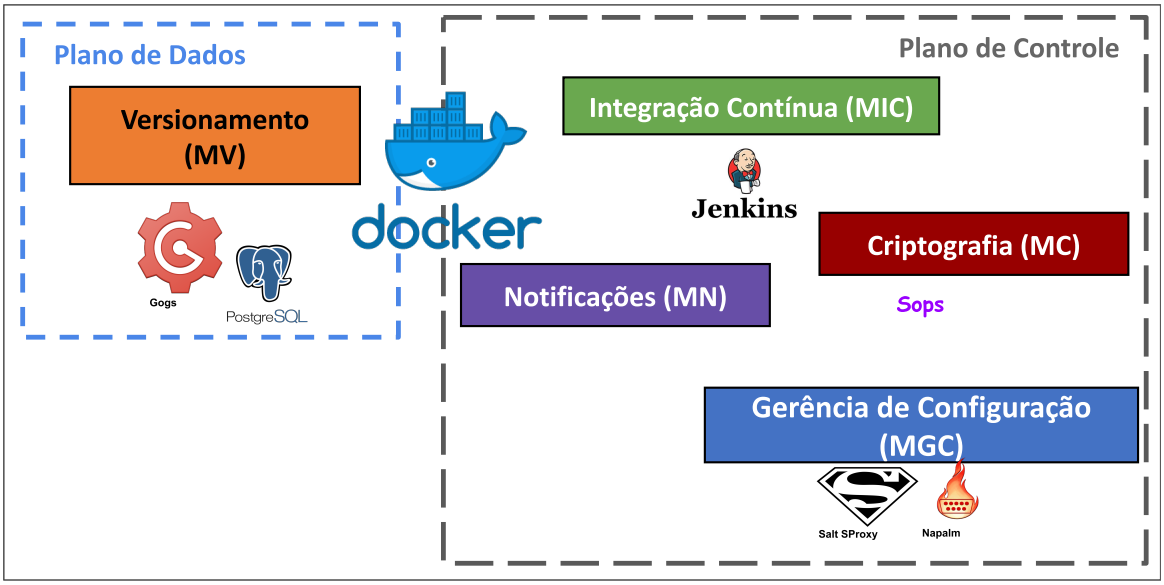


Figura 5 – Correlação das ferramentas com os módulos do PipeConf. Fonte: Elaboração própria.

4.2.1 Ferramentas para Gerência de Configuração

O Salt Super Proxy (ou Salt SProxy) automatiza o gerenciamento da configuração de ativos de rede a partir do SaltStack, porém, sem a necessidade de usar o módulo Salt-Proxy Minion. Logo, possibilita usufruir das funcionalidades do SaltStack sem a sobrecarga de ter de gerenciar um SProxy Minion para cada ativo. Esta ferramenta localiza-se no módulo MGC ilustrado na Figura 5. Uma comparação entre o Salt SProxy e outras ferramentas de gerência de configuração é apresentada na Tabela 2. De acordo com a tabela, é possível observar que as ferramentas Ansible, SaltStack e Salt SProxy possuem características semelhantes.

Tabela 2 – Ferramentas de gerência de configuração.

	Ferramentas de gerência de configuração				
Características	***Salt SProxy***	***SaltStack***	***Puppet***	***Ansible***	***Chef***
Código fonte aberto	✓	✓	✓	✓	✓
Suporte a ativos de rede	✓	✓	✓	✓	✓
Linguagens de programação	Python	Python	Ruby	Python	Ruby
Múltiplos fabricantes	✓	✓	-	✓	-
Suporte a Napalm	✓	✓	-	✓	-
Executa em contêiner	✓	✓	✓	✓	✓

Com o intuito de fomentar a escolha do Salt SProxy como ferramenta de gerenciamento da configuração dos ativos de rede, foi realizada uma análise a partir do código produzido pelas ferramentas Ansible e SaltStack; além da diferença entre o funcionamento do SaltStack e o Salt SProxy. Os trechos de código mostrados a seguir foram gerados por Wilson (2017).

Apresenta-se primeiramente o Código-fonte 1 do arquivo **config.play**, que é utilizado

pela ferramenta Ansible para aplicar uma configuração em dois ativos de rede, definida no arquivo modelo **initial.j2**. Esse trecho tem o mesmo efeito prático que o código utilizado pelo SaltStack mostrado no Código-fonte 2. O detalhe da configuração a ser aplicada nos dois ativos de rede é irrelevante para efeitos de comparação da sintaxe de código entre o Ansible e SaltStack.

Código-fonte 1 Exemplo do arquivo com a sintaxe suportada pelo Ansible para aplicar configurações em um ativo de rede.

```
1  - name: Apply basic config
2    hosts: cisco-cpe
3    tasks:
4      - name: Generate local configuration for hosts
5        local_action: template dest={{ playbook_dir }}/gen-config/initial-{{
             hostname }}.conf src={{ playbook_dir }}/source/initial.j2
6      - name: Gen .diff file (apply change)
7        napalm_install_config:
8              hostname: "{{ host }}"
9              username: "{{ username }}"
10             password: "{{ password }}"
11             dev_os: "{{ dev_os }}"
12             config_file: gen-config/initial-{{ hostname }}.conf
13             commit_changes: True
14             replace_config: True
15             diff_file: initial-{{ hostname }}.diff
16             optional_args: {'auto_rollback_on_error': False}
```

Da mesma maneira, o Código-fonte 2 exibe o conteúdo do arquivo **config.sls**, que exemplifica a aplicação de uma configuração nos mesmos ativos de rede definida no arquivo **router-config.j2**. Novamente, o detalhe da configuração a ser aplicada é irrelevante para efeitos de comparação do código.

Código-fonte 2 Exemplo do arquivo com a sintaxe suportada pelo SaltStack para aplicar configurações em um ativo de rede.

```
1  full_config:
2    netconfig.managed:
3      - template_name: salt://router-config.j2
4      - replace: True
```

A partir dos trechos de código anteriormente apresentados, observa-se que o SaltStack é mais simples que o Ansible para executar a mesma atividade. Esta característica se torna ainda mais importante se considerada o estado da comunicação entre componentes do PipeConf, também gerenciados via *software*. Acredita-se que, quanto menor for a quantidade de texto produzido pela ferramenta, o código gerado estará menos suscetível a erros.

O SaltStack, entretanto, utiliza os componentes Salt Master e Salt Proxy Minion para gerenciar os ativos de rede. Como o Salt SProxy não é um componente interno do SaltStack,

mas sim um *plug-in* associado, ele funciona sem a necessidade de execução dos referidos componentes. A Figura 6 apresenta uma comparação entre as topologias do SaltStack e Salt SProxy. Este último foi escolhido para compor a arquitetura do PipeConf, pois utiliza a mesma sintaxe de comandos e configurações que o SaltStack; além de suportar o Napalm e prover maior escalabilidade com a utilização de menos componentes de *software*, como pode ser observado pela figura.

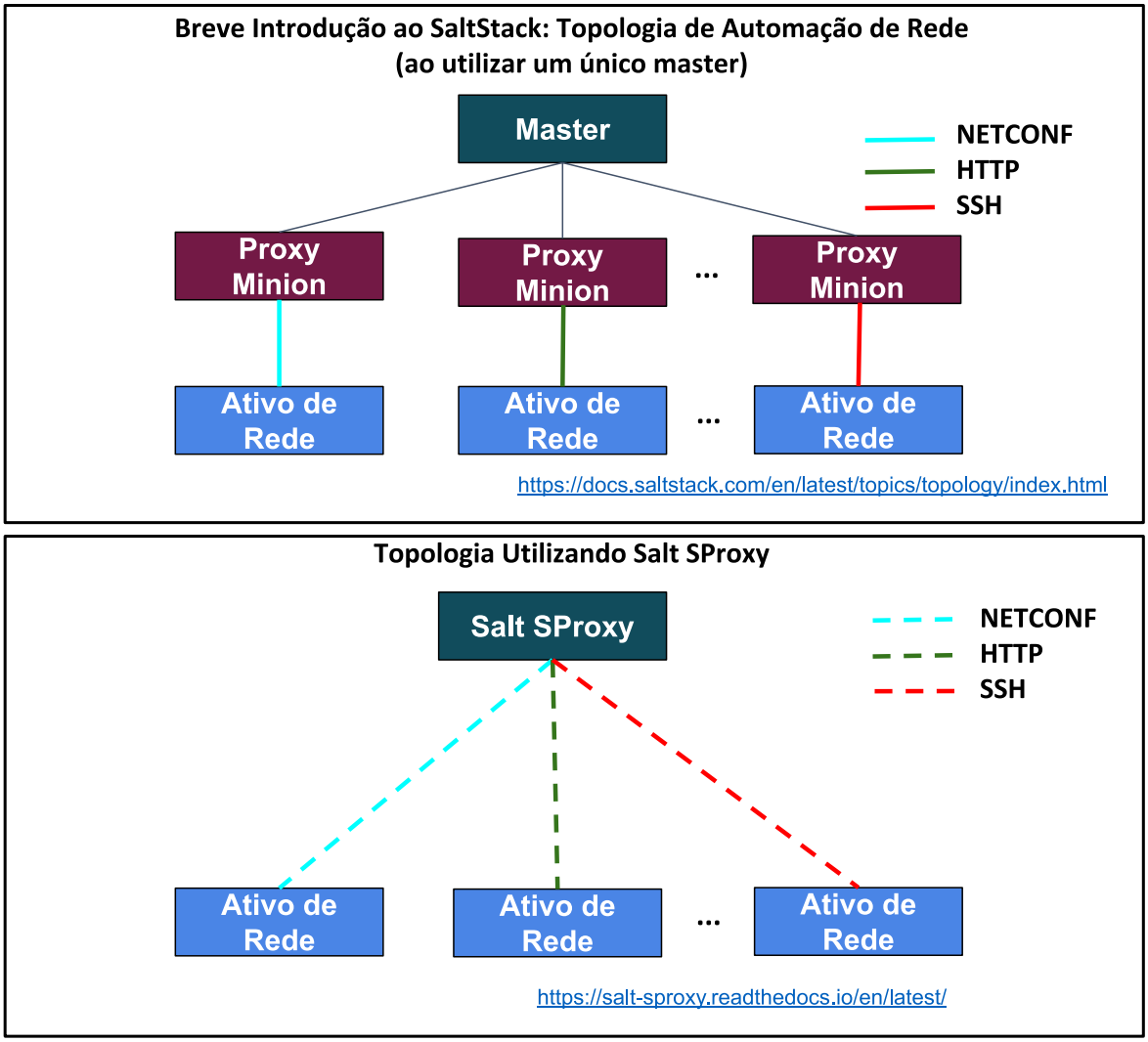


Figura 6 – Comparação entre as topologias de funcionamento do SaltStack e do Salt SProxy. Adaptado de: (ULINIC, 2019).

Com relação ao Napalm, essa biblioteca também está presente no módulo MGC apresentado na Figura 5 e funciona associado ao Salt SProxy. O mesmo possibilita o suporte a diversos modelos e fabricantes de ativos de rede[1]; que executam os seguintes sistemas operacionais: Arista EOS, Juniper JunOS, Cisco {IOS, IOS-XR, NX-OS}, Fortinet FortiOS, Mikrotik RouterOS, Palo Alto NOS, Pluribus NOS e VyOS.

[1] A lista completa de *drivers* suportados, bem como os parâmetros opcionais, podem ser encontrados na documentação do Napalm: https://napalm.readthedocs.io/en/latest/support/index.html.

4.2.2 Ferramenta de Virtualização

O Docker é uma plataforma de virtualização leve e portável, que facilita a criação, a implantação e a execução de aplicativos utilizando contêineres, sem sobrecarregar os recursos computacionais existentes (PIRES; MILITÃO, 2019). O Docker é utilizado por todos os módulos descritos anteriormente, para facilitar a instalação e execução de outros componentes, como por exemplo: Gogs, PostgreSQL (banco de dados suportado pelo Gogs), Jenkins e Salt SProxy. Emilsson (2020) mediu o desempenho entre o Docker, LXD[2], Podman[3] e Buildah[4] em um ambiente controlado e após a análise dos resultados obtidos observou-se que a diferença de performance entre eles é bem pequena. Com isso, o Docker foi selecionado por critérios de familiaridade e experiência em ambientes de produção.

4.2.3 Ferramenta de Controle de Versão

Localizado no MV, o Gogs é responsável pelo versionamento do código do *pipeline*, das informações de acesso a cada ativo de rede, bem como dos arquivos de configuração dos mesmos. A escolha desta ferramenta foi decidida pela comparação com outras similares em relação à quantidade de recursos mínimos de CPU e memória requeridos. Ademais, também foi considerada a capacidade de instalação em um servidor local, caso necessário. Conforme apresenta a Tabela 3, o Gogs requer uma quantidade de recursos de CPU e memória bem menor do que as demais ferramentas. O PostgreSQL foi selecionado como sistema de banco de dados do tipo objeto relacional por ser utilizado de maneira nativa pelo Gogs.

Tabela 3 – Ferramentas web para gerenciamento de repositórios Git.

	Ferramentas			
Características	***Gogs***[5]	***Gitlab***[6]	***Github Enterprise***[7]	***Bitbucket Enterprise***[8]
Código fonte aberto	✓	✓	✓	-
Instalação Local	✓	✓	✓	✓
Memória	64 MB	4 GB	32 GB	3 GB
CPU	0.1 core	4 cores	4 cores	2 cores
Executa em contêiner	✓	✓	-	-

4.2.4 Ferramenta de CI/CD

O Jenkins é a ferramenta responsável pela execução do código do *pipeline* de integração dos componentes da solução. Além disso, está incumbido de exibir o status da execução e

[2] LXD: https://linuxcontainers.org/lxd/introduction/
[3] Podman: https://podman.io
[4] Buildah: https://buildah.io
[5] Gogs: https://github.com/gogs/gogs
[6] Gitlab CE: https://docs.gitlab.com/ce/install/requirements.html
[7] Github: https://enterprise.github.com/faq
[8] BitBucket: https://confluence.atlassian.com/bitbucketserver/supported-platforms-776640981.html

encaminhar as notificações ao MN, que por sua vez as enviará aos usuários interessados. O Jenkins está presente no módulo MIC apresentado na Figura 5 e a sua escolha foi realizada após comparação com o Gitlab, em relação ao uso dos recursos computacionais. Ambas as soluções proveem uma versão gratuita e são utilizadas para CI/CD. Entretanto, como mostra a Tabela 4, o Jenkins requer menos capacidade de memória, além de não necessitar de um banco de dados.

Tabela 4 – Ferramentas para integração, entrega e implantação contínua.

Características	**Ferramentas**	
	Jenkins[9]	***Gitlab***[10]
Código fonte aberto	✓	✓
Interface Web	✓	✓
Instalação Local	✓	✓
Memória	1 GB	4GB
Banco de dados	-	PostgreSQL
Executa em contêiner	✓	✓
Suporte a *pipeline*	✓	✓

4.2.5 Ferramenta para Edição de Arquivos Criptografados

O Sops foi a ferramenta selecionada para edição de arquivos criptografados, tendo em vista que tem suporte a uma quantidade maior de serviços de gerenciamento de chaves criptográficas se comparado a outras opções como: Git-crypt, BlackBox e Transcrypt. Os dados na Tabela 5 demonstram esta comparação. O Sops suporta a utilização de serviços externos de gerência de chaves simétrica e assimétrica, que podem ser utilizados para criptografar ou descriptografar o conteúdo de arquivos de diversos formatos. Neste trabalho, o Sops está localizado no módulo MC e foi configurado para usar uma chave simétrica AES (*Advanced Encryption Standard*) no modo GCM (*Galois Counter Mode*)[11] de 256 bits; armazenada e gerenciada pelo Amazon KMS[12,13].

4.2.6 Servidor SMTP

Opcionalmente, o PipeConf pode utilizar um serviço de SMTP Server para envio de mensagens aos interessados no status das atividades. Este componente está presente no módulo MN, mas não é um componente padrão fixo da solução, podendo ser utilizado opcionalmente uma outra solução externa, desde que o Jenkins esteja adequadamente configurado.

[9] Jenkins: https://www.jenkins.io/doc/book/installing/
[10] Gitlab EE: https://docs.gitlab.com/ee/install/requirements.html
[11] AES-GCM: https://csrc.nist.gov/publications/detail/sp/800-38d/final
[12] AWS-KMS: https://aws.amazon.com/kms
[13] Detalhes técnicos da criptografia simétrica do KMS: https://bit.ly/3qQd5h4

Tabela 5 – Ferramentas de gerenciamento de segredos.

Ferramentas	Criptografia suportada		Serviços de gerenciamento de chaves criptográficas			
	Simétrica	**Assimétrica**	**PGP**	**Amazon KMS**	**Google KMS**	**Azure Key**
Sops	✓	✓	✓	✓	✓	✓
Git-crypt	✓	-	✓	-	-	-
Blackbox	✓	-	✓	-	-	-
Transcrypt	✓	-	✓	-	-	-

4.3 Fluxo de Trabalho

O PipeConf pode ser utilizado para executar atividades relacionadas à gerência da configuração de ativos de redes. Como um primeiro exemplo, pode-se citar o cadastro das informações de acesso a um ou mais ativos. Além disso, pode ser empregado para obter o *backup* da configuração de *switches* e roteadores. Por fim, como sua principal funcionalidade, pode ser utilizado para aplicar alterações na configuração dos ativos em uma rede.

A Figura 7 apresenta o fluxo das atividades realizadas quando a equipe de administradores de rede precisa cadastrar as informações de acesso a um ativo utilizando o PipeConf. As atividades que compõem o *pipeline* de cadastro são as seguintes:

- Um ou mais membros da equipe inicia uma iteração com a interface web do Jenkins para dar início à execução do *pipeline*;
- Durante a execução, o Jenkins aciona o Sops para criptografar as informações de acesso recém cadastradas pelo membro da equipe de administradores de rede;
- Os arquivos são enviados para o Gogs para serem versionados e armazenados;
- Ao final da execução do *pipeline*, o Jenkins envia uma notificação à equipe de administradores e remove os arquivos temporários gerados durante o processo.

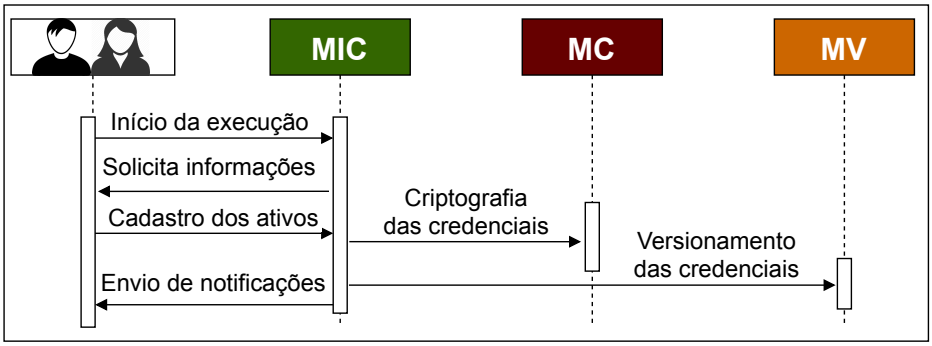


Figura 7 – Fluxo de trabalho para cadastro dos ativos de rede. Fonte: Elaboração própria.

As notificações enviadas pelo PipeConf ao usuário podem ser uma mensagem via *email* e/ou Slack[14]. A notificação contém algumas informações sobre a execução do *pipeline*, bem como um link para visualização do relatório das atividades. A equipe tem a opção de acompanhar em tempo real a execução de todo processo pela interface web do Jenkins.

As informações de acesso aos ativos de rede são armazenadas de forma criptografada. Antes do Jenkins enviar as informações ao Gogs, o *pipeline* aciona o Sops para criptografar as informações. O Código-fonte 3 apresenta um exemplo das configurações de acesso a um ativo criptografadas automaticamente pelo Sops. Esta funcionalidade do PipeConf destaca-se como um diferencial sobre as demais ferramentas de gerência de configuração de ativos de rede.

Código-fonte 3 Exemplo do arquivo criptografado que armazena dado de acesso de um ativo de rede.

```
{
   "data": "ENC[AES256_GCM,data:0z06QqjehuiG=,tag:T54Dopsfyu2BEA==,type:str]",
   "sops": {
      "kms": [
         {
            "arn": "arn:aws:kms:us-east-2:255686512659:key/d38c3af4-e577-4
            634-81b2-26a54a7ba9b6",
            "created_at": "2020-04-23T06:32:25Z",
            "enc": "AQICAHj6EaqIUwHQ7E/BBEr6RCx3wnszmObHV6TxtBq9xDq/G2RQ==",
            "aws_profile": "default"
         }
      ],
      "gcp_kms": null,
      "azure_kv": null,
      "lastmodified": "2020-04-23T06:32:26Z",
      "mac": "ENC[AES256_GCM,data:ujf8sgsatvs=,tag:a3PUjg/x4vC5A==,type:str]",
      "pgp": null,
      "unencrypted_suffix": "_unencrypted",
      "version": "3.5.0"
   }
}
```

A partir do Código-fonte 3, observa-se que as informações de acesso aos ativos podem ser versionadas no Git sem a menor preocupação de ter o conteúdo visualizado por alguém sem a devida autorização. O trecho do Código-fonte 4 foi criptografado pelo Sops resultando no conteúdo do Código-fonte 3. O Código-fonte 4 seria armazenado no Git sem nenhuma criptografia se não fosse a utilização do Sops. Dados sensíveis, tais como endereço IP, usuário, senha e porta do serviço de acesso remoto, ficariam em texto plano e poderiam ser visualizadas facilmente por qualquer pessoa que tivesse acesso ao repositório Git. De posse dessas informações, uma atividade maliciosa poderia ser executada no respectivo ativo de rede.

[14] Slack: https://slack.com/

Código-fonte 4 Exemplo do arquivo que armazena dado de acesso de um ativo de rede sem criptografia.

```
proxy:
  proxytype: napalm
  driver: ios
  host: 192.168.122.79
  username: 'aecio'
  passwd: 'teste'
  optional_args:
    port: 22
    transport: ssh
    secret: ''
    dest_file_system: 'flash:'
```

A Figura 8 apresenta o fluxo das atividades realizadas quando a equipe de administradores de rede precisa obter a cópia de segurança da configuração de um ou mais ativos de rede utilizando o PipeConf. As atividades que compõem o *pipeline* de *backup* são as seguintes:

- Um ou mais membros da equipe inicia uma iteração com a interface web do Jenkins para começar ou agendar um horário no qual o *pipeline* seja executado automaticamente;
- Durante a execução do *pipeline*, o Jenkins aciona o Gogs para obter as informações de acesso aos ativos de rede;
- Como o conteúdo dos arquivos está criptografado, o Sops é acionado para descriptografar as informações de acesso;
- Um contêiner Docker é criado para executar o Salt SProxy;
- O Salt SProxy lê as informações de acesso aos ativos de rede e faz uso da biblioteca Napalm para verificar quais estão acessíveis, com o intuito de obter o *backup* da configuração dos mesmos[15];
- O Sops é novamente acionado para criptografar os arquivos de *backup* da configuração dos ativos, por conterem informações sensíveis do ponto de vista de segurança;
- Os arquivos de *backup* são armazenados e versionados no Gogs;
- Ao final do *pipeline*, o Jenkins envia uma notificação à equipe de administradores, remove os arquivos temporários e finaliza a execução do contêiner do Salt SProxy.

Um ponto importante a ser observado é que os arquivos gerados no *backup* contêm exatamente a sintaxe de comandos nativa do modelo e fabricante para cada ativo de rede. O

[15] O *backup* da configuração é obtido a partir da memória (*running-config*) e do arquivo de inicialização (*startup-config*) de cada ativo.

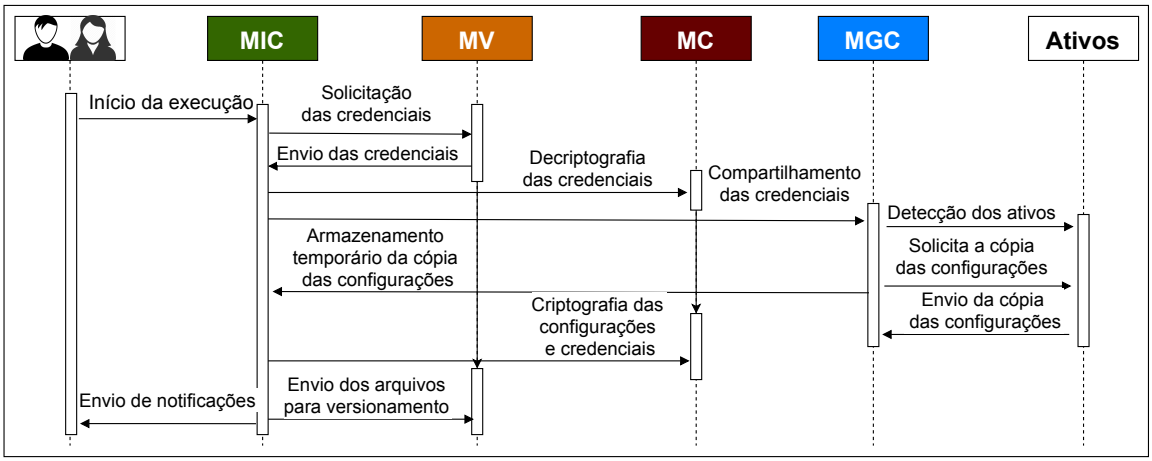


Figura 8 – Fluxo de trabalho para *backup* de configuração dos ativos. Fonte: Elaboração própria.

PipeConf trata a configuração dos ativos como código, podendo ser versionada e possibilitando o rastreamento de mudanças, reversão em caso de problemas, auditoria, padronização, documentação como código e distribuição da configuração a partir de um ponto central. Destaca-se que não é necessário aprender uma linguagem de programação ou a sintaxe de uma ferramenta de gerenciamento de configuração, tais como: Puppet, Ansible, SaltStack ou Chef. A equipe de administradores pode continuar gerenciando os ativos com o conhecimento adquirido ao longo do tempo, sem manter uma forte dependência com os fornecedores de ativos de rede. Esse fluxo de trabalho é recomendado aos profissionais que ainda não se adaptaram (ou estão começando a se adaptar) à abordagem de Infraestrutura como Código.

A Figura 9 apresenta o fluxo das atividades realizadas quando a equipe de administradores de rede precisa aplicar alterações na configuração de um ou mais ativos utilizando o PipeConf. As atividades que compõem o *pipeline* de alteração de configuração são as seguintes:

- A equipe envia as novas configurações dos ativos de rede para o Gogs;
- Ao receber as novas configurações, o Gogs envia uma notificação de evento para o Jenkins, que inicia automaticamente a execução do *pipeline*;
- Após receber as informações de acesso, o Jenkins aciona o Sops para descriptografar os dados;
- Um contêiner Docker é criado para executar o Salt SProxy;
- O Salt SProxy lê as informações de acesso e faz uso do Napalm para obter o *backup* da configuração dos ativos **antes** de executar qualquer alteração;
- Em seguida, o Salt SProxy aplica as novas configurações nos ativos e executa um novo *backup* **depois** das alterações serem realizadas;
- Novamente, o Sops é acionado para criptografar os arquivos de backup da configuração dos ativos, por conterem informações sensíveis do ponto de vista de segurança;

- Os arquivos de *backup* são armazenados e versionados no Gogs;
- Por fim, o Jenkins envia uma notificação à equipe de administradores, remove os arquivos temporários e finaliza a execução do contêiner com o Salt SProxy.

Com o fluxo descrito acima, é importante destacar o duplo processo de *backup* das configurações, antes e depois das alterações. Isso possibilita que a equipe de administração possa ter acesso ao estado de cada ativo antes e depois das mudanças serem aplicadas, permitindo a realização da reversão (*rollback*) da configuração.

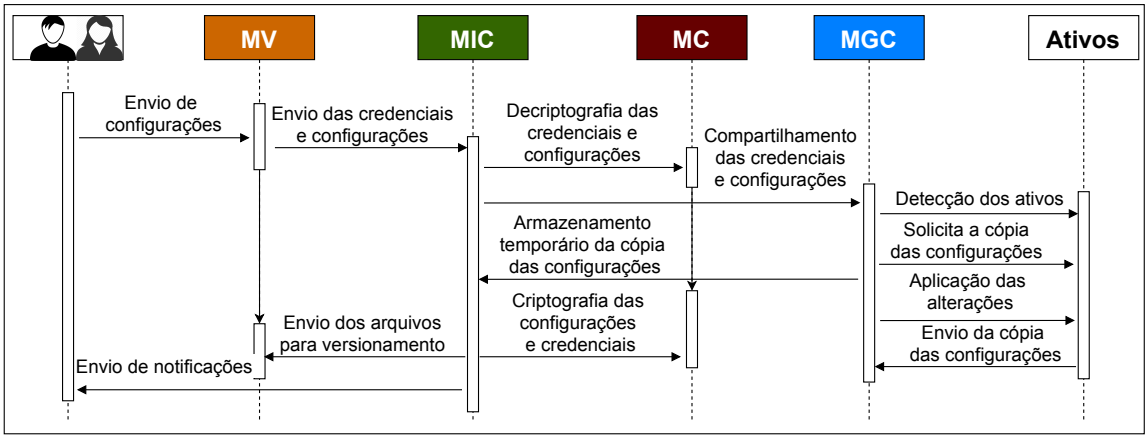


Figura 9 – Fluxo de trabalho para atualização das configurações. Fonte: Elaboração própria.

4.4 Pré-requisitos de Funcionamento

Os pré-requisitos para o funcionamento do PipeConf[16] são listados a seguir, divididos em três categorias específicas: configuração dos ativos de rede, pré-requisitos de *hardware* e liberação de portas no *firewall*.

4.4.1 Configuração dos Ativos de Rede

Para acessar e gerenciar cada ativo de rede, o PipeConf necessita que cada ativo esteja previamente conectado à rede. A equipe de administradores de rede também precisa cadastrar as configurações de acesso para cada ativo no PipeConf. As seguintes informações de acesso são requeridas:

- Nome do ativo de rede;
- Tipo do ativo de rede, se *switch* ou roteador;
- O endereço IPv4 ou o nome DNS do ativo de rede;

[16] Instruções de instalação disponíveis em: <https://gitlab.com/aeciopires/pipeconf/-/tree/master/docs>.

- Nome do *driver* correspondente ao fabricante do ativo, como por exemplo, **ios** para ativo Cisco-IOS, **junos** para Juniper ou **panos** para Palo Alto[1];
- Credenciais de acesso (nome de usuário e senha) ao ativo com permissão de administrador;
- Tipo de acesso remoto SSH, Telnet, HTTP ou HTTPS, que pode variar de acordo com o modelo e fabricante do ativo;
- Porta relacionada ao serviço de acesso remoto, como por exemplo, 22/TCP para o SSH; 23/TCP para o Telnet, 80/TCP para o HTTP e 443/TCP para o HTTPS;
- A senha para acessar o modo administrador do ativo de rede, que pode ser igual ou diferente da senha de acesso remoto.

4.4.2 Requisitos de *Hardware*

O PipeConf pode ser instalado em servidores físicos ou virtuais. Também é necessário que haja conectividade entre o(s) servidor(es) do PipeConf e os ativos de rede a serem gerenciados. A Tabela 6 exibe os requisitos de *hardware* para executar o PipeConf.

Tabela 6 – Requisitos de *hardware* do PipeConf.

	Tipos de Servidor	
Requisitos	***PipeConf Server***	***PipeConf Node***
Distribuição GNU/Linux	Ubuntu | CentOS | Debian	Ubuntu | CentOS | Debian
Memória	2 GB	1 GB
CPU	1 core	1 core
HD	10 GB	10 GB

Conforme mostrado na Figura 4, o PipeConf pode ser instalado em apenas um servidor no caso do PipeConf-Server ter acesso direto à rede na qual estão os *switches* e roteadores. Caso contrário, serão necessários dois servidores (físicos ou virtuais), um para executar os módulos instalados no PipeConf-Server e outro para executar o módulo instalado no PipeConf-Node, podendo estes servidores estarem em redes físicas ou virtuais separadas. A especificação de *hardware* do PipeConf-Server na Tabela 6 atende aos dois modelos de instalação citados na Figura 4.

4.4.3 Liberação de Portas no *Firewall*

Por motivos de segurança, o PipeConf não precisa estar acessível na Internet. Uma lista de portas específicas pode ser liberada no *firewall* da rede interna (se houver), permitindo que a equipe de administradores de rede acesse a solução. Para tal, as seguintes portas com destino ao PipeConf-Server devem ser liberadas:

- 22/TCP: com destino também ao PipeConf-Node, para acessar o Jenkins utilizando o protocolo SSH;
- 80 ou 443/TCP: para acessar o Jenkins utilizando o protocolo HTTP/HTTPS);
- 81 ou 444/TCP: para acessar o Gogs utilizando o protocolo HTTP/HTTPS;
- 10022/TCP: para acessar o Gogs utilizando o protocolo SSH.

4.5 Aplicabilidade da Solução Proposta

O PipeConf pode gerenciar a configuração de *switches* e roteadores em uma LAN (*Local Area Network*), MAN (*Metropolitan Area Network*) ou mesmo WAN (*Wide Area Network*), desde que haja conectividade entre os ativos e o(s) servidor(es) que hospeda(m) a solução. Portanto, independente do tipo de rede na qual está gerenciando ativos, o PipeConf apresenta as seguintes funcionalidades:

- Interface web para cadastro dos ativos de rede a serem gerenciados;
- Versionamento da configuração de ativos de redes;
- *Backup* sob-demanda ou agendado da configuração de *switches* e roteadores;
- Aplicação automatizada e simultânea da configuração em múltiplos ativos de rede;
- Identificação dos ativos de rede que estão acessíveis ou não durante a aplicação de novas configurações ou obtenção do *backup*;
- Encriptação/decriptação das informações de acesso aos ativos;
- Encriptação/decriptação do *backup* das configurações dos ativos;
- Visualização das atividades em tempo real por meio de uma interface web;
- Capacidade de envio de notificações por *email* e Slack, para uma ou mais pessoas;
- Execução do *pipeline* sob-demanda ou de forma agendada.

Com essas funcionalidades, o PipeConf pode ser indicado para utilização em diversos casos de uso. Por exemplo, a solução pode ser utilizada em laboratórios de redes de computadores, com o intuito de auxiliar no ensino de disciplinas de administração de redes de computadores. O PipeConf pode ainda ser utilizado em demonstrações relacionadas à gerência de uma rede, evidenciando a abordagem de Infraestrutura como Código. Outro cenário recomendado é a utilização em ambientes de produção de empresas e organizações, para administrar a configuração dos ativos de rede. Com isso, busca-se diminuir o tempo de aplicação de novas configurações,

facilitar a reversão de configurações, favorecer auditorias, aumentar a produtividade da equipe de administradores, reduzir o tempo de inatividade dos ativos devido a erros de configuração e reduzir o impacto dos efeitos da gerência da rede no negócio ou atividade fim da empresa.

Neste capítulo foi detalhada a solução proposta para realizar o gerenciamento da configuração de ativos de rede heterogêneos, composta por um conjunto de ferramentas existentes, mas que foram integradas para utilizar uma abordagem de Infraestrutura como Código. Em seguida, será apresentada uma validação do PipeConf mediante a execução de determinadas análises, com o intuito de investigar critérios de eficiência e eficácia da ferramenta.

5 VALIDAÇÃO DA SOLUÇÃO PROPOSTA

Para validar o desempenho da solução proposta, este capítulo apresenta uma avaliação quantitativa da solução. São apresentadas as análises de desempenho e perfilamento, a metodologia utilizada, a definição das métricas analisadas e a descrição do *hardware* e *software* utilizados em cada análise, bem como os resultados obtidos. Primeiramente, é apresentada uma visão geral sobre a avaliação quantitativa utilizada nesta pesquisa. Em seguida é descrito o ambiente de experimentação utilizado nas análises. Logo após, são apresentados as análises de desempenho e perfilamento contendo a descrição das atividades realizadas e os resultados obtidos. Por fim, é apresentada uma análise qualitativa no código da solução proposta e do código gerado pela mesma com o objetivo de detectar a possível presença de vulnerabilidades de segurança propostas por Rahman, Parnin e Williams (2019).

5.1 Descrição da Avaliação

Salienta-se que a validação realizada neste trabalho não está relacionada com um processo formal de verificação de *software*, mas sim com a validação da hipótese de que o PipeConf é capaz de gerenciar ativos de rede de maneira eficiente e eficaz. Nesta pesquisa, a **eficiência** do PipeConf é caracterizada como sendo a capacidade de executar um mesmo conjunto de atividades em um número crescente de ativos de rede, apresentando um aumento linear no tempo de execução das atividades. Outro fator importante associado à eficiência do PipeConf diz respeito à heterogeneidade dos ativos. Por sua vez, a **eficácia** é definida nesta pesquisa como sendo a capacidade do PipeConf de executar atividades definidas pelo time de administração dos ativos de rede executando e produzindo um código com pouca ou nenhuma vulnerabilidade de segurança.

Primeiramente, realizou-se uma avaliação quantitativa com o intuito de demonstrar a eficiência do PipeConf. Uma avaliação empírica de desempenho foi realizada a partir de atividades específicas de uso, através das metodologias de **experimentação** e **medição** propostas por MetodologiaCientífica.org (2019). Utilizando-se dessas metodologias é possível compreender o comportamento da solução mediante fatores pré-definidos de entrada; para medir e analisar os resultados obtidos a partir de determinadas métricas a serem coletadas. Mais especificamente, busca-se comparar o desempenho da execução de um conjunto de atividades perante 2 aspectos importantes: **celeridade** (quão rápida é a execução) e **escala** (quantidade de ativos). Considerando esses 2 aspectos, os principais objetivos da avaliação quantitativa de desempenho são:

- Analisar o tempo total de execução do PipeConf ao realizar um conjunto de atividades em uma quantidade significativa de ativos, bem como analisar o consumo de recursos como CPU e memória;

- Comparar o tempo de execução de um mesmo subconjunto de atividades realizadas pelo PipeConf e por uma ferramenta existente no mercado que também ofereça suporte a ativos de rede heterogêneos;
- Caracterizar o tempo de execução de cada atividade realizada pelo PipeConf para identificar pontos de "gargalo" no seu funcionamento.

Com relação aos critérios de qualidade do *software* produzido, diversos aspectos podem ser considerados para uma avaliação qualitativa do PipeConf. A princípio, considerou-se avaliar dois desses aspectos importantes, um relacionado à **usabilidade** e outro à **segurança** da ferramenta. Contudo, o fator usabilidade será motivo de trabalhos futuros a serem realizados experimentos com usuários reais. Portanto, o critério de eficácia do PipeConf demonstrado neste trabalho considera a avaliação da presença de falhas de segurança no código gerado.

A Seção 5.2 apresenta maiores detalhes sobre o ambiente configurado para as experimentações por meio dos componentes de *software* e *hardware* utilizados, bem como as métricas selecionadas para avaliação de desempenho. Na Seção 5.3, duas análises quantitativas são descritas e os respectivos resultados são apresentados. A Seção 5.4 apresenta os resultados de uma análise qualitativa no código da solução e do código gerado, com o objetivo de detectar a possível presença de vulnerabilidades de segurança propostas por Rahman, Parnin e Williams (2019).

5.2 Ambiente de Experimentação

Para realizar a avaliação do PipeConf, foi implementada uma Prova de Conceito (PoC, do inglês *Proof of Concept*) baseada em um conjunto de configurações descritas como código[1]. Estas configurações são aplicadas em ativos de rede simulados, buscando alterar a configuração corrente desses ativos, executar um *backup* e versioná-lo em um repositório de código para possibilitar a rastreabilidade das mudanças e a reversão de alguma configuração indesejada.

O ambiente de avaliação consiste em ativos de rede simulados, tendo em vista que a utilização de ativos físicos reais requer, não apenas um espaço físico apropriado para as instalações, mas também um alto investimento financeiro. Para contornar essas restrições, o uso de ferramentas de simulação de ativos de rede é eficiente, pois simula com precisão o comportamento e a execução de funcionalidades reais dos ativos, a um custo acessível. O GNS3[2] é uma das ferramentas de simulação de ambiente de rede mais utilizadas no meio acadêmico e foi utilizado como ferramenta de simulação dos ativos de rede nesta pesquisa. A partir do GNS3, o ativo de rede definido para ser gerenciado nas análises de desempenho e perfilamento é o roteador Cisco modelo C3640-IK9O3S-M.

[1] Configurações a serem aplicadas como código pelo PipeConf: https://bit.ly/3cumq9t
[2] GNS3: https://gns3.com

Para coletar as métricas de consumo de CPU e memória do PipeConf foram utilizadas as ferramentas Prometheus[3], Grafana[4], Node Exporter[5] e o CAdvisor[6]. Elas foram selecionadas por critérios de familiaridade e experiência em ambientes de produção, pois não é o foco da pesquisa analisar quais ferramentas são mais eficientes para a coleta das métricas de CPU e memória. Os procedimentos de instalação dessas ferramentas no ambiente de experimentação estão disponíveis na documentação do PipeConf[7].

As configurações dos sistemas computacionais utilizados para executar todos os componentes do PipeConf, do GNS3 e os ativos de rede simulados, bem como as versões dos componentes de *software* utilizados, foram as seguintes:

- PipeConf-Server e PipeConf-Node (ambos executados em um computador local):
  - *Notebook* para executar os planos de dados e de controle do PipeConf;
  - Sistema operacional Ubuntu Desktop 18.04 64 bits;
  - Processador Intel(R) Core(TM) i5-4210U de 1.70GHz com 4 núcleos;
  - Memória de 8GB;
  - SSD SATA III de 480GB com taxas de 540MB/s para leitura e 500MB/s para escrita;
- Servidor GNS3 (executado na nuvem para simular os ativos de rede):
  - Instância na *Google Cloud Platform*[8] (GPC);
  - Sistema operacional Ubuntu Minimal 18.04 64 bits;
  - Processador Intel(R) Xeon(R) de 2.0GHz com 24 núcleos;
  - Memória de 40GB;
  - SSD SATA III de 30GB com taxas de 540MB/s para leitura e 500MB/s para escrita.
- Versões dos componentes de *software* do PipeConf:
  - Docker Community Edition 19.03.13;
  - Gogs 0.12.3;
  - PostgreSQL 12.3;
  - Jenkins 2.263.1;
  - Salt SProxy 2020.7.0;
  - Napalm 3.0.1;
  - Sops 3.6.0.

[3] Prometheus: https://prometheus.io
[4] Grafana: https://grafana.com
[5] Node Exporter: https://github.com/prometheus/node_exporter
[6] CAdvisor: https://github.com/google/cadvisor
[7] Monitorando o PipeConf: https://bit.ly/2KeRM8b
[8] GCP: https://cloud.google.com/compute/vm-instance-pricing

- Versões dos componentes de *software* utilizados na coleta de métricas do PipeConf:

  - Prometheus 2.24.1;

  - Grafana 7.3.7;

  - Node Exporter 1.0.1;

  - CAdvisor 0.36.0.

A topologia do ambiente de experimentação é mostrada na Figura 10. Nesse ambiente, o PipeConf-Server e o PipeConf-Node são executados no mesmo computador e obtém acesso direto aos ativos de rede, mesmo que estes estejam simulados em um servidor remoto. A comunicação entre os dois computadores ocorre por meio da Internet sem uso de VPN (*Virtual Private Network*), em que há a possibilidade da perda de pacotes e variação na latência. Estes fatores também estão presentes em um ambiente de produção e podem alterar o tempo médio de execução das atividades, além do comportamento da solução durante as análises. Contudo, considera-se que essa influência foi minimizada nas análises realizadas dos resultados, com a utilização de técnicas estatísticas apropriadas. O GCP foi escolhido para hospedar o servidor utilizado na simulação dos ativos de rede por apresentar uma melhor relação financeira de custo-benefício em comparação com a AWS[9] e Microsoft Azure[10].

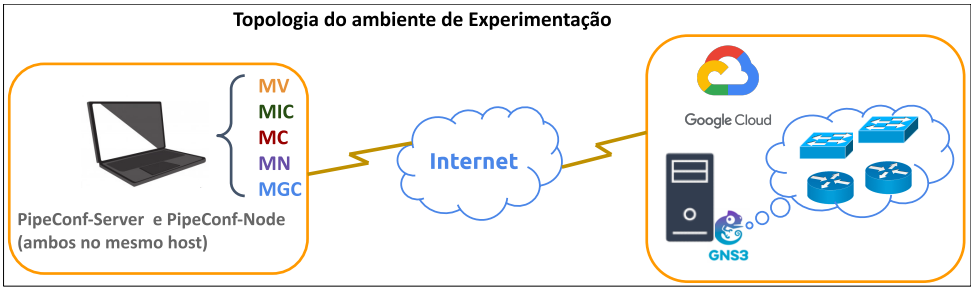


Figura 10 – Topologia do ambiente de experimentação. Fonte: Elaboração própria.

5.3 Avaliação Quantitativa de Desempenho

Esta seção apresenta duas análises quantitativas em termos da **Desempenho** do sistema e do **Perfilamento dos tempos de execução**. As duas próximas subseções descrevem os objetivos relacionados a cada uma dessas análises. Primeiramente, na Subseção 5.3.1, detalha-se o tempo médio de execução das atividades do PipeConf em resposta ao aumento no número de ativos configurados de forma simultânea. Além disso, métricas de monitoramento do sistema foram coletadas para investigar o consumo de recursos dos componentes que compõem a solução. Logo, pretende-se mostrar o quão eficiente é o PipeConf em realizar suas atividades em um grande número de ativos. De forma complementar, a análise descrita na Subseção 5.3.2 tem o propósito

[9] AWS-EC2: https://aws.amazon.com/pt/ec2/pricing/on-demand/

[10] Microsoft Azure: https://azure.microsoft.com/pt-br/pricing/details/virtual-machines/linux/

de caracterizar o tempo de execução de cada atividade realizada ao longo do *pipeline*, permitindo assim um perfilamento da execução do PipeConf em busca de pontos de "gargalo" no sistema. Uma análise comparativa entre o PipeConf e o Unimus[11] foi realizada e está descrita também nesta última subseção.

Para formalizar as atividades da execução do PipeConf, foi definida uma configuração genérica (alterar o endereço de um servidor NTP), que pode ser realizada nos *switches* e roteadores de diferentes modelos e fabricantes. Dessa forma, acredita-se que os resultados podem ser facilmente replicados e o fator que realmente influencia os resultados é a quantidade de ativos de rede. A principal métrica como critério de celeridade da solução proposta é o **tempo médio de execução** das atividades. As atividades realizadas pelo PipeConf para configurar esta atividade de configuração são mostradas na Tabela 7.

Tabela 7 – Descrição das atividades executadas pelo PipeConf.

Atividades	Descrição
AT01	Obter do repositório Git os arquivos de credenciais e configuração.
AT02	Realizar descriptografia dos arquivos de credenciais e configuração.
AT03	Efetivar a detecção dos ativos conectados.
AT04	Obter informações sobre o modelo de cada ativo conectado.
AT05	Fazer *backup* da memória (*running-config*) antes da alteração.
AT06	Fazer *backup* da inicialização (*startup-config*) antes da alteração.
AT07	Alterar configuração do servidor NTP (a.ntp.br).
AT08	Fazer *backup* da memória (*running-config*) depois da alteração.
AT09	Fazer *backup* da inicialização (*startup-config*) depois da alteração.
AT10	Aplicar criptografia nos arquivos de *backup*.
AT11	Versionar os arquivos de *backup* no Git.
AT12	Enviar notificações da execução do PipeConf aos administradores.

Nas duas análises quantitativas propostas, a avaliação consistiu em medir o tempo médio de execução do PipeConf à medida que o número de ativos configurados dobra de 1 até 128. O tempo de execução foi computado a partir do início da primeira atividade (**AT01**) até a conclusão da última (**AT12**). Cada experimento foi executado 30 vezes com o intuito de prover uma confiabilidade estatística mínima, na qual foi utilizado um nível de confiança de 95% para a média.

O consumo dos recursos do sistema utilizados pelo PipeConf foi coletado por meio de medições no computador que o mesmo se encontra instalado. A coleta das métricas de consumo de recursos foi realizada de forma paralela às atividades executadas pelo PipeConf, com intervalo de 60 segundos entre as mesmas.

Nos resultados apresentados a seguir, não são considerados o tempo necessário para instalação e configuração dos componentes de *software* e *hardware* envolvidos, bem como o tempo para configurar os pré-requisitos de funcionamento do PipeConf citados no Capítulo

[11] Unimus: https://unimus.net

4. Portanto, parte-se do pressuposto de que cada ativo de rede simulado já está devidamente inicializado e acessível via SSH pelo PipeConf-Server e PipeConf-Node.

5.3.1 Análise de Desempenho

A Figura 11 apresenta diagramas de caixa que ilustram a distribuição dos tempos (em minutos) que o PipeConf precisou para executar todas as atividades definidas anteriormente, levando em consideração as 30 execuções de cada experimento, e em função do aumento exponencial de ativos configurados de 1 até 128. Este tipo de gráfico fornece uma análise visual da posição, dispersão, simetria, caudas e valores discrepantes (*outliers*) acerca do conjunto de dados obtidos. Os pontos em vermelho indicam o valor médio obtido em cada conjunto de dados. Além disso, pode-se comparar as médias e os respectivos intervalos de confiança também presentes na figura. A Tabela 8 sumariza os valores absolutos dos dados estatísticos obtidos.

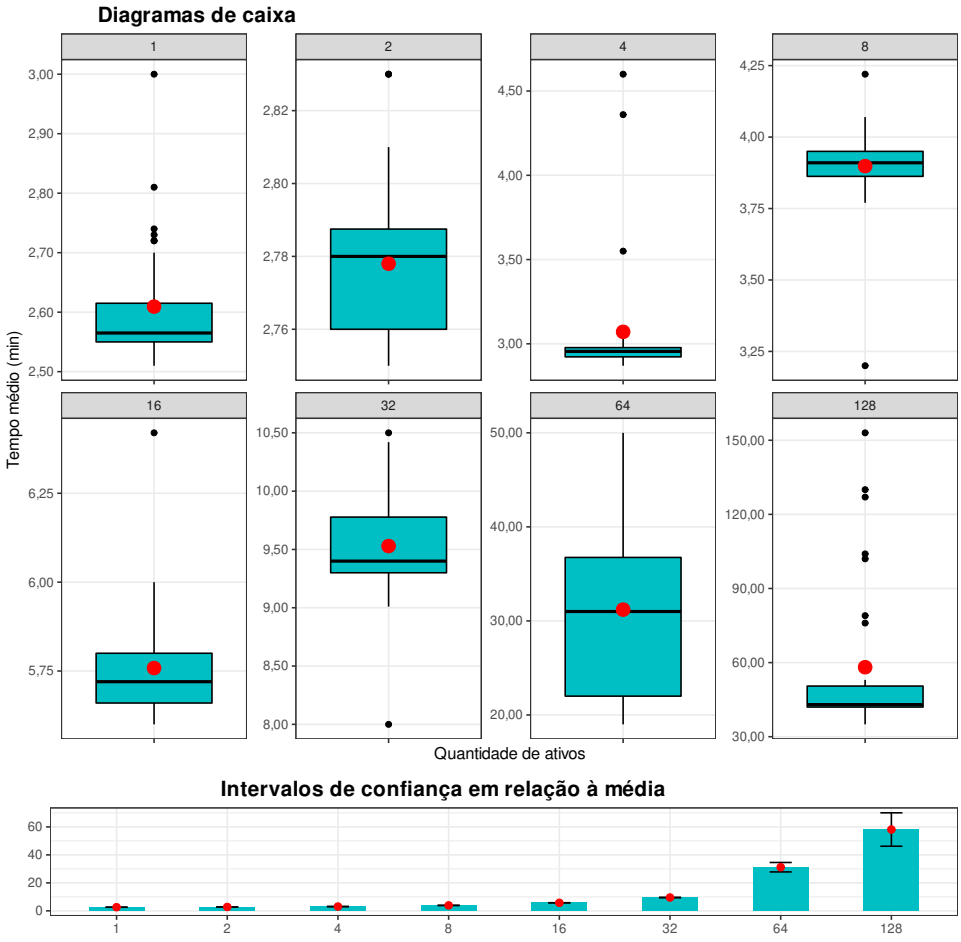


Figura 11 – Dados do Pipeconf referentes aos tempos de configuração em função do número de ativos gerenciados. Fonte: Elaboração própria.

Ao analisar a Figura 11 de forma geral, observa-se que a dispersão dos dados é baixa, uma vez que o *intervalo interquartil* (IIQ) é inferior a um minuto para quase todos os dados, exceto para os conjuntos com 64 (14,75 minutos) e 128 (8,5 minutos) ativos. Considera-se que esta

Tabela 8 – Dados estatísticos dos tempos de configuração do PipeConf.

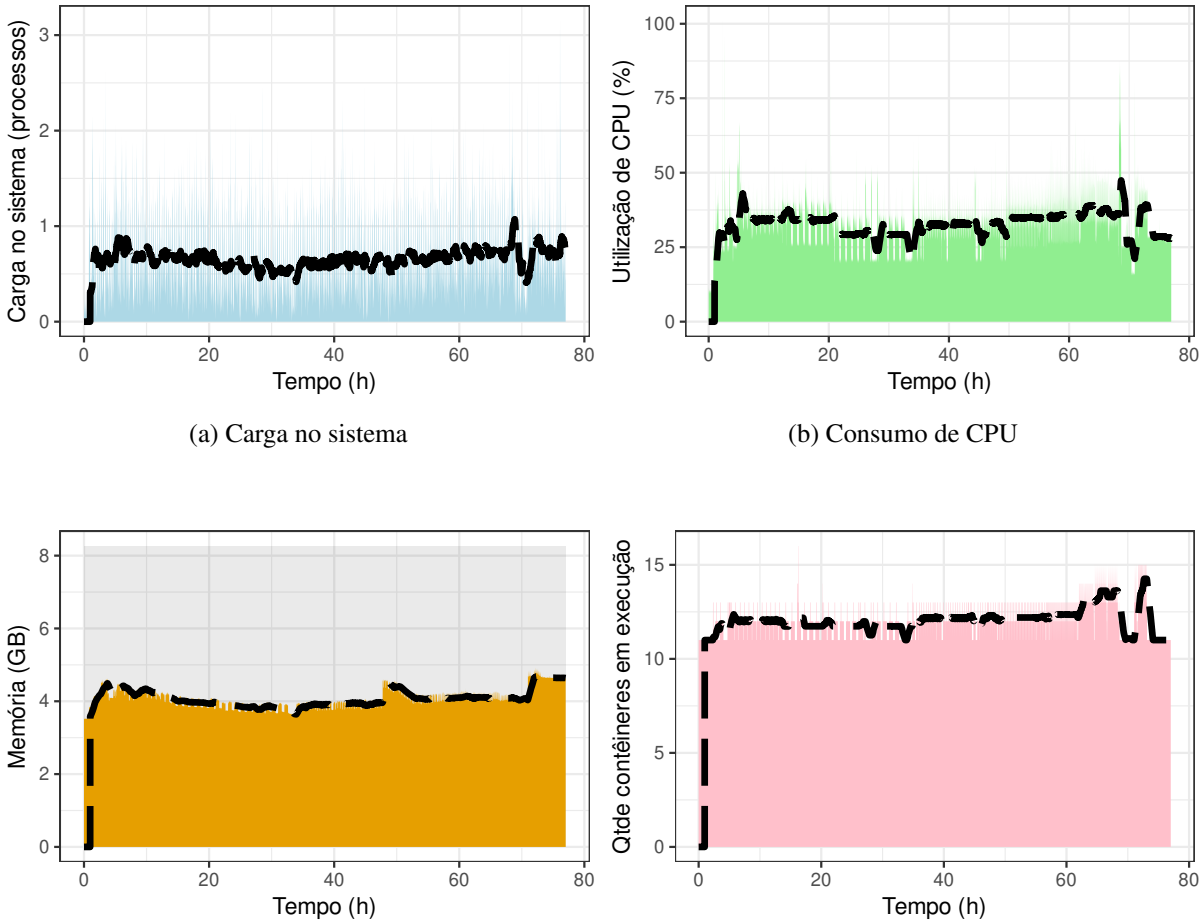
Medidas (minutos)	**Número de ativos**							
	1	**2**	**4**	**8**	**16**	**32**	**64**	**128**
1º Quartil	2,55	2,76	2,92	3,86	5,66	9,30	22,00	42,00
Mediana	2,56	2,78	2,96	3,91	5,72	9,40	31,00	43,00
Média	2,61	2,78	3,07	3,90	5,76	9,53	31,20	58,13
3º Quartil	2,62	2,79	2,98	3,95	5,80	9,78	36,75	50,50
Desvio padrão	0,11	0,02	0,40	0,16	0,16	0,47	9,01	31,98

dispersão também é baixa para 128 ativos em relação ao tempo médio total de configuração dos mesmos. A dispersão é considerada alta para os dados com 64 ativos, muito embora não impacte na análise do tempos médio detalhado mais adiante. Ainda de forma geral, os gráficos apontam que os dados não são enviesados, pois não há uma grande diferença entre os valores médios e medianos, com uma única exceção para o conjunto de 128 ativos. Contudo, esta diferença também não traz impacto significativo nas análises adiante envolvendo intervalos de confiança para a média. Esta análise indica um comportamento aceitável nos dados obtidos, trazendo uma maior segurança na análises subsequentes onde assume-se que os efeitos de aspectos externos aos experimentos foram mitigados.

De maneira mais específica, o tempo médio gasto pelo PipeConf para realizar as atividades aumenta com o número de ativos configurados, como esperado. Contudo, o aumento no tempo médio não segue a mesma dimensão do acréscimo na quantidade de ativos, indicando um ganho proporcional à medida que mais dispositivos são configurados. Por exemplo, o tempo médio de configurar 128 ativos foi menor do que uma hora e não é 128 vezes maior do que configurar um único ativo. Este comportamento é intuitivo e se deve ao fato do PipeConf conseguir executar algumas atividades em paralelo. O entendimento de como essas atividades são paralelizadas também é importante para encontrar possíveis pontos de melhoria do sistema.

Com o intuito de comprovar a escalabilidade do sistema, foram coletadas métricas de utilização dos recursos computacionais utilizados pelo PipeConf ao executar todos experimentos realizados neste trabalho. A Figura 12 destaca quatro destas métricas, que foram monitoradas a cada 60s de um período de aproximadamente 77h de execução de todos os experimentos, no *notebook* onde os componentes do PipeConf foram executados. A Figura 12(a) representa a carga no sistema; as Figuras 12(b) e (c) demonstram respectivamente os consumos de CPU e memória; e a Figura 12(d) ilustra a quantidade de contêineres em execução no sistema, ao longo de todo o período de experimentação. A linha preta em destaque nos gráficos indica o valor da média móvel para a respectiva métrica. As configurações do *notebook* utilizado para gerenciar as configurações dos ativos com o PipeConf estão descritas na Seção 5.2.

De uma maneira geral, a Figura 12 demonstra que o consumo dos recursos do sistema onde o PipeConf foi hospedado permanece estável ao longo de toda a execução dos experimentos, independentemente da quantidade de ativos configurados pelo mesmo. A carga ilustrada na



(a) Carga no sistema

(b) Consumo de CPU

(c) Consumo de memória

(d) Número de contêineres

Figura 12 – Variação da média móvel a cerca das métricas gerais do consumo de recursos pelo PipeConf durante 77h de execução dos experimentos. Fonte: Elaboração própria.

Figura 12(a) representa a quantidade de processos nos estados de execução ou espera no sistema. Simplificadamente, uma CPU contendo apenas um núcleo pode lidar com um processo por vez. Uma carga média de 1 significa que um núcleo está ocupado 100% do tempo. Se a média da carga cair para 0,5, a CPU ficou ociosa por 50% do tempo. Caso a média suba para 1,5, a CPU estava ocupada o tempo todo enquanto havia (em média) um outro processo esperando 50% do tempo. A regra geral é que a média de carga do sistema não deve exceder o número de processadores lógicos presente no computador (KREEFTMEIJER, 2018). Como o *notebook* utilizado para executar o PipeConf possui 4 núcleos, uma carga estável deve ficar abaixo de 4.

Quanto ao consumo de CPU e memória, observa-se também uma regularidade no uso destes recursos nas Figuras 12(b) e (c); com índices quase sempre inferiores à metade do total disponível e poucos picos esporádicos ao longo do tempo. Por fim, a Figura 12(d) apresenta o número de contêineres Docker utilizados na operação do PipeConf, que variou em torno de 12 unidades. Esse comportamento ocorre porque o PipeConf inicia sob demanda um contêiner para executar o Salt SProxy, que por sua vez realiza as atividades **AT03** a **AT09** citadas na Tabela

7. Ao final da **AT09**, o contêiner é encerrado e isso faz com que sejam liberados os recursos de CPU e memória RAM. Esse ciclo acontece durante cada uma das 30 repetições, independente de estar gerenciando 1 ou mais ativos de rede. Destaca-se que, mesmo quando o PipeConf gerenciou mais de um ativo de rede, o número de contêineres não aumentou e isso indica um reaproveitamento de recursos de CPU e memória para gerenciar vários ativos simultaneamente. Dessa forma, comprova-se a característica de eficiência do PipeConf no uso dos recursos.

5.3.2 Análise de Perfilamento

Os dois principais objetivos desta análise são: (i) caracterizar os tempos de execução das atividades referentes ao PipeConf e (ii) analisar comparativamente o funcionamento do mesmo com o Unimus. Com a caracterização proposta foi possível perfilar o tempo de realização das atividades em busca de pontos de melhoria no sistema. A análise comparativa também possibilitou identificar um ponto de gargalo na execução do *pipeline*, conforme detalhado mais adiante. Contudo, a busca por uma ferramenta para comparar com o PipeConf não foi uma tarefa trivial e esse processo está descrito no Apêndice A.

Resumidamente, não foi encontrada uma solução (de mercado ou acadêmica) que realize todas ou boa parte das tarefas executadas pelo PipeConf. Além disso, fatores como a ausência de licenças gratuitas, o curto tempo de avaliação e a falta de usabilidade se mostraram impeditivos para o uso de grande parte das ferramentas pesquisadas. Das 10 soluções investigadas, apenas o **Unimus** se mostrou viável, tendo em vista que a equipe de desenvolvedores gentilmente disponibilizou, para fins desta pesquisa, uma licença temporária para a configuração de até 32 ativos simultâneos (a licença padrão permite apenas 5). Isso possibilitou uma comparação bem mais extensa em termos de quantidade de ativos, porém o Unimus realiza apenas 2 das 12 atividades executadas pelo PipeConf. Mesmo assim, optou-se por comparar as ferramentas em relação as atividades executadas em comum com o intuito de apresentar mais evidências da eficiência do PipeConf. As atividades realizadas pelo Unimus são as seguintes:

- **AT05** - Fazer *backup* da memória (*running-config*) antes da alteração; e
- **AT07** - Alterar configuração do servidor NTP (a.ntp.br).

Foi utilizada a versão 2.0.6 do Unimus e a versão 5.7.31 do MySQL[12], que é o banco de dados utilizado pelo mesmo. Os procedimentos de instalação dessas ferramentas durante a execução das análises quantitativas também estão disponíveis na documentação do PipeConf[13].

Para efeito de comparação com o Unimus, primeiramente foram perfilados os tempos médios de execução de cada atividade executada pelo PipeConf com até 32 ativos gerenciados, em relação à tarefa de configurar um servidor NTP descrita no início desta seção. A Figura

[12] MySQL: https://www.mysql.com
[13] Instalando o Unimus: https://bit.ly/3cumq9t

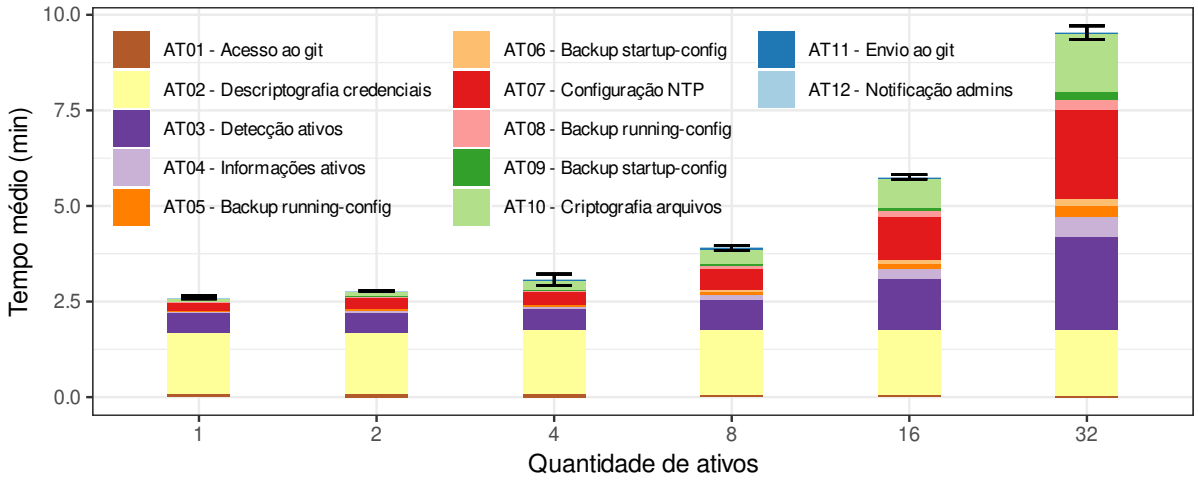


Figura 13 – Tempo médio total de configuração dos ativos e perfilamento dos tempos médios em cada atividade executada pelo PipeConf. Fonte: Elaboração própria.

13 apresenta a distribuição dos tempos médios (em minutos) para executar cada uma das 12 atividades, levando em consideração 30 repetições de cada experimento.

Um importante destaque na Figura 13 é o tempo médio total de configuração relativamente baixo (menos de 10 minutos para configurar 32 dispositivos simultâneos). Percebe-se também como cada atividade impacta no tempo médio total de configuração e como este impacto aumenta com o número de ativos. Por exemplo, verifica-se que a **AT01** influencia pouco no tempo total independente do número de ativos, já que o acesso ao repositório Git ocorre sempre muito rápido. Já a decriptografia da **AT02** demora aproximadamente 2 minutos para ser executada e apresenta um comportamento de tempo constante devido à forma como os arquivos de credenciais são identificados dinamicamente e testar alternativas constitui uma perspectiva de trabalho futuro.

Observa-se ainda que o tempo da **AT03** cresce com o aumento de ativos, pois um contêiner do Salt SProxy é iniciado e executa uma função interna do código SaltStack para detectar quais estão acessíveis. Essa verificação não é uma simples troca de mensagem, mas sim a execução do módulo *net.connected*[14], que demora alguns segundos. Ao final, o contêiner do Salt SProxy é interrompido. No *log* do PipeConf é possível constatar quais ativos estão ou não acessíveis durante as atividades, dessa forma os administradores da rede poderão tomar as devidas providências para corrigir o comportamento dos ativos afetados. Na **AT04**, outro contêiner do Salt SProxy é iniciado e continua executando até o fim da **AT09**. Neste ínterim, o PipeConf obtém algumas informações de cada ativo, tais como: fabricante, modelo, sistema operacional, arquitetura computacional, entre outras. Essas informações são necessárias ao Napalm (biblioteca usada pelo Salt SProxy), para determinar qual sintaxe de comandos será utilizada para interagir com o ativo antes de iniciar as atividades seguintes. As atividades **AT05** e **AT06** realizam o

[14] Módulo net.connected: https://bit.ly/2MOQrWA

backup das configurações antes de realizar alterações, o que possibilita reversão das ações caso aconteça algum problema. No geral demoram poucos segundos, mas tendem a aumentar o tempo com o número de ativos. A **AT07** mostrou-se impactar predominantemente na configuração de um grande número de ativos. Isto ocorre porque as configurações são paralelizadas de acordo com o número de núcleos de processamento disponíveis, conforme detalhado mais adiante. As atividades **AT08** e **AT09** comportam-se de maneira semelhante às **AT05** e **AT06**, respectivamente. Ao final da **AT09**, o contêiner do Salt SProxy é encerrado e recursos de CPU e memória são liberados. O tempo da **AT10** cresce com o aumento de ativos e testar ferramentas alternativas de criptografia é certamente uma perspectiva de trabalho futuro. Por fim, **AT11** e **AT12** são rápidas e variam pouco em relação às demais atividades.

Esta análise de perfilamento foi importante pois possibilitou detectar que o crescimento no tempo de execução da **AT07** (configurar NTP) está relacionado ao número de núcleos de processamento presentes no *notebook* utilizado[15]. Mais especificamente, o Salt SProxy inicializa a configuração de até 4 ativos simultâneos por vez, tendo em vista que o *notebook* utilizado nos experimentos possui apenas 4 núcleos. Dessa forma, um maior benefício de realizar mais tarefas em paralelo não é alcançado. Este comportamento fica ainda mais evidente em uma comparação entre o PipeConf e o Unimus, retratada na Figura 14 e na Tabela 9.

Percebe-se que o Unimus apresenta tempos de execução das suas atividades bem mais estáveis com o aumento no número de ativos (Figura 14). Já com o PipeConf (Figura 14), o aumento nos tempos de execução das atividades equivalentes é bem acentuado com mais ativos, principalmente devido à limitação das tarefas paralelas na atividade **AT07**. Com base em uma avaliação empírica, o Unimus inicializa um processo paralelo para configuração de cada ativo gerenciado e deixa o sistema operacional responsável por gerenciar tais processos. O ganho de desempenho é significativo, principalmente a partir de 16 ativos.

Para uma melhor compreensão dos dados, a Tabela 9 apresenta os tempos médios das atividades comuns ao Unimus e PipeConf. Mais importante do que analisar os valores absolutos, destaca-se as tendências nos resultados representadas em cores distintas (verde significa os melhores tempos, e vermelho os piores tempos). Por exemplo, o tempo para realizar o *backup* das configurações não é tão discrepante entre o Unimus e o PipeConf, chegando a ser melhor com o último para até oito ativos e com uma diferença máxima de aproximadamente 18 segundos para 32. Por outro lado, para configurar o servidor NTP (**AT07**), os tempos com o Unimus são sempre menores, novamente, por usufruir de um maior paralelismo. A maior diferença chega a mais de dois minutos para 32 ativos.

Objetivando aferir uma possível melhoria na execução do PipeConf mediante a análise de paralelismo da atividade **AT07**, um novo conjunto de experimentos foi realizado. Esta análise busca demonstrar o benefício de um maior nível de paralelismo e a ideia é atestar que configurar mais ativos simultaneamente repercute no tempo total obtido. Para tal, repetiu-se os mesmos

[15] Salt SProxy Issue: https://github.com/mirceaulinic/salt-sproxy/issues/200

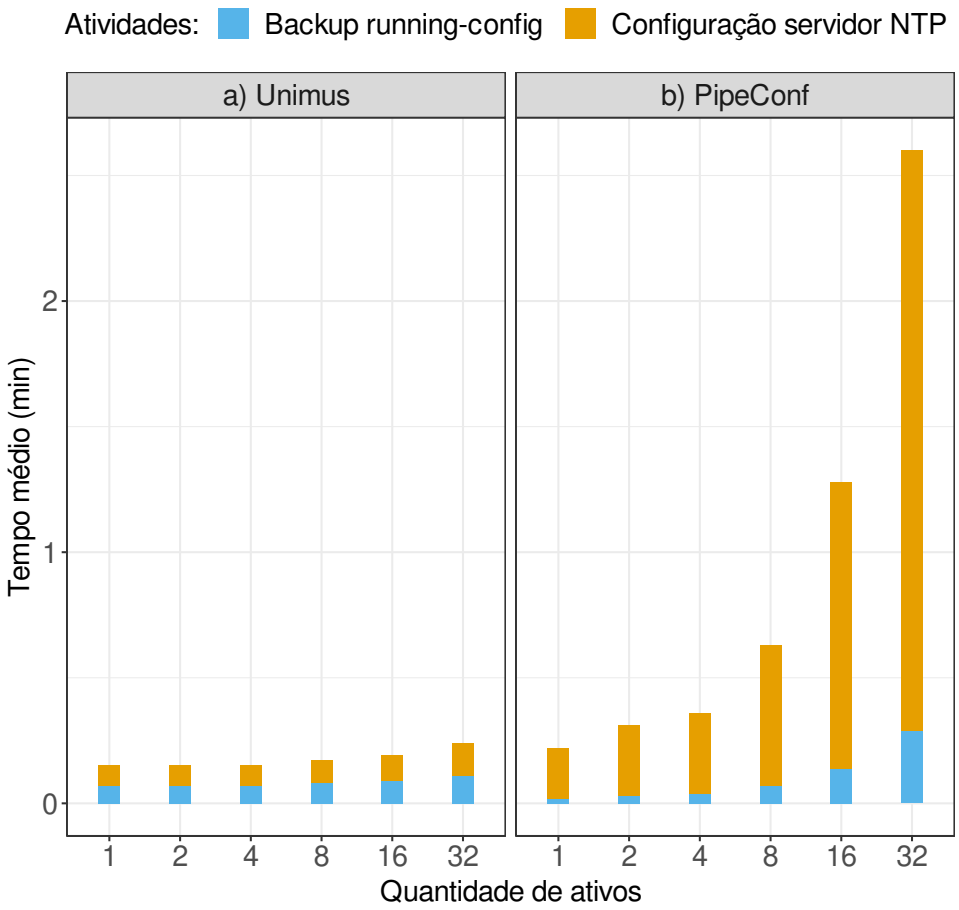


Figura 14 – Comparação dos tempos médios de configuração do Unimus e do Pipeconf, considerando as atividades comuns às duas soluções. Fonte: Elaboração própria.

experimentos até 128 ativos, porém o módulo MGC foi executado no mesmo servidor da GCP, no qual também foram simulados os ativos de rede. Para facilitar a compreensão dos resultados desses novos experimentos, foi denominada de **PipeConf++** a execução do módulo MGC no servidor da GCP apresentado na Figura 10. Como o servidor possui 24 núcleos, foi possível iniciar a gerência da configuração de até 24 ativos de forma paralela, proporcionando uma redução nos tempos de execução. Percebe-se os benefícios dessa paralelização tendo em vista que a única diferença entre as duas versões do PipeConf é o número de núcleos utilizados. Além disso, considera-se que este é um importante resultado da investigação proposta, pois possibilitará trabalhos futuros no sentido de melhorar o desempenho da solução.

A Figura 15 apresenta o tempo médio total de configuração das duas versões do PipeConf, variando o número de ativos até 128. Este gráfico demonstra claramente o benefício de paralelizar as atividades de configuração do PipeConf, ainda mais evidente com um grande número de ativos gerenciados. Observa-se na escala logarítmica que, intuitivamente, o PipeConf tende a dobrar o tempo de execução à medida que dobra o número de ativos de rede gerenciados, chegando a precisar de aproximadamente 58 minutos para gerenciar 128 ativos de rede. Por

Tabela 9 – Comparação dos tempos médios (em minutos) de *backup* e configuração do Unimus e Pipeconf, considerando os dois passos comuns às duas soluções.

Nº de Ativos	**Tempos Médios (min) dos Passos**			
	Unimus (erro máximo 0.01)		**PipeConf (erro máximo 0.07)**	
	Backing up	**Configurando**	***Backing up***	**Configurando**
1	0,07	0,08	0,02	0,2
2	0,07	0,08	0,03	0,28
4	0,07	0,08	0,04	0,32
8	0,08	0,09	0,07	0,56
16	0,09	0,1	0,14	1,14
32	0,11	0,13	0,29	2,31

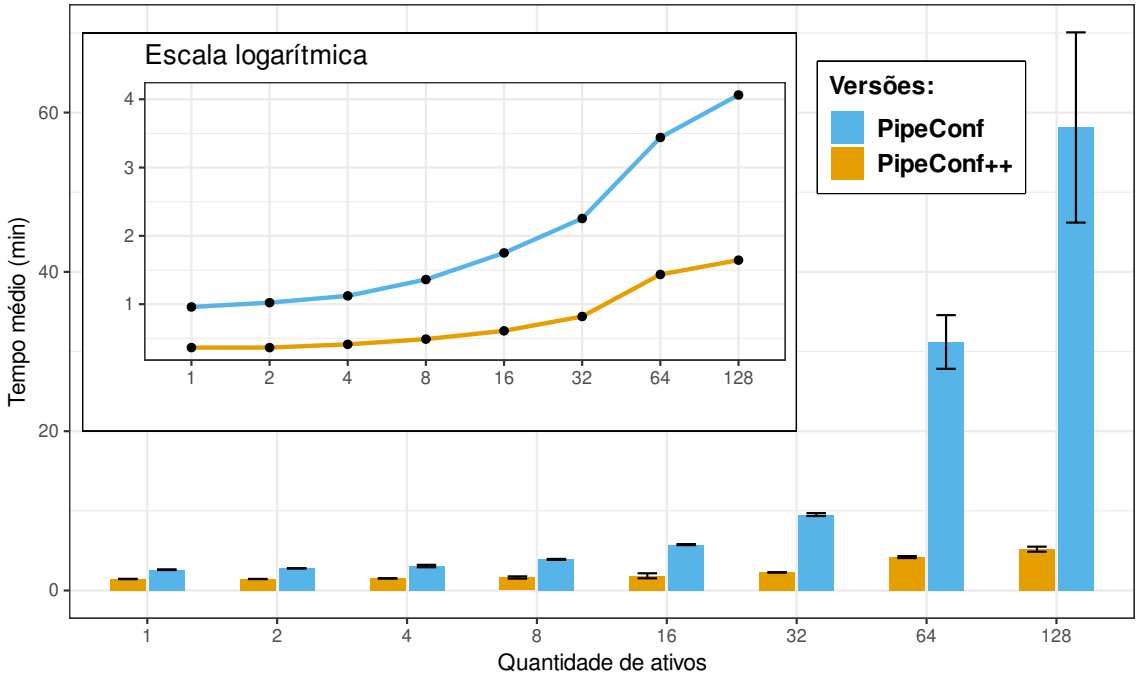


Figura 15 – Comparação dos tempos médios de configuração para as duas versões executadas do Pipeconf. Fonte: Elaboração própria.

outro lado, o PipeConf++ também apresenta uma tendência exponencial pelas retas na escala logarítmica, porém, com uma taxa de crescimento bem menor do que os tempos do PipeConf. Para gerenciar os mesmos 128 ativos, por exemplo, o PipeConf++ leva em média 6 minutos. Este comportamento é intuitivo e se deve ao fato do PipeConf conseguir executar mais tarefas em paralelo, embora limitado ao número de núcleos disponíveis no computador em que o módulo MGC é executado.

5.3.3 Considerações Sobre os Resultados Quantitativos

Em primeiro lugar, considera-se que o desempenho do PipeConf foi satisfatório para gerenciar um grande número de ativos simultâneos. A eficiência se traduz na escalabilidade do sistema, que proporcionalmente reduz o tempo de configuração à medida que aumenta o número de ativos gerenciados. Assume-se que configurar 128 ativos de rede em menos de uma hora, realizando todos os respectivos *backups* para reverter o processo caso necessário, e ainda criptografando todos os arquivos para efeito de segurança, é altamente eficiente. Tudo isso sem demandar uma alta carga no sistema onde o PipeConf é executado. Provavelmente, realizar essas tarefas de forma manual, ou mesmo através de *scripts* distintos que não executem um processo holístico de configuração, pode acarretar em inconsistências e falhas na rede, além de certamente demorar mais tempo.

Vale salientar que apesar do Unimus ter apresentado tempos de execução menores que o PipeConf, o primeiro realiza apenas um subconjunto das atividades que o segundo se propõe a executar. Ademais, o Unimus não faz uso de uma abordagem de Infraestrutura como Código ao realizar suas tarefas. Apesar de possuir uma interface web simples e intuitiva, o mesmo só executa as atividades por meio de um agendamento ou sob demanda a partir de interações com a referida interface. Pelas investigações conduzidas, o PipeConf apresenta-se como uma ferramenta bem mais completa do que as demais encontradas no mercado para gerenciar configurações de ativos de rede. Contudo, a análise possibilitou encontrar pontos passíveis de melhoria e que serão alvos de trabalhos futuros.

Como última consideração a respeito dos resultados quantitativos, ressalta-se que a heterogeneidade dos ativos também foi considerado. Embora a solução proposta nesta pesquisa ofereça suporte a ativos de rede de diferentes modelos e fabricantes, foi utilizado apenas um modelo de ativo de rede nos resultados descritos. Isto ocorreu pelos seguintes motivos:

- Os ativos simulados devem estar contidos na lista de modelos suportados pelo Napalm[16];
- Há uma limitação na variedade de modelos de ativos simulados disponíveis para serem utilizados gratuitamente com o GNS3;
- Dos demais modelos disponíveis, a memória necessária (variando entre 2 a 4GB por ativo, em comparação com 300 MB para o roteador Cisco modelo C3640-IK9O3S-M) para executar experimentos com grande quantidade de dispositivos é impeditiva para simular na nuvem, tendo em vista o alto custo de infraestrutura.

Contudo, também foram realizados teste menores envolvendo outros tipos de ativos. A partir dos resultados desses testes, percebeu-se que o comportamento dos tempos de configuração não varia significativamente dos apresentados nesta seção. Inclusive, uma das características

[16] Napalm: https://napalm.readthedocs.io/en/latest/

interessantes no uso do PipeConf é justamente a facilidade em realizar as atividades pretendidas, pois o administrador não necessita conhecer a sintaxe de diferentes modelos de ativos. Logo, acredita-se que as análises delineadas para um único modelo podem ser generalizadas, ainda que uma avaliação empírica com modelos distintos também seja alvo de trabalhos futuros.

5.4 Análise do Código da Solução

Como uma solução baseada em *software*, verificar a eficácia do PipeConf em relação à segurança do código produzido é indispensável. Considera-se que uma solução bem sucedida não deve apresentar falhas que venham a comprometer a segurança dos ativos e da rede.

Segundo Rahman, Parnin e Williams (2019), vulnerabilidades de segurança no código (do inglês *security code smell*) são padrões de codificação que podem indicar uma falha de segurança que pode ser explorada por um atacante possibilitando a execução de atividades maliciosas e/ou obtenção privilégios. Com base também nos estudos (Rahman; Parnin; Williams, 2019) e (FERNANDES, 2020), foi realizada uma análise no código do PipeConf e no código gerado por ele durante a realização dos *backups* da configuração dos ativos. O intuito foi identificar a possível presença de vulnerabilidades de segurança no código que possam comprometer o funcionamento da solução proposta nesta pesquisa.

Em (Rahman; Parnin; Williams, 2019), os autores propõem em seu estudo 7 vulnerabilidades de segurança no código para uma infraestrutura como código gerenciada com o Puppet[17], de acordo com o tipo da vulnerabilidade encontrada no código e devidamente classificada pelo *Common Weakness Enumeration*[18] (CWE), que é uma lista utilizada como base para ferramentas que medem a segurança em *hardwares* e *software*. Na prática, funciona como um padrão adotado para identificação, mitigação e prevenção de vulnerabilidades (FERNANDES, 2020). As vulnerabilidades de segurança no código propostas são apresentadas na Tabela 10.

Todos os arquivos referentes ao código do PipeConf[19] e aos *backups* da configuração dos ativos gerados pelo mesmo foram submetidos para uma análise automática através dos seguintes serviços: **Codeac**[20], **Codacy**[21], **Sonarqube**[22] e **ShellCheck**[23]. Também foi realizada uma análise manual nesses mesmos arquivos para complementar a análise automática. Em nenhuma das análises foi identificada a presença das vulnerabilidades de segurança citadas na Tabela 10.

No código dos *pipelines* executado pelo PipeConf não foi identificada a utilização de contas de usuários administrativos ou senhas explícitas. As informações de usuário e senha

[17] Puppet: https://puppet.com
[18] CWE: https://cwe.mitre.org
[19] Código do Pipeconf: https://gitlab.com/aeciopires/pipeconf
[20] Codeac: https://www.codeac.io
[21] Codacy: https://codacy.com
[22] Sonarqube: https://www.sonarqube.org
[23] ShellCheck: https://www.shellcheck.net e https://github.com/koalaman/shellcheck

Tabela 10 – Vulnerabilidades de segurança no código propostas por Rahman, Parnin e Williams (2019).

Vulnerabilidade	**Código CWE**	**Descrição**
1) Admin por padrão	CWE-250	Execução do código com privilégios desnecessários.
2) Senha vazia	CWE-258	Senha vazia no arquivo de configuração.
3) Credenciais definidas em texto plano	CWE-789, CWE259	Uso de credenciais em texto plano.
4) Vinculação com endereço IP desnecessário	CWE-284	Controle de acesso da conexão inadequado.
5) Comentários que possibilitem a dedução sobre o acesso a algum serviço	CWE-546	Comentários que permitam deduzir como obter o acesso a determinado serviço ou funcionalidade.
6) Ausência de uso de protocolo HTTPS com TLS	CWE-319	Transmissão de informações sensíveis de forma não criptografada.
7) Uso de algoritmos de criptografia fraca	CWE-327, CWE-326	Uso de algoritmo criptográfico inseguro ou inadequado.

são cadastradas na interface web do Jenkins e armazenadas de forma criptografada. Por esse motivo, as vulnerabilidades 1 e 2 citados na Tabela 10 não foram encontradas no código-fonte. Ainda que, durante a configuração do ambiente, seja possível ao administrador cadastrar no Jenkins uma conta administrativa e senha vazia para integrar algum serviço ao PipeConf, isso não configura um requisito para o seu funcionamento e também não é recomendado para ambientes de produção.

Todos os arquivos que contêm as credenciais de acesso aos ativos de rede e os *backups* de configuração gerados pelo PipeConf são criptografados com o uso do Sops[24], em conjunto com o uso do serviço AWS-KMS[25]. Para isso, utiliza-se o algoritmo de criptografia simétrica AES no modo GCM com 256 bits (AES-256-GCM). Tais arquivos são transportados, armazenados e versionados no Git de forma criptografada. Dessa forma, não é possível visualizar as informações sensíveis à cerca dos ativos de rede e isso impede a ocorrência das vulnerabilidade 3 e 7.

Após uma análise minuciosa no código do PipeConf não foi identificada a utilização de endereços IPs desnecessários, tais como 0.0.0.0/0. A Subseção 4.4.3 contém as informações apropriadas para o administrador da rede configurar regras no *firewall* permitindo acesso apenas aos endereços IP de origem estritamente necessários. Dessa forma, a vulnerabilidade 4 também está ausente no código. Ainda com base nesta análise, não foi encontrada a vulnerabilidade 5, pois nenhum comentário está disponível que permita deduzir sobre como acessar algum serviço integrado ao PipeConf.

[24] Sops: https://github.com/mozilla/sops
[25] AWS-KMS: https://aws.amazon.com/kms

O Jenkins, o Git e os serviços de envio de notificação aos usuários (*email* e Slack) podem ser configurados pelo administrador de rede para utilizar o protocolo HTTP ou HTTPS. Este último é o padrão recomendado para ambientes de produção. Não há limitações no código do PipeConf que interfira no uso do protocolo HTTPS com TLS (*Transport Layer Security*). Vale salientar que para fazer uso desse protocolo, é necessário utilizar um certificado X.509[26] autoassinado ou assinado por alguma autoridade certificadora (CA, do inglês *certification authority*). Por exemplo, uma delas é o ***Let's Encrypt***[27], que oferece os serviços de uma autoridade certificadora gratuitamente. Dessa forma, conclui-se que a vulnerabilidade 6 também está ausente no código.

Neste capítulo foi apresentada a avaliação quantitativa da solução proposta e desenvolvida nesta pesquisa. As análises foram constituídas por um ambiente controlado com ativos de rede simulados ao realizar o gerenciamento da configuração utilizando a abordagem Infraestrutura como Código. Foram apresentados as análise de desempenho e perfilamento, a metodologia utilizada em cada análise, a definição das métricas analisadas e a descrição do *hardware* e *software* utilizados, bem como os resultados obtidos. Algumas oportunidades de melhoria no PipeConf foram identificadas e podem ser investigadas em trabalhos futuros. O capítulo seguinte apresenta as considerações finais da pesquisa.

[26] Certificado X.509: https://www.ssl.com/faqs/what-is-an-x-509-certificate/
[27] Let's Encrypt: https://letsencrypt.org

6 CONSIDERAÇÕES FINAIS

Este capítulo final conclui o trabalho realizado ao consolidar os resultados obtidos e extrair as conclusões relevantes. Além disso, as principais contribuições alcançadas durante a elaboração deste trabalho são apresentadas, bem como perspectivas de trabalhos futuros.

Esta pesquisa possibilitou a aplicação da abordagem de Infraestrutura como Código para gerenciar a configuração de ativos de rede heterogêneos, através de uma solução integrada de *software*. Além da implementação da solução proposta, foi elaborada uma prova de conceito por meio de um ambiente simulado, a fim de realizar uma avaliação quantitativa da eficiência em termos de tempo de execução e desempenho do sistema. Para isso, foram realizados experimentos variando o número de ativos de rede gerenciados e coletadas diversas métricas a respeito da execução do PipeConf. Ainda com o intuito de validar a ferramenta, uma análise da eficácia do PipeConf em relação à presença de vulnerabilidades também foi apresentada.

A partir dos resultados obtidos, considera-se que o PipeConf apresentou um desempenho satisfatório mesmo com um grande número de ativos gerenciados, embora o desempenho do PipeConf depende diretamente do número de núcleos de CPU disponíveis. Usar a abordagem de IaC para configurar de forma holística ativos simultâneos garantiu um processo simples, rápido, seguro e reversível. Indiscutivelmente esse processo é bem menos eficiente quando realizado de forma manual, pois está suscetível a problemas incorridos pelas pessoas que administram tais recursos. Acredita-se ainda que utilizar diversas ferramentas separadamente não proporciona o mesmo benefício, além de necessariamente exigir experiência no uso de diferentes soluções. Embora tenham sido identificadas algumas oportunidades de melhoria no sistema, tais melhorias são exequíveis e podem ser investigadas em trabalhos futuros. Portanto, conclui-se que os objetivos iniciais desta pesquisa foram atendidos por meio do uso do PipeConf que, dentro do contexto investigado, se mostrou eficiente e eficaz em gerenciar a configuração de ativos de rede simultâneos.

6.1 Contribuições da Pesquisa

A realização desta pesquisa proporcionou as seguintes contribuições:

- Revisão sistemática da literatura na área de IaC para identificar e analisar os principais trabalhos relacionados a esta pesquisa;
- Definição de um arquitetura de *software* modular para gerenciamento automatizado de configurações de ativos de rede, que leva em consideração diferentes modelos e fabricantes, além de realizar versionamento e *backup* de arquivos;

- A implementação da solução integrada como prova de conceito da arquitetura proposta;
- Validação da solução integrada[1] em ambiente simulado, com o intuito de analisar os resultados obtidos segundo critérios de eficiência e eficácia;
- Compartilhamento do conjunto de dados (do inglês *dataset*)[2] dos experimentos realizados e das configurações necessárias para replicação dos mesmos, envolvendo o PipeConf e o Unimus para fins comparativos;

Como resultado da implementação da solução proposta foi realizado o depósito de patente no Instituto Nacional da Propriedade Industrial (INPI[3]) sob o número **BR202019017745-0**, com base nos fluxos de trabalhos. Também foi registrado o código fonte da solução proposta no INPI sob o número **BR512019001905-9**.

6.2 Trabalhos Futuros

Conforme delineadas ao longo do Capítulo 5, outras pesquisas podem ser realizadas com objetivo de melhorar a execução do PipeConf. Por exemplo, adotar outro componente de *software* que paralelize o gerenciamento dos ativos de forma independente do número de núcleos físicos de processamento parece promissor. Ou ainda, investigar uma abordagem alternativa para decriptografar de maneira mais rápida os arquivos necessários. Além disso, uma análise mais extensiva da solução envolvendo experimentos empíricos com ativos de modelos e fabricantes distintos pode ser realizada. A intenção é confirmar a hipótese de que o PipeConf mitiga a dificuldade de gerenciar ativos heterogêneos, não só em relação ao tempo de configurar sintaxes diferentes, mas também em relação à necessidade de conhecer previamente tais sintaxes. Por fim, uma análise qualitativa da solução para avaliar a usabilidade da solução com experimentos envolvendo usuários reais também é pretendida.

[1] Instruções de instalação da Prova de Conceito: https://gitlab.com/aeciopires/pipeconf/-/tree/master/docs.
[2] Dataset do PipeConf: https://tinyurl.com/y5kny7yb
[3] INPI: https://www.gov.br/inpi/pt-br

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, L.; MACIEL JR., P. D.; VERDI, F. L. Cloud network slicing: A systematic mapping study from scientific publications. Research Square, Jun 2020. Disponível em: <http://dx.doi.org/10.21203/rs.3.rs-34336/v1>. Citado 2 vezes nas páginas 29 e 32.

ARTA, M. et al. DevOps: Introducing Infrastructure-as-Code. In: *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press, 2017. (ICSE-C '17), p. 497–498. ISBN 9781538615898. Disponível em: <https://doi.org/10.1109/ICSE-C.2017.162>. Citado na página 15.

CHEN, H.; WANG, A.; LOO, B. T. Towards Example-Guided Network Synthesis. In: *Proceedings of the 2nd Asia-Pacific Workshop on Networking*. New York, NY, USA: Association for Computing Machinery, 2018. (APNet '18), p. 65–71. ISBN 9781450363952. Disponível em: <https://doi.org/10.1145/3232565.3234462>. Citado na página 36.

CHRISTOPHER, R. Strategic it management: how companies can benefit from an increasing it influence. *Journal of Enterprise Information Management*, Emerald Publishing Limited, v. 32, n. 2, p. 251–273, Jan 2019. ISSN 1741-0398. Disponível em: <https://doi.org/10.1108/JEIM-08-2018-0172>. Citado na página 15.

DAVIS, J.; DANIELS, R. *What Is DevOps?* 1st. ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2018. ISBN 9781492039877. Citado na página 22.

DUPLYAKIN, D.; HANEY, M.; TUFO, H. Architecting a Persistent and Reliable Configuration Management System. In: *Proceedings of the 6th Workshop on Scientific Cloud Computing - ScienceCloud '15*. New York, New York, USA: ACM Press, 2015. (ScienceCloud '15), p. 11–16. ISBN 9781450335706. Disponível em: <https://doi.org/10.1145/2755644.2755647>. Citado na página 35.

DWARAKI, A. et al. GitFlow. In: *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research - SOSR '15*. New York, New York, USA: ACM Press, 2015. (SOSR '15), p. 1–6. ISBN 9781450334518. Disponível em: <https://doi.org/10.1145/2774993.2775064>. Citado na página 36.

EMILSSON, R. Monography: Container Performance Benchmark Between Docker, LXD, Podman & Buildah. In: . University of Skovde, 2020. Disponível em: <http://www.diva-portal.org/smash/get/diva2:1450777/FULLTEXT01.pdf>. Citado na página 45.

FERNANDES, L. D. G. A. e. C. N. S. D. D. Security smells em infraestrutura como codigo utilizando docker. 2020. Disponível em: <http://sbseg.sbc.org.br/2020/pdfs/artigos_curtos_mencao_honrosa_2.pdf>. Citado na página 69.

GARROS, D.; CAEN, F. NetDevOps Survey. In: . [s.n.], 2019. Disponível em: <https://dgarros.github.io/netdevops-survey/reports/2019.html>. Citado na página 16.

HUMBLE, J.; FARLEY, D. G. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010. Disponível em: <http://my.safaribooksonline.com/9780321601919>. Citado na página 23.

INDIA, S. What is devops pipeline? simplified in 200 words. In: . [s.n.], 2019. Disponível em: <https://medium.com/tech-in-200-words/what-is-devops-pipeline-simplified-in-200-words-cb524f94842c>. Citado 2 vezes nas páginas 17 e 23.

ISMAIL, H.; HAMZA, H. S.; MOHANED, S. M. Semantic Enhancement for Network Configuration Management. In: *2018 IEEE Global Conference on Internet of Things (GCIoT)*. [s.n.], 2018. p. 1–5. Disponível em: <https://doi.org/10.1109/GCIoT.2018.8620148>. Citado na página 17.

JIANG, Y.; ADAMS, B. Co-Evolution of Infrastructure and Source Code: An Empirical Study. In: *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE Press, 2015. (MSR '15), p. 45–55. ISBN 9780769555942. Disponível em: <https://doi.org/10.1109/MSR.2015.12>. Citado 2 vezes nas páginas 17 e 35.

KREEFTMEIJER, J. Understanding system load and load averages. In: . [s.n.], 2018. Disponível em: <https://blog.appsignal.com/2018/03/28/understanding-system-load-and-load-averages.html>. Citado na página 62.

LEITE, L. et al. A Survey of DevOps Concepts and Challenges. *ACM Computing Surveys*, Association for Computing Machinery, New York, NY, USA, v. 52, n. 6, p. 1–35, jan 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3359981>. Citado na página 37.

LIU, H. H. et al. Automatic life cycle management of network configurations. In: *Proceedings of the Afternoon Workshop on Self-Driving Networks*. New York, NY, USA: Association for Computing Machinery, 2018. (SelfDN 2018), p. 29–35. ISBN 9781450359146. Disponível em: <https://doi.org/10.1145/3229584.3229585>. Citado na página 16.

MCKINSEY. *McKinsey & Company*. 2016. Acessado: 04 agosto 2020. Disponível em: <http://www.mckinsey.com>. Acesso em: 04 agosto 2020. Citado na página 16.

METODOLOGIACIENTíFICA.ORG. Metodologia Experimental. In: . [s.n.], 2019. Disponível em: <https://www.metodologiacientifica.org/metodos-de-procedimentos/metodo-experimental/>. Citado na página 55.

MORALES, J. A.; YASAR, H.; VOLKMAN, A. Implementing DevOps practices in highly regulated environments. In: . New York, New York, USA: ACM Press, 2018. p. 1–9. ISBN 9781450364225. Disponível em: <https://doi.org/10.1145/3234152.3234188>. Citado na página 37.

MORRIS, K. *Infrastructure as Code - Managing Servers in the Cloud*. 1st. ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2016. ISBN 978-1491924358. Citado 2 vezes nas páginas 17 e 21.

MOURA, A.; SAUVÉ, J. P.; BARTOLINI, C. Business-driven IT management - upping the ante of IT : exploring the linkage between it and business to improve both IT and business results. *IEEE Communications Magazine*, v. 46, n. 10, p. 148–153, 2008. Disponível em: <https://doi.org/10.1109/MCOM.2008.4644133>. Citado na página 15.

OPARA-MARTINS, J.; SAHANDI, R.; TIAN, F. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, Springer Verlag, v. 5, n. 1, p. 4, dec 2016. ISSN 2192-113X. Disponível em: <http://journalofcloudcomputing.springeropen.com/articles/10.1186/s13677-016-0054-z>. Citado na página 38.

OTTE, S. Version control systems. *Computer Systems and Telematics*, p. 11–13, 2009. Citado na página 27.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*. Swindon, GBR: BCS Learning & Development Ltd., 2008. (EASE'08), p. 68–77. Citado na página 29.

PIRES, A.; MILITÃO, J. Integração Contínua com Jenkins. In: . Novatec, 2019. v. 1. ISBN 978-85-7522-722-0. Disponível em: <https://novatec.com.br/livros/jenkins>. Citado na página 45.

PRESTON, H. What does 'Network as code' mean? In: . [s.n.], 2018. Disponível em: <https://blogs.cisco.com/developer/what-does-network-as-code-mean>. Citado na página 23.

RAHMAN, A.; MAHDAVI-HEZAVEH, R.; WILLIAMS, L. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, v. 108, p. 65 – 77, 2019. ISSN 0950-5849. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0950584918302507>. Citado na página 20.

Rahman, A.; Parnin, C.; Williams, L. The seven sins: Security smells in infrastructure as code scripts. p. 164–175, 2019. Citado 5 vezes nas páginas 9, 55, 56, 69 e 70.

RUPARELIA, N. B. The history of version control. *ACM SIGSOFT Software Engineering Notes*, ACM New York, NY, USA, v. 35, n. 1, p. 5–9, 2010. Citado na página 27.

SHAH, J. A.; DUBARIA, D. NetDevOps: A New Era Towards Networking & DevOps. In: *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*. IEEE, 2019. p. 0775–0779. ISBN 978-1-7281-3885-5. Disponível em: <https://ieeexplore.ieee.org/document/8992969/>. Citado 3 vezes nas páginas 16, 22 e 37.

TIAN, B. et al. Safely and automatically updating in-network ACL configurations with intent language. In: *Proceedings of the ACM Special Interest Group on Data Communication*. New York, NY, USA: ACM, 2019. (SIGCOMM '19), p. 214–226. ISBN 9781450359566. Disponível em: <https://doi.org/10.1145/3341302.3342088>. Citado na página 36.

TORBERNTSSON, K.; RYDIN, Y. *A Study of Configuration Management Systems : Solutions for Deployment and Configurationof Software in a Cloud Environment*. 2014. (TVE, 14 013). Disponível em: <https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A732615&dswid=-7098>. Citado na página 24.

ULINIC, M. Automating Networks Using Salt, Without Running Proxy Minions. In: . [s.n.], 2019. Disponível em: <https://ripe79.ripe.net/presentations/30-RIPE-79_-Automating-Networks-Using-Salt-Without-Running-Proxy-Minions.pdf>. Citado 2 vezes nas páginas 8 e 44.

VILALTA, R. et al. Network Programmability and Automation in Optical Networks. In: . Springer, 2020. p. 223–234. ISBN 9783030380847. ISSN 16113349. Disponível em: <http://link.springer.com/10.1007/978-3-030-38085-4>. Citado na página 35.

WIERINGA, R. et al. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirements Engineering*, v. 11, n. 1, p. 102–107, Mar 2006. ISSN 1432-010X. Disponível em: <https://doi.org/10.1007/s00766-005-0021-6>. Citado na página 32.

WILSON, K. Resources for the Central Orchestration talk at AusNOG 2017. In: . [s.n.], 2017. Disponível em: <https://github.com/kenwilson/central-orchestration-ausnog2017>. Citado na página 42.

ZOLKIPLI, N. N.; NGAH, A.; DERAMAN, A. Version control system: A review. *Procedia Computer Science*, v. 135, p. 408–415, 01 2018. Citado na página 27.

APÊNDICE A – FERRAMENTAS DE CONFIGURAÇÃO DE ATIVOS DE REDE HETEROGÊNEOS

Este apêndice apresenta não apenas uma lista de ferramentas que foram investigadas com o intuito de analisar comparativamente com o desempenho do PipeConf, mas também um breve registro de como essa investigação foi conduzida. Os dois grandes desafios para validar a ferramenta consistiram em realizar experimentos empíricos com ativos e usuários reais, além de encontrar uma solução que realize o processo de configuração de ativos de maneira completa como o PipeConf, seguindo os preceitos de IaC.

Em um primeiro momento, buscou-se identificar a existência de um *dataset* público que contenha os registros de tempo necessários para executar o *backup* e alteração de configuração em *switches* e/ou roteadores. O objetivo seria gerar uma carga de trabalho sintética que correspondesse ao funcionamento de uma LAN, MAN ou WAN, dada a impossibilidade de uma experimentação empírica. Nesta busca, o mesmo processo de revisão sistemática da literatura apresentado no Capítulo 3 foi empregado, a partir da *string* de busca listada a seguir. Contudo, não foi localizado nenhum *dataset* com registros de tempo úteis para ser utilizado na avaliação proposta.

```
( dataset
  AND
  time
  AND
  (switch OR router)
  AND
  (config OR configuration)
  AND
  backup
)
```

O segundo desafio foi identificar uma ferramenta que possa ser utilizada em uma análise comparativa e que exponha os registros de todas as atividades de configuração e *backup* realizadas. Realizou-se uma busca extensiva na Internet através de motores de busca e sites empresariais específicos, bem como grupos de discussão especializados no Facebook[1]. Entretanto, não houveram respostas relevantes e as ferramentas encontradas são detalhadas a seguir.

[1] P.ex., a pergunta no grupo de discussão "Emuladores & Simuladores de Redes": https://bit.ly/37cItyB; cuja temática é diretamente relacionada e conta com a participação de mais 5.000 profissionais, a maioria brasileiros.

Ao todo foram investigadas as 10 ferramentas citadas na Tabela 11. Como mencionado anteriormente, apenas o **Unimus** se mostrou parcialmente viável, tendo em vista que realiza bem menos tarefas que o PipeConf e que possui uma licença gratuita para configuração de até 5 ativos. Entretanto, mediante contato com a equipe de desenvolvimento da ferramenta, foi possível obter uma licença de quatro semanas para configurar até 32 ativos simultâneos. Isso possibilitou uma avaliação de escala mais próxima do desejado. Optou-se então por comparar os dois passos de configuração comuns ao Unimus e ao PipeConf, e o registro dos tempos está devidamente citado na Seção 5.3.

Tabela 11 – Ferramentas de Configuração de Ativos de Rede Heterogêneos.

Ferramentas	**Métricas de tempo**	**É pago?**	**Período de teste**	**Observações**
Unimus[2]	✓	✓	✓	- Gratuito até 5 ativos e licença adicional de cada ativo custa anualmente US$ 4.5 + IOF; - Ativos suportados[3] e funcionalidades[4].
pyATS[5]	-	-	✓	- Gratuito, porém de complexa usabilidade; - Nos testes realizados não foi possível gerenciar nenhum ativo utilizando-o[6].
NetBox[7]	-	-	✓	- Usa o Napalm apenas para receber o *backup* da configuração e não exibe métricas de tempo relacionada a esta atividade; - Essa funionalidade é essencial para as análises pretendidas[8].
Frinx[9]	-	✓	✓	-
Itential Automation Platform[10]	-	✓	✓	-
NetRMI[11]	-	✓	✓	-
Apstra[12]	-	✓	✓	-
Rconfig[13]	-	✓	✓	-
Network Automation Manager - NAM[14]	-	✓	✓	-
NetWork Configuration Manager - NCM[15]	-	✓	✓	-

[2] Unimus: https://unimus.net
[3] Ativos suportados pelo Unimus: https://bit.ly/3n9hGbQ
[4] Funcionalidades do Unimus: https://unimus.net/features.html
[5] pyATS: https://developer.cisco.com/pyats/
[6] Instalação do pyATS: https://bit.ly/34oo47Z
[7] NetBox: https://github.com/netbox-community/netbox
[8] NetBox utilizando Napalm: https://netbox.readthedocs.io/en/stable/additional-features/napalm/
[9] Frinx: https://frinx.io
[10] Itential: https://www.itential.com
[11] NetRMI: https://www.infoblox.com/products/netmri/
[12] Apstra: https://apstra.com
[13] Rconfig: https://www.rconfig.com
[14] NAM: https://www.solarwinds.com/network-automation-manager
[15] NCM: https://bit.ly/2JS1Wf8