DOO: Solução Integrada de Gerenciamento de Serviços

Ramon dos R. Fontes[1], **Thiago de A. Lima**[1]

[1]Programa de Pós-Graduação em Tecnologia da Informação (PPGTI)
Universidade Federal do Rio Grande do Norte (UFRN)
Caixa Postal 1524 – 59078-900 – Natal – RN – Brasil

ramon.fontes@ufrn.br, thiago.abreu@ufrn.br

Abstract. *The management of services in a computer network requires manual effort to maintain the infrastructure. Typically, network administrators need to access multiple servers, install various software, and configure numerous files manually. This manual approach often results in errors, impacting the availability of services. The infrastructure as code approach emerges as a solution to minimize errors associated with traditional service management. In this context, this work introduces* `DOO`*, acronym for DevOpsOrchestrator, an integrated platform that utilizes infrastructure as code to automate service management.*

Resumo. *O gerenciamento de serviços em uma rede de computadores demanda um esforço considerável para manter a infraestrutura. Tipicamente, os administradores de rede são obrigados a acessar múltiplos servidores, instalar diversos softwares e configurar numerosos arquivos manualmente. Essa abordagem manual frequentemente resulta em erros, afetando a disponibilidade dos serviços. A adoção da infraestrutura como código surge como uma solução para minimizar os equívocos associados ao gerenciamento tradicional de serviços. Neste contexto, apresentamos o* `DOO`*, ou DevOpsOrchestrator, uma plataforma integrada que utiliza a infraestrutura como código para automatizar o gerenciamento de serviços.*

1. Introdução

O termo serviço é definido como uma funcionalidade oferecida aos usuários de uma rede com qualidade específica [Rodosek 2002]. Neste contexto, os provedores de serviços empregam uma combinação de pessoas, processos e tecnologia da informação para fornecer funcionalidades aos seus clientes. A gestão operacional dos serviços é tipicamente de responsabilidade da equipe de Tecnologia da Informação (TI), composta por diversos profissionais especializados que desenvolvem, disponibilizam e mantêm os serviços.

Na abordagem tradicional, o administrador do sistema acessa os servidores diretamente por meio da *Command-Line Interfaces* (CLIs) e realiza mudanças manualmente ou através da criação de scripts responsáveis pela configuração de serviços. De acordo com estudos realizados por [Kundnani 2019], cerca de 95% das alterações de rede são feitas manualmente e, aproximadamente, 70% das violações de políticas são causadas por erros humanos. Além disso, cerca de 75% das despesas operacionais são destinadas a alterações e soluções de problemas de rede.

Com o objetivo de minimizar os erros decorrentes de alterações manuais, o uso de técnicas de automação permite que serviços sejam gerenciados com maior agilidade.

Um dos maiores benefícios da implementação de automação é que cada mudança de configuração está localizada em um arquivo [Shah and Dubaria 2019], o que proporciona maior controle das modificações e facilita a resolução de problemas. A Infraestrutura como código (do inglês, *Infrastructure as Code* (IaC)) é um termo usado para descrever o processo de automação de infraestrutura de servidor e rede através do uso de software [Morris 2016]. Nessa abordagem, é escrito um código que ao ser executado, se define, atualiza e implanta a infraestrutura. [Brikman 2017] considera tanto a criação de recursos (máquinas virtuais, balanceadores de carga, bancos de dados, por exemplo.), bem como o gerenciamento das configurações dentro desses recursos como parte das ferramentas IaC.

A adoção da Infraestrutura como Código elimina a exigência da equipe de TI gerenciar manualmente tarefas repetitivas e rotineiras. Isso simplifica o provisionamento dinâmico de uma variedade de serviços e recursos, permitindo a execução do código criado sem a necessidade de acessar os servidores diretamente. [Masek et al. 2018] afirma que a automatização do gerenciamento da rede e seus serviços através de IaC é uma forma de representar a gestão de infraestrutura de rede, fornecendo uma arquitetura que depende do versionamento dos códigos em um repositório de código-fonte que os engenheiros de DevOps (do inglês, *Development Operations*) irão consumir. Além da automação, um ambiente definido em formato de código pode ser implantado em qualquer infraestrutura, diminuindo os erros e inconsistências [Jiang and Adams 2015].

Diante da importância de ferramentas IaC, este trabalho propõe uma solução de software denominada `DOO`, acrônimo para DevOpsOrchestrator, que consiste em uma plataforma integrada que utiliza a infraestrutura como código para automatizar o gerenciamento de serviços. Essa plataforma permite, dentre outras coisas, que administradores de rede utilizem uma interface gráfica para acessar servidores e realizar configurações que antes eram feitas manualmente. A implantação do `DOO` traz consigo principalmente a capacidade de reduzir erros decorrentes de configurações incorretas, agilizar a entrega de mudanças e promover uma maior consistência nos serviços prestados.

Além desta seção introdutória, este artigo segue a seguinte estrutura: a Seção 2 apresenta os trabalhos relacionados; o DevOpsOrchestrator (`DOO`) é apresentado na Seção 3, explicando sua arquitetura e camadas; a Seção 4 descreve o ambiente de validação e os casos de uso de automação de serviços utilizando o `DOO`; e, finalmente, a Seção 5 apresenta as conclusões.

2. Trabalhos relacionados

Nos trabalhos que serão apresentados a seguir, várias abordagens foram realizadas com o objetivo de automatizar serviços de TI. Em todos os trabalhos identificados, uma ferramenta de IaC foi empregada, resultando em melhorias na gestão desses serviços em comparação com a abordagem tradicional.

Em [Bertrand et al. 2020] é apresentado um grupo de infraestrutura dentro de uma unidade denominado de *Integrated Control System Division* (ICS) que é responsável pelos sistemas de controle das instalações do *European Spallation Source* (ESS). O ESS tem como função projetar, implementar e operar a infraestrutura de TI necessária para executar de forma confiável o ecossistema do Experimental *Physics Industrial Control System* (EPICS)[1]. A equipe de infraestrutura do ESS gerencia um grande número de re-

[1]EPICS é um conjunto de ferramentas e aplicativos de software que fornecem uma infraestrutura de soft-

des, dispositivos físicos e máquinas virtuais, fazendo uso da IaC para automatizar e tornar a implantação repetível, reproduzível e confiável. A equipe contou com um servidor interno GitLab para armazenamento do código e JFrog Artifactory[2] para hospedar artefatos binários, fazendo uso extensivo dos pipelines de CI/CD do GitLab para integração contínua. Os fluxos de trabalho foram construídos em Ansible, AWX (uma *GUI* para Ansible) e CSEntry (uma aplicação web personalizada desenvolvida internamente). No artigo, os autores mostraram como funciona alguns desses fluxos, como é o caso de uma nova máquina virtual cadastrada no CSEntry que geram dois gatilhos: (i) sincronização do inventário e (ii) atualização dos serviços principais. Essas tarefas são processadas em segundo plano por *RQ Workers*[3], os comandos são enviados para o AWX usando a *Application Programming Interface* (API) REST e a biblioteca ansible-tower-cli[4], que realiza uma configuração completa para cada serviço (DHCP, DNS e Radius) a partir de playbooks Ansible usando o inventário.

Uma abordagem de IaC que considera diferentes modelos e fabricantes de ativos de rede foi proposto em [Pires et al. 2021]. O trabalho propõe a integração de ferramentas existentes de IaC através da arquitetura denominada PipeConf, possibilitando a centralização de automação das atividade de backup, configuração dos ativos, além de permitir mudanças sem recompilar ou reiniciar os aparelhos. A arquitetura proposta pelos autores abstrai a sintaxe dos diferentes modelos de dispositivos, facilitando a interoperabilidade e o gerenciamento de grande quantidade de dispositivos. Foi realizada uma prova de conceito da arquitetura para avaliação quantitativa da eficiência no gerenciamento da configuração de até 128 ativos de rede. Neste trabalho, o PipeConf foi formado por diversos módulos, sendo eles: módulo de versionamento; módulo de criptografia; o módulo de gerência de configuração; módulo de notificação; e o módulo de integração contínua.

Em [Hariyadi and Marzuki 2020] é descrito um processo de automação da criação de *Virtual Private Server* (VPS) para dar suporte à prática em cursos de gerenciamento de rede na Universidade de Bumigora, na Indonésia. Tradicionalmente, a cada semestre, os administradores de redes precisavam criar as VPS para cada aluno que participaria das aulas práticas de gerenciamento de rede. Esse processo era realizado manualmente por meio da GUI da Web do software *Proxmox Virtual Environment* (PVE), e levava aproximadamente 2 minutos e 15 segundos por usuário. No entanto, a configuração manual das VPS e sistemas tornou-se ineficaz e ineficiente com o aumento do número de usuários de VPS e a repetição da atividade a cada semestre. Diante desse cenário, os autores desenvolveram uma solução automatizada para a criação de VPS utilizando a ferramenta Ansible através de um Playbook e um arquivo no formato YAML, que possuía tarefas relacionadas à automação do gerenciamento das VPSes.

O framework NetInfra, apresentado por [McGhee et al. 2022], unifica o gerenciamento dos serviços DHCP e DNS em um único código, reduzindo a repetição de trabalho e simplificando a configuração desses serviços. Ele aborda a integração desses serviços, comumente usados em centros de pesquisa, que, embora inter-relacionados, são

---

ware para uso na construção de sistemas de controle distribuídos para operar dispositivos como aceleradores de partículas.

[2] `https://jfrog.com/artifactory/`

[3] `https://python-rq.org/docs/workers/`

[4] `https://docs.ansible.com/ansible-tower/latest/html/towercli/index.html`

tradicionalmente fornecidos por softwares separados, exigindo configurações distintas e gerenciamento independente.

Autores	Repositório	Pipeline	Ferramenta	Interface gráfica	Código gerado	Serviços
[Bertrand et al. 2020]	Gitlab	Sim	Ansible	Sim	Não	Criação de VM, configuração de DNS e DHCP
[Pires et al. 2021]	Git	Sim	Saltstack	Sim	Não	Configuração de ativos de rede
[Hariyadi and Marzuki 2020]	-	Não	Ansible	Não	Não	Gerenciamento de VPS
[McGhee et al. 2022]	-	Não	Ansible	Não	Não	Configuração de DNS e DHCP
DOO	Gitlab	Sim	Ansible	Sim	Sim	Instalação e configuração de serviços no geral

Tabela 1. Comparação entre os trabalhos relacionados

A Tabela 1 apresenta as principais características de cada um dos trabalhos relacionados. Como pode ser observado, todos os trabalhos utilizam alguma ferramenta de IaC. A respeito da geração de código, somente o DOO gera automaticamente o código mediante preenchimento de formulário pelo usuário. Além disto, o DOO tem o diferencial de permitir uma abordagem generalista, automatizando diferentes serviços de rede, ao contrário dos trabalhos relacionados que focam em algum serviço específico. Além disso, o código é gerado pela interface gráfica, o que facilita a automatização e diminui a necessidade de especialista em programação. Em geral, os trabalhos relacionados focam em alguns serviços definidos, de tal forma que, para adicionar novos serviços se fazem necessárias mudanças em seus sistemas.

3. DevOpsOrchestrator (DOO)

A fim de automatizar serviços de TI, o DevOpsOrchestrator (DOO) foi concebido levando em consideração os conceitos de infraestrutura como código. Esta solução permite aos profissionais realizar modificações em servidores e arquivos de configuração através de uma interface gráfica. Essa interface é capaz de gerar o código da infraestrutura correspondente, salvá-lo em um sistema de versionamento e provisionar as mudanças necessárias. Como a gestão de serviços engloba múltiplos setores e faz uso de diversos softwares, os quais, frequentemente, não estão integrados entre si, é proposta uma arquitetura destinada a centralizar essas informações, visando agilizar o desenvolvimento do código responsável pela implementação das mudanças no serviço. A seguir são apresentados detalhes sobre sua implementação, a arquitetura e também algumas limitações identificadas durante o desenvolvimento do DOO.

3.1. Implementação

O DOO foi desenvolvido na linguagem de programação Python, com o framework Django e o banco de dados PostgreSQL. O mesmo segue o padrão arquitetural Model-Template-View (MTV). Para o desenvolvimento da aplicação foram consideradas as dependências listadas na Tabela 2.

Dependências	Tipo	Versão
Ansible	Biblioteca	2.14.5
Bootstrap	Biblioteca	5
Django	Biblioteca	4.2
GitPython	Biblioteca	3.1.31
Postgresql	Banco de Dados	15
PyYAML	Biblioteca	6
Python	Linguagem	3.11

Tabela 2. Dependências de software utilizadas.

A biblioteca Ansible para Python oferece modelos padrão para aplicação, como as classes Playbook, Task e outras, além de fornecer a lógica para importação de arquivos YAML. Após serem manipulados na aplicação, esses arquivos são exportados utilizando a biblioteca PyYAML. Em seguida, a biblioteca GitPython é utilizada para salvar o código no repositório. O `DOO` possui licença Apache 2.0 e o código-fonte pode ser acessado em seu repositório público[5].

3.2. Arquitetura

Conforme ilustrado na Figura 1, a arquitetura do `DOO` é composta por cinco camadas, sendo que cada uma delas é formada por módulos que permitem a adição de novas tecnologias e ferramentas. As camadas são: (i) **dados**, responsável por servir a interface com dados sobre as solicitações de mudança do serviço e sobre o inventário de máquinas; (ii) **GUI**, responsável por apresentar as informações obtidos na camada superior e auxiliar na criação de tarefas para alterar o serviço; (iii) **codificação**, onde o código de alteração do serviço é gerado; (iv) **repositório**, camada responsável pelo armazenamento do código; e (v) **provisionamento**, onde o código será executado e as modificações serão consolidadas.

3.2.1. Camadas

A seguir, as camadas serão apresentadas com maiores detalhes, seguindo a metodologia Top-Down, iniciando pela camada mais alta (dados) e avançando até a camada mais baixa (provisionamento).

- **Dados:** Engloba as ferramentas utilizadas pela equipe de TI para registrar as solicitações de criação, alteração ou exclusão do serviço e para monitoramento. Também inclui as ferramentas de inventário dos ativos de TI. Essa camada fornecerá informações sobre as solicitações de mudanças do serviço e os ativos que serão modificados. Essas informações podem ser obtidas via APIs e alimentarão a interface da aplicação.
- **GUI:** A interface gráfica web apresentará as informações obtidas na camada superior e guiará o usuário na criação de tarefas, na escolha dos servidores-alvo e na manipulação dos serviços. Tarefas comuns, como instalar um programa em um servidor, alterar arquivos de configuração e adicionar informações a um serviço,

[5]`https://github.com/DevOpsOrchestrator/DOO`

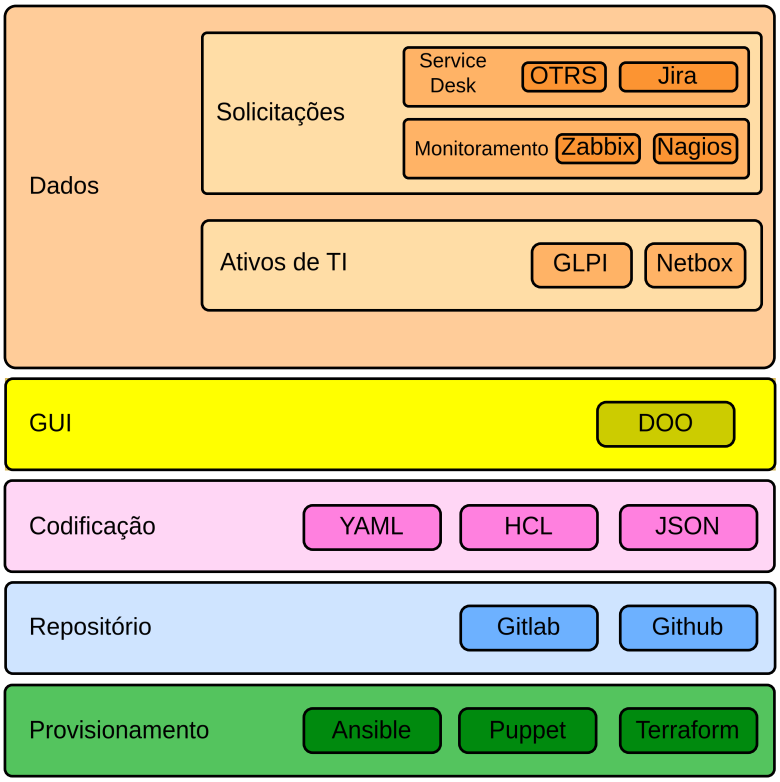

Figura 1. Arquitetura proposta do DOO.

serão executadas na interface. Os inputs realizados na interface serão transmitidos para a camada de codificação.

- **Codificação:** Nessa camada, será gerado o código necessário para a manutenção do serviço. Cada ferramenta de IaC utiliza uma linguagem específica para definir a infraestrutura, e a escolha da linguagem nesta camada determinará a ferramenta utilizada para o provisionamento da mudança;
- **Repositório:** É responsável por armazenar e versionar o código criado na camada de codificação. O versionamento permite e facilita a recuperação do ambiente para qualquer estado definido. A ferramenta utilizada nessa camada deve implementar o uso de pipelines, com isso, cada modificação salva no repositório gera um pipeline que realizará a alteração nos servidores selecionados na camada anterior;
- **Provisionamento:** Essa camada se comunica com os servidores-alvo e aplica as configurações armazenadas no repositório. Além disso, ela é responsável por orquestrar essas mudanças, caso afetem mais de um servidor. O pipeline criado na camada anterior, possui a ferramenta de provisionamento que executará o código. A camada de provisionamento possui as credenciais necessárias para acessar e realizar as alterações.

Através da Figura 2 é possível visualizar a integração dos softwares com a ferramenta e como eles se comunicam com o objetivo de automatizar um serviço. Na (1) **camada de dados** os dados das possíveis alterações estão nos softwares utilizados pela equipe de TI, assim como as informações sobre a rede e seus servidores, que estão armazenadas em softwares específicos para o gerenciamento da infraestrutura de redes. Esses dados alimentam o (2) **DOO** por meio de uma API REST. Com base nessas informações

e por meio de inputs na interface gráfica, na (3) **camada de codificação**, o DOO gera o código Ansible que será versionado nos projetos do (4) **Gitlab**, acionando, assim, o Pipeline que irá provisionar os (5) **serviços**.

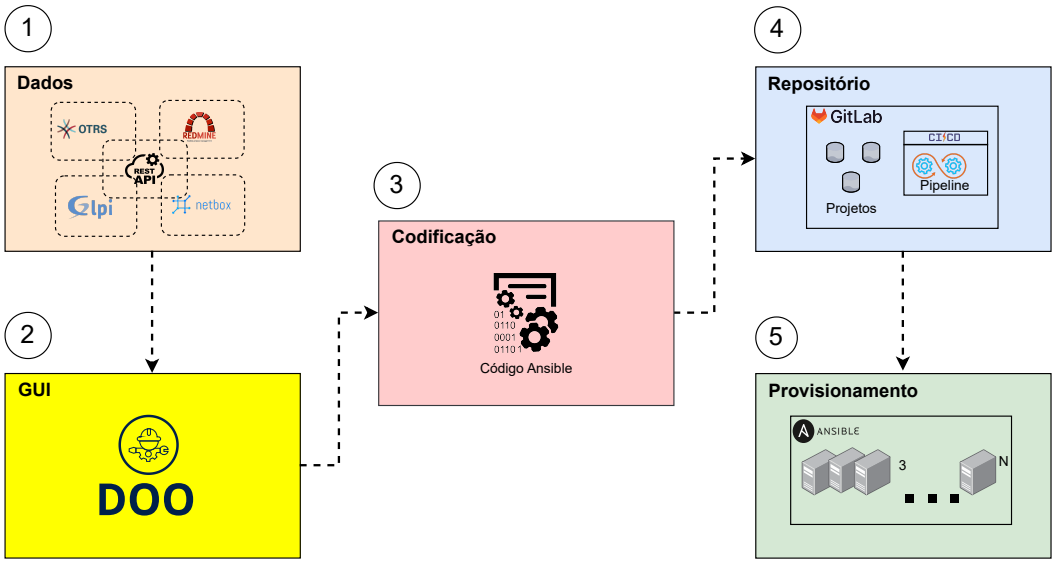

Figura 2. Diagrama de integração de softwares

Na interface é possível criar o inventário de servidores que serão alvos das alterações. Essa funcionalidade proporciona uma maneira conveniente e organizada de gerenciar os servidores envolvidos no processo de automação. Esses dados são criptografados utilizando a ferramenta Vault[6]. O Ansible, por sua vez, utiliza o *Secure Shell* (SSH), um protocolo de rede seguro amplamente adotado. O protocolo SSH estabelece uma conexão segura e criptografada que permite aos usuários se conectarem e controlarem dispositivos remotamente em uma rede. Essa abordagem garante a confidencialidade e integridade das informações transmitidas, bem como a autenticação dos usuários envolvidos.

O DOO possibilita o cadastro de variáveis específicas para os servidores, incluindo as credenciais de acesso necessárias. Essa funcionalidade é fundamental para garantir a correta autenticação e autorização dos usuários que interagem com os servidores remotos.

3.3. Limitações

Embora tenha sua vantagens, já foram identificadas algumas limitações durante o desenvolvimento do DOO que deverão ser resolvidas em novas versões. O DOO atualmente trabalha com Ansible para a criação do código de IaC. Portanto, as telas são voltadas para a criação do código utilizando essa linguagem, o que impossibilita a conversão do código para outras linguagens. Autores como [Chiari et al. 2023] e [Artac et al. 2018], optaram por criar uma linguagem própria para representar a infraestrutura, possibilitando a exportação desse código para outras linguagens de IaC. No entanto, mapear os serviços de rede e todas as possíveis configurações para uma linguagem própria que permita a

[6]https://docs.ansible.com/ansible/2.9/user_guide/vault.html

exportação para as linguagens mais utilizadas de IaC requer um esforço considerável que não será considerado na etapa atual de desenvolvimento do DOO.

Ainda sobre a linguagem Ansible, a primeira versão do DOO não contempla todas as funcionalidades da linguagem, tais como a ação *block* que agrupa um conjunto de *tasks*; *loop* para iterar em uma lista de itens; *when* para adicionar uma condição de execução de uma *task*; e a criação e uso de *roles*, que são utilizadas para agrupar vários *playbooks*. O DOO utiliza exclusivamente a *collection* padrão do Ansible, conhecida como *Ansible Builtin*. As *collections* no Ansible são pacotes modulares que organizam e distribuem funcionalidades relacionadas, como módulos, papéis e plugins.

Atualmente, o DOO não realiza testes automatizados para verificar se as implantações foram realizadas com sucesso e identificar possíveis impactos na infraestrutura, contando apenas com a verificação padrão do Ansible durante a execução das tarefas. A verificação contínua do código e os testes automatizados podem ser adicionados manualmente no pipeline do repositório.

4. Estudos de Casos

O DOO foi validado através de três estudos de casos em um testbed emulado e controlado. Os casos de uso escolhidos foram: (i) Serviço Web, que refere-se à instalação e configuração de um serviço Web através da configuração de um servidor Apache; (ii) Serviço de DNS, onde é feita a instalação e gerenciamento de um serviço DNS por meio da configuração de um servidor Berkeley Internet Name Domain 9 (BIND9); e (iii) Serviço de Firewall, onde é realizado o gerenciamento de um serviço de firewall por meio da configuração das regras do iptables. O repositório do código-fonte do DOO[7] contém os passos de reprodutibilidade para todos os casos de uso. Por esta razão, este artigo não traz detalhes sobre a execução de comandos.

Testbed

Para a montagem do testbed foi utilizado o emulador de rede Containernet [Peuster et al. 2018] que, dentre outras coisas, permite a criação e o gerenciamento de redes virtuais compostas por contêineres interconectados, proporcionando um ambiente flexível e escalável para testes e experimentos. Uma das principais vantagens do Containernet é a capacidade de emular redes complexas e distribuídas utilizando contêineres isolados. Cada contêiner representa um dispositivo de rede, como roteadores, comutadores ou servidores, e pode executar diferentes aplicativos.

O funcionamento da solução depende de quatro contêineres que possuem dependências entre si. Os contêineres são:

- DevOpsOrchestrator[8] - consiste na própria aplicação que inclui a interface gráfica acessível por meio de um navegador na porta 8000;
- Postgresql[9] - onde está o banco de dados utilizado pelo DOO. Foi necessário criar uma imagem nova, pois a padrão não tinha os pacotes *net-tools*, *iproute2* e *ethtool*, necessários para as configurações realizadas pelo containernet;

[7] https://github.com/DevOpsOrchestrator/DOO/
[8] https://hub.docker.com/r/devopsorchestrator/doo
[9] https://hub.docker.com/r/devopsorchestrator/postgres_validacao

- GitLab[10] - repositório onde o código Ansible gerado pelo DOO é versionado. Pode ser acessado por meio do navegador na porta 80;
- GitLab Runner[11] - utilizado pelo GitLab para executar o pipeline que realizará as modificações nos servidores alvo via SSH. Foi criado uma imagem personalizada com a instalação do Ansible.

O Containernet utiliza um script Python para definir a topologia da rede e toda configuração interna necessária. Para tanto, basta importar a biblioteca e criar os servidores, bem como as conexões entre eles. Cada estudo de caso terá um script Python contendo a infraestrutura da rede, que inclui os contêineres mencionados anteriormente, além de outros necessários para o estabelecimento do ambiente de validação, como terminais que representam usuários finais ou mesmo administradores de rede. O script de execução completo de cada estudo de caso pode ser obtido através do repositório do código-fonte do DOO.

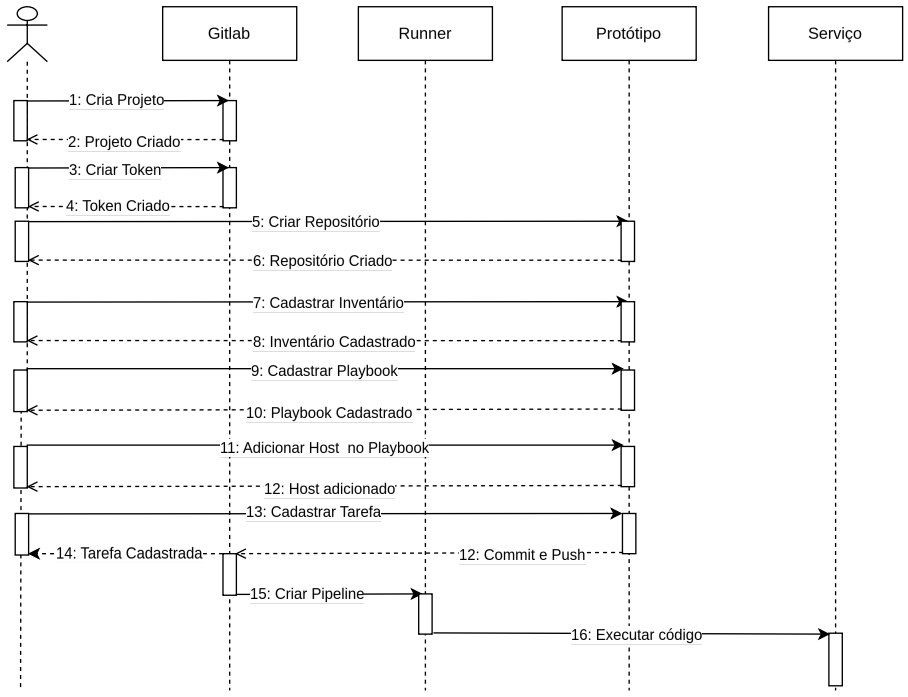

Figura 3. Fluxo para alterar o serviço

Essa configuração inicial servirá como base para os casos de uso, permitindo o acesso ao GitLab e ao DOO. Para realizar as alterações no serviço será necessário seguir o fluxo descrito na Figura 3, onde o administrador de rede realiza as seguintes etapas: cadastra um projeto no GitLab, cadastra um token de acesso ao projeto, acessar o DOO e cadastra um repositório. A partir desse ponto, é possível configurar o serviço por meio de tarefas que serão inseridas através da interface gráfica.

A partir de agora os estudos de casos serão apresentados em maiores detalhes. Para fins de conveniência, o repositório do código-fonte do DOO possui todos os passos de reprodutibilidade necessários dos estudos de casos apresentados abaixo.

[10] https://hub.docker.com/r/gitlab/gitlab-ce
[11] https://hub.docker.com/r/devopsorchestrator/gitlab_runner

4.1. Serviço Web

Este estudo de caso refere-se à instalação e gerenciamento de um serviço Web por meio da configuração de um servidor Apache. Como consequência, um cliente poderá acessar a página web por meio do navegador. A Figura 4 ilustra a topologia de rede deste estudo de caso. Ela consiste em cinco contêineres e um *switch*. Dois contêineres abrigam a aplicação e o banco de dados do DOO, enquanto outros dois contêineres compõem o repositório, sendo eles o Gitlab e o Gitlab Runner. Além disso, há um contêiner que representa uma máquina com o sistema operacional Ubuntu, onde o serviço Web será instalado.

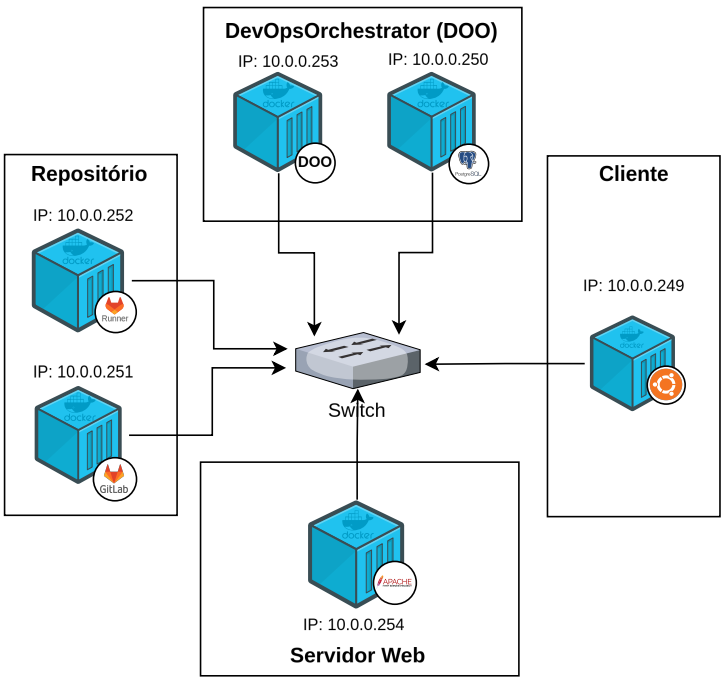

Figura 4. Topologia de rede serviço Web

A instalação de um serviço Web normalmente envolve várias etapas, como acessar o servidor, instalar pacotes e realizar configurações manualmente. Executar essas tarefas em vários servidores pode ser custoso. No DOO é possível cadastrar um inventário de máquinas com suas credenciais de acesso, o que permite definir os servidores alvos das configurações de forma mais eficiente.

Após adicionar um inventário no DOO, a próxima etapa consiste em criar o arquivo que agrupará as tarefas de configuração e selecionar os servidores a serem modificados a partir do inventário. As tarefas são criadas utilizando a interface do usuário, na qual são selecionadas as ações a serem realizadas. Para a instalação do serviço Web, foi definida a ação APT, que utiliza o gerenciador de pacotes do Ubuntu para instalar o pacote apache2, além da especificação da versão do pacote. Com a tarefa de instalação criada, é necessário definir um procedimento para iniciar o serviço. Para isso, é criado um *handler*, uma funcionalidade que permite selecionar uma ação a ser executada após o término de uma tarefa. No caso em questão, um *handler* é configurado e utiliza a ação `sysvinit`, um sistema de inicialização de serviços do Linux.

Após completar todas essas etapas na interface do usuário, o DOO gera o código Ansible necessário e o armazena no projeto criado no Gitlab. Isso inicia um pipeline que

executa o código e provisiona as modificações nos servidores. Assim, temos a definição do serviço por meio do código, o qual pode ser implantado em vários servidores. Com a utilização do DOO foi possível instalar o serviço Web e torná-lo acessível. O DOO agiliza a construção do código, mantendo as boas práticas de desenvolvimento, versionando o código e proporcionando um ponto único de modificação.

4.2. Serviço DNS

Este estudo de caso refere-se à instalação e gerenciamento de um serviço DNS por meio da configuração de um servidor BIND9[12]. Utilizando à ferramenta DOO será possivel adicionar novas zonas, permitindo que um cliente acesse um servidor através da resolução de nomes. Na Figura 5 podemos visualizar a topologia de rede deste estudo de caso. Ela consiste em sete contêineres e um *switch*. Dois contêineres abrigam a aplicação e o banco de dados do DOO, enquanto outros dois contêineres compõem o repositório, sendo eles o Gitlab e o Gitlab Runner. Além disso, existem dois contêineres que representam os servidores alvos, um para o servidor Web e outro para o sistema operacional Ubuntu, onde o Serviço DNS será instalado. Também há um outro contêiner que representa uma máquina cliente.

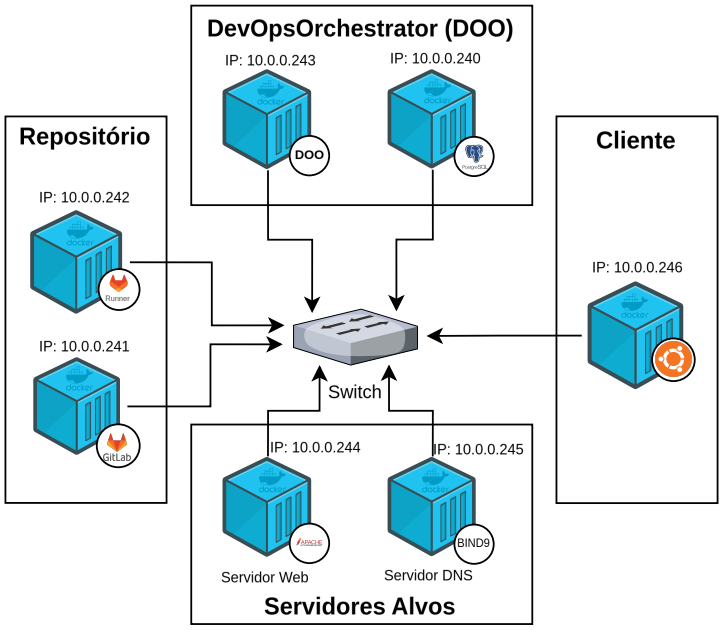

Figura 5. Topologia de rede serviço DNS

Servidores de nomes como o BIND9, uma vez instalados, exigem um trabalho considerável para gerenciar as zonas, pois envolvem etapas de criação e edição de vários arquivos para adicionar ou atualizar uma zona. Isso pode levar a erros devido à interação humana, resultando em arquivos sem padrões e sem histórico de modificações. A funcionalidade de template do DOO auxilia no gerenciamento desse serviço.

Após criar as tarefas de instalação e inicialização do BIND9 no DOO, é possível criar *templates* que facilitarão o provisionamento dessas mudanças. Para adicionar uma nova zona, é necessário modificar o arquivo `named.conf` adicionando as linhas que

[12] https://www.isc.org/bind/

indicam a localização do arquivo da zona. Em seguida, é preciso criar um arquivo de zona seguindo um padrão específico e, por fim, reconfigurar o servidor executando um comando `rndc`. Para isso, dentro do *template* são definidas três tarefas: a primeira utiliza a ação `blockinfile` do Ansible para adicionar uma nova linha no `named.conf`; a segunda tarefa utiliza a ação `copy` para criar um arquivo de zona; enquanto a última tarefa utiliza a ação `shell` para executar o comando de reconfiguração do BIND9. Essas ações fazem uso de variáveis que serão solicitadas durante o provisionamento.

Após a criação do *template*, pode-se prosseguir com a criação de uma solicitação de mudança no `DOO`, onde serão inseridos os dados da zona que será criada. Assim, na tela de provisionamento, será possível selecionar o *template* e adicionar uma zona com apenas um clique. O `DOO` possibilita agilizar consideravelmente o processo de criação de uma zona, eliminando a necessidade de acesso manual aos servidores, padronizando os arquivos de configuração e reduzindo possíveis erros. Assim, o cliente que solicita a criação de uma zona pode ter seu pedido atendido, após o provisionamento, acessar o servidor por meio do seu nome.

4.3. Serviço de Firewall

Este estudo de caso refere-se ao gerenciamento de um serviço de firewall por meio da configuração das regras do iptables[13] em um servidor. Com a utilização do `DOO`, será possível habilitar o encaminhamento de pacotes entre duas redes e realizar a sua filtragem. Dessa forma, o cliente de uma rede só poderá acessar os servidores localizados em outra rede nas portas especificadas no firewall. A Figura 6 ilustra a topologia de rede deste estudo de caso. Ela consiste em seis contêineres e dois *switches*. Dois contêineres abrigam a aplicação e o banco de dados do `DOO`, enquanto outros dois contêineres compõem o repositório, sendo eles o Gitlab e o Gitlab Runner. Além disso, há um contêiner com o sistema operacional Ubuntu que desempenhará o papel de firewall, onde serão realizadas as configurações de roteamento e filtragem de pacotes entre as duas redes. Também existe um contêiner que representa uma máquina cliente.

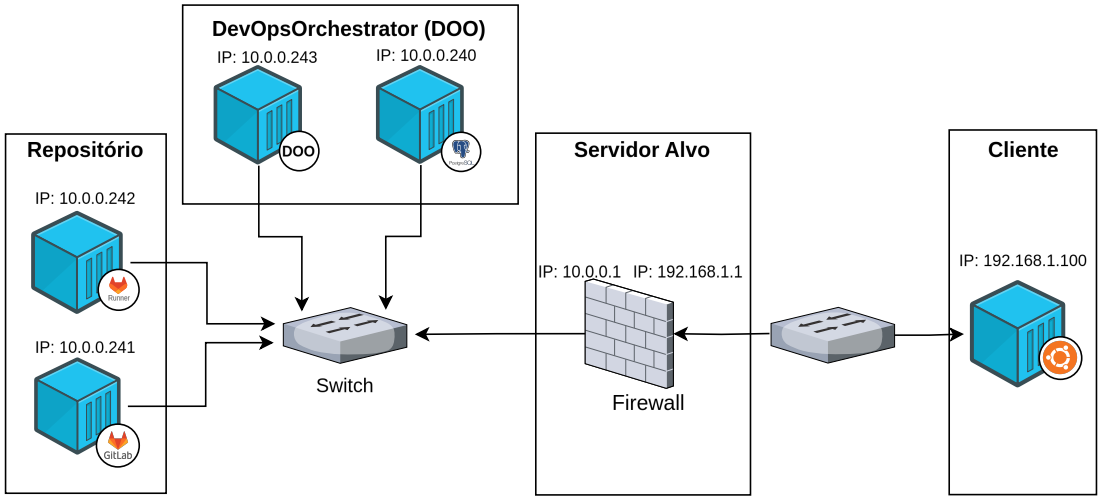

Figura 6. Topologia de rede serviço Firewall

[13] `https://www.netfilter.org/projects/iptables/index.html`

A configuração correta do firewall é vital para proteger a rede contra ameaças e manter a integridade dos dados. A manutenção manual pode levar a problemas como atualizações de regras inconsistentes e brechas de segurança. Automatizar essa manutenção é essencial para garantir a eficácia contínua das defesas cibernéticas, minimizando o risco de ataques e mantendo a segurança da infraestrutura. Neste caso de uso, para possibilitar que as duas redes da topologia fossem acessíveis, foram criadas duas tarefas: a primeira utiliza a ação `lineinfile` do Ansible para adicionar uma linha ao arquivo `sysctl.conf`, permitindo o encaminhamento dos pacotes entre as redes; já a segunda tarefa utiliza a ação `shell` para executar uma regra iptables que realiza o *Network Address Translation* (NAT) entre as redes.

Com as redes acessíveis, foram criados dois *templates* para gerenciar o bloqueio e a liberação de portas no firewall. Tipicamente, cada *template* possui uma tarefa que utiliza a ação `shell` do Ansible para executar um *script* iptables. No *template* de bloqueio, a porta a ser bloqueada é passada como variável e o *script* verifica se a porta está liberada e, em seguida, realiza o bloqueio. Já no *template* de liberação, o *script* verifica se a porta está bloqueada e a libera. Realizadas as configurações necessárias, é possível responder a uma solicitação de mudança, bloqueando ou liberando portas no firewall com apenas alguns cliques. Isso agiliza as respostas às solicitações, elimina os erros associados à execução manual de regras e mantém um histórico de atualizações no serviço. Com isso, podemos gerenciar as portas do firewall, limitando o acesso do cliente por meio do provisionamento das mudanças utilizando os templates no `DOO`.

5. Conclusão

Este trabalho apresentou uma solução integrada de software denominada de DevOpsOrchestrator que emprega a abordagem de infraestrutura como código (IaC) para o gerenciamento de serviços de rede. Os principais trabalhos relacionados identificados da literatura foram apresentados, onde destacou-se a relevância do `DOO` se comparado a estes trabalhos. Também foi estabelecido um ambiente de validação no qual foram conduzidos três casos de uso para automatização de serviços. Os resultados indicaram que é viável administrar serviços de rede por meio de uma interface gráfica, utilizando os princípios de IaC para provisionamento de serviços. Para trabalhos futuros, considera-se superar as limitações atualmente identificadas e desenvolver uma Domain-Specific Language (DSL) que represente as configurações de diversos serviços, permitindo a geração de código para as principais linguagens de IaC utilizadas no mercado.

Referências

Artac, M., Borovšak, T., Di Nitto, E., Guerriero, M., Perez-Palacin, D., and Tamburri, D. A. (2018). Infrastructure-as-code for data-intensive architectures: a model-driven development approach. In *2018 IEEE international conference on software architecture (ICSA)*, pages 156–15609. IEEE.

Bertrand, B., Armanet, S., Christensson, J., Curri, A., Harrisson, A., and Mudingay, R. (2020). ICS Infrastructure Deployment Overview at ESS. In *Proc. ICALEPCS'19*, number 17 in International Conference on Accelerator and Large Experimental Physics Control Systems, pages 875–879. JACoW Publishing, Geneva, Switzerland. https://doi.org/10.18429/JACoW-ICALEPCS2019-WEAPP04.

Brikman, Y. (2017). Terraform: Up and running, first edit. *O'Reilly Media*.

Chiari, M., Xiang, B., Nedeltcheva, G. N., Di Nitto, E., Blasi, L., Benedetto, D., and Niculut, L. (2023). Doml: A new modelling approach to infrastructure-as-code. In *International Conference on Advanced Information Systems Engineering*, pages 297–313. Springer.

Hariyadi, I. P. and Marzuki, K. (2020). Implementation of configuration management virtual private server using ansible. *MATRIK: Jurnal Manajemen, Teknik Informatika dan Rekayasa Komputer*, 19(2):347–357.

Jiang, Y. and Adams, B. (2015). Co-evolution of infrastructure and source code-an empirical study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 45–55. IEEE.

Kundnani, S. K. (2019). Delivering network automation - cisco.

Masek, P., Stusek, M., Krejci, J., Zeman, K., Pokorny, J., and Kudlacek, M. (2018). Unleashing full potential of ansible framework: University labs administration. In *2018 22nd conference of open innovations association (FRUCT)*, pages 144–150. IEEE.

McGhee, E. J., Krobatsch, T., and Milton, S. (2022). Netinfra - a framework for expressing network infrastructure as code. In *Practice and Experience in Advanced Research Computing*, PEARC '22, New York, NY, USA. Association for Computing Machinery.

Morris, K. (2016). *Infrastructure as code: managing servers in the cloud*. "O'Reilly Media, Inc.".

Peuster, M., Kampmeyer, J., and Karl, H. (2018). Containernet 2.0: A rapid prototyping platform for hybrid service function chains. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 335–337.

Pires, A., Jr., P. M., Pessoa, D., Matos, F., and Santos, A. (2021). Pipeconf: Uma arquitetura integrada para configuração automatizada de ativos de rede heterogêneos. In *Anais do XXVI Workshop de Gerência e Operação de Redes e Serviços*, pages 110–123, Porto Alegre, RS, Brasil. SBC.

Rodosek, G. D. (2002). *A Framework for IT Service Management*. PhD thesis, Ludwig Maximilian University of Munich.

Shah, J. A. and Dubaria, D. (2019). Netdevops: A new era towards networking & devops. In *2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON)*, pages 0775–0779. IEEE.