Provisionamento Automatizado de Infraestruturas usando Grandes Modelos de Linguagem

Matheus Kobashigawa Luna[1], **Paulo Silas Severo de Souza**[1]

[1]Universidade Federal do Pampa (Unipampa)
Alegrete/RS - Brasil

{matheusluna.aluno, paulosilas}@unipampa.edu.br

Resumo. *Infraestruturas modernas de computação têm exigido cada vez mais automação e flexibilidade para atender às crescentes demandas das aplicações. Nesse sentido, abordagens como Infraestrutura como Código (Infrastructure as Code — IaC) têm ganhado destaque, permitindo a definição e o gerenciamento de recursos de infraestrutura por meio de código. Apesar de seus benefícios, a implementação de IaC pode ser desafiadora, especialmente em ambientes de larga escala, já que tecnologias de IaC tipicamente implementam sintaxes específicas com estruturas que podem se tornar complexas de acordo com a quantidade de recursos e configurações necessárias. Este trabalho propõe a utilização de Grandes Modelos de Linguagem (Large Language Models - LLMs) para automatizar tarefas de IaC, facilitando o provisionamento e gerenciamento de infraestruturas computacionais. A abordagem inclui a geração de código IaC a partir de descrições textuais em Português ou Inglês, bem como a tradução de comandos IaC para linguagem natural.*

1. Introdução

Operadores de infraestrutura de TI têm sido desafiados a provisionar e gerenciar ambientes computacionais que consigam atender às demandas cada vez mais estritas de aplicações modernas, que frequentemente requerem alta disponibilidade, escalabilidade e baixa latência [Buyya et al. 2018] [Satyanarayanan et al. 2019]. Nesse contexto, DevOps [Ebert et al. 2016] emerge como uma abordagem que visa integrar boas práticas de desenvolvimento de software (Dev) com operações de TI (Ops). Dentre as promessas de DevOps, destaca-se a mitigação de erros humanos e a aceleração em especial da parte final do ciclo de desenvolvimento de software, que envolve o provisionamento e a manutenção de ambientes de TI.

Para que as expectativas de DevOps sejam atendidas, estratégias de automação precisam ser implantadas para garantir a consistência e a eficiência das operações [Bass 2017]. Nesse contexto, um dos processos automatizados é o provisionamento da infraestrutura, que serve como base para demais atividades de desenvolvimento e operação. Assim, IaC surge como uma prática fundamental no ecossistema DevOps, permitindo a definição e o gerenciamento de recursos de infraestrutura por meio de código [Rahman et al. 2019].

Ferramentas de IaC agem como intermediárias entre descrições alto nível dos recursos necessários e a configuração efetiva dos ambientes. Dada a complexidade e a diversidade de infraestruturas computacionais, ferramentas de IaC tipicamente adotam Linguagens de Domínio Específico (*Domain-Specific Languages* — DSLs) ou implementam

estruturas sintáticas bastante específicas para garantir que a tradução das descrições em código seja precisa [Guerriero et al. 2019].

Este trabalho propõe a utilização de modelos de Inteligência Artificial (IA), em especial LLMs [Raiaan et al. 2024], para simplificar a geração de arquivos de IaC. Especificamente, a proposta é desenvolver um sistema que traduza descrições textuais em comandos de IaC e vice-versa, facilitando a criação e a manutenção de scripts de IaC por operadores de infraestrutura.

O restante deste trabalho está organizado como segue. A Seção 2 apresenta o referencial teórico, abordando conceitos de DevOps, IaC e LLMs. A Seção 4 detalha a solução proposta, incluindo a arquitetura do sistema e detalhes de implementação. Por fim, a Seção 5 apresenta as considerações finais, discutindo os resultados preliminares obtidos e listando passos futuros para a continuidade do trabalho.

2. Referencial Teórico

Esta seção discute os principais conceitos que fundamentam este trabalho, incluindo DevOps e sua transformação no ciclo de vida de desenvolvimento de software, bem como a automação de infraestrutura por meio de IaC e o potencial de Grandes Modelos de Linguagem na automação de processos de TI.

2.1. DevOps

Profissionais de TI e empresas de tecnologia têm enfrentado desafios significativos no desenvolvimento de software, devido à crescente complexidade e criticidade dos requisitos dos projetos. Por um lado, há uma pressão por entregas rápidas para garantir a competitividade dos clientes no mercado. Por outro lado, os requisitos de desempenho têm se tornado cada vez mais estritos, especialmente em casos de uso emergentes (por exemplo, Realidade Virtual e Aumentada, Internet das Coisas, etc.) [Satyanarayanan et al. 2019].

Um dos desafios para a realização de entregas rápidas e de alta qualidade consiste na discrepância entre equipes de desenvolvimento e operações [Ebert et al. 2016]. Por um lado, equipes de desenvolvimento tendem a introduzir mudanças frequentes nas aplicações, principalmente motivadas por metodologias ágeis. Por outro lado, equipes de operações tipicamente recebem a incumbência de manter a estabilidade do ambiente computacional, e por isso, comumente estabelecem processos rígidos de implantação e manutenção, que acabam desacelerando o ciclo de desenvolvimento.

DevOps surge com a ideia de aproximar equipes de desenvolvimento e operações para otimizar o ciclo de vida de desenvolvimento de software, mantendo tanto a agilidade quanto a estabilidade dos sistemas desenvolvidos [Ebert et al. 2016] [Zhu et al. 2016]. Para isso, uma das principais premissas de DevOps é a automação de processos, que visa reduzir o tempo entre a concepção de uma mudança no sistema e sua implantação em produção, ao passo que mitigando a ocorrência de erros humanos. Mesmo que o conceito de DevOps seja de certa forma abstrato, há algumas práticas frequentemente associadas a ele que se destacam no processo de implementação da ideia:

- **Integração Contínua (do inglês: *Continuous Integration* - CI) [Shahin et al. 2017]:** estimula desenvolvedores a integrar com frequência o código produzido em repositórios compartilhados. A cada integração, são

realizados testes automatizados para verificar a qualidade do código e sua compatibilidade com o restante da aplicação.

- **Entrega Contínua (do inglês: *Continuous Delivery* - CD) [Shahin et al. 2017]:** estende a ideia de CI, permitindo que as aplicações sejam disponibilizadas de forma automatizada em ambientes de teste (para garantir a qualidade do software em uma espécie de continuação do processo de CI) e produção (onde de fato os usuários finais acessam o software).

Mesmo com fluxos (tipicamente conhecidos como *pipelines*) de CI/CD bem definidos, o processo de implantação de software é bastante desafiador, especialmente por conta da dependência com a infraestrutura de TI. Mais especificamente, mesmo que o software seja disponibilizado de forma automatizada, sua execução depende de uma infraestrutura computacional previamente configurada. Nesse sentido, abordagens de automação de infraestrutura vêm à tona como elementos críticos para a efetiva implementação de DevOps [Ebert et al. 2016].

2.2. Infraestrutura como Código

Uma das abordagens mais populares para a automação de infraestrutura é conhecida como IaC [Morris 2016]. A ideia central de IaC é permitir que operadores de TI definam e gerenciem recursos de infraestrutura (por exemplo, servidores, recursos de rede e armazenamento, etc.) por meio de código. Como resultado, além de reduzir a ocorrência de erros humanos gerados durante configurações manuais, IaC facilita a reprodutibilidade no nível de gerência de infraestruturas, o que pode gerar benefícios em ambientes de larga escala.

A adoção de IaC envolve algumas decisões de projeto relacionadas à flexibilidade do processo de automação (modelos de mutabilidade) e à maneira na qual as instruções de automação são especificadas (paradigmas de programação):

- **Modelo de Infraestrutura Mutável:** permite que a infraestrutura seja modificada após ser provisionada. Tal modelo enfatiza a flexibilidade, pois viabiliza ajustes finos após a implantação, mas abre espaço para inconsistência de configuração, o que pode dificultar a reprodutibilidade das automações.
- **Modelo de Infraestrutura Imutável:** não permite modificações na infraestrutura após a etapa de provisionamento. Favorece a consistência, pois a infraestrutura sempre reflete as instruções dos arquivos de configuração, ao passo que torna a reconfiguração da infraestrutura um processo mais custoso.
- **Paradigma Imperativo:** Baseia-se em comandos específicos que descrevem como a infraestrutura deve ser configurada, sendo a execução desses comandos realizada na ordem especificada.
- **Paradigma Declarativo:** Baseia-se em descrições do estado desejado da infraestrutura, permitindo que ferramentas de IaC configurem o ambiente automaticamente com base nessas descrições e em suas próprias regras de negócio internas (que podem definir, por exemplo, a ordem de execução dos comandos).

Dada a ampla gama de ações englobadas no âmbito de automação de infraestrutura, ferramentas de IaC tipicamente se especializam em tarefas específicas, como configuração de servidores, instalação de software básico e provisionamento de aplicações. Para a etapa de implantação de software, uma das ferramentas mais populares é o Ansible[1]. Dentre os diferenciais do Ansible, destaca-se sua abordagem conhecida

[1] `https://www.ansible.com/`

como “sem agentes”, que dispensa a instalação de componentes específicos nos servidores gerenciados. A Figura 1 apresenta uma visão geral dos componentes que compõem a arquitetura do Ansible.

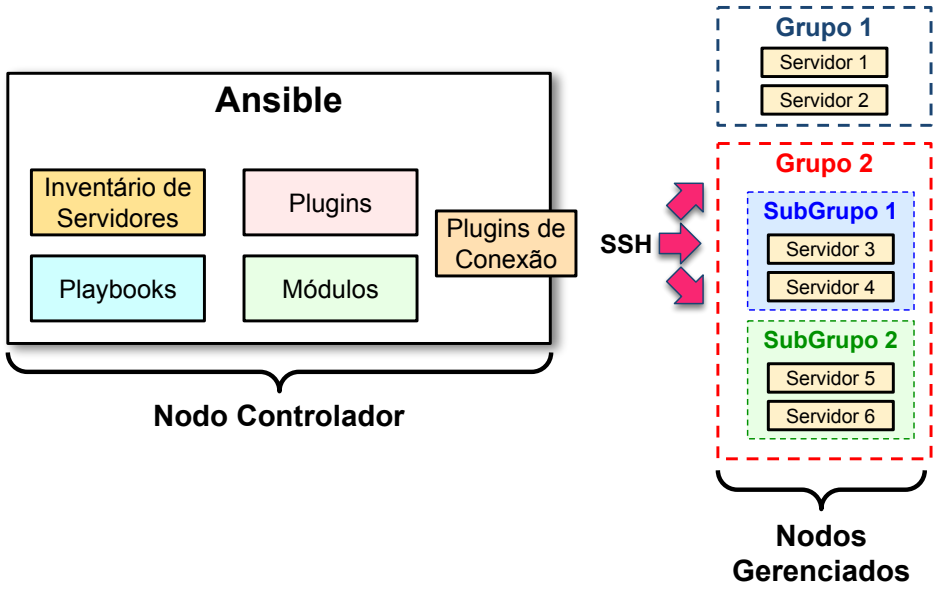

Figura 1. Visão geral da arquitetura do Ansible.

A arquitetura do Ansible baseia-se em dois componentes principais: o nodo controlador e os nodos gerenciados. Enquanto o nodo controlador fica responsável por executar os comandos do Ansible, os nodos gerenciados compreendem a infraestrutura-alvo (isto é, onde os recursos são provisionados). Como forma de simplificar a comunicação entre o nodo controlador e os nodos gerenciados, o Ansible utiliza o protocolo *Secure Shell* (SSH), que requer apenas que o nodo controlador tenha as credenciais de acesso aos nodos gerenciados.

No nodo controlador, o Ansible utiliza um componente chamado Inventário para listar os nodos gerenciados, que permite inclusive a definição de grupos de servidores (por exemplo, produção, teste, desenvolvimento, etc.). Como o Ansible adota o paradigma declarativo, o nodo controlador conta com módulos e plugins para encapsular comandos que devem ser executados nos nodos gerenciados. Com base nos módulos e plugins existentes, o Ansible viabiliza a criação de Playbooks, que são arquivos escritos no formato YAML[2] (do inglês: *YAML Ain’t Markup Language*) que descrevem as tarefas a serem executadas nos nodos gerenciados.

Apesar de sua arquitetura simplificada que facilita a automação baseada na abordagem “sem agentes”, o uso efetivo do Ansible pode se tornar complexo em ambientes de larga escala. Dentre os motivos para tal complexidade, destacam-se a sintaxe específica do YAML (que requer cuidados com indentação e formatação) e a quantidade de convenções necessárias para que o Ansible entenda corretamente as instruções de automação.

[2] `https://yaml.org/`

2.3. Grandes Modelos de Linguagem

Modelos de IA têm ganhado cada vez mais destaque em diversas áreas do conhecimento por automatizar tarefas com base na identificação de padrões em grandes volumes de dados. Dentre as ramificações de IA encontra-se o Processamento de Linguagem Natural (PLN), que visa a criação de modelos capazes de compreender e gerar linguagem humana [Chowdhary and Chowdhary 2020]. A notoriedade do PLN se dá pela vasta gama de aplicações, que englobam desde sumarização de textos até conversação automatizada (aplicações estas que são comumente chamadas de *chatbots*).

Um dos grandes desafios de PLN é a própria representação da linguagem natural, que é intrinsecamente não estruturada [Alammar and Grootendorst 2024]. Por um lado, conteúdos em linguagem natural são altamente variáveis, por exemplo em termos de significado e estilo, dependendo do contexto. Por outro lado, dados representados em computadores tendem a seguir padrões bem-definidos para facilitar tarefas como indexação e operações lógicas. Dados os casos de uso promissores de PLN e sua complexidade inerente, diversos estudos têm sido realizados ao longo dos anos no âmbito da representação de linguagem de forma estruturada.

Dentre os avanços mais significativos em PLN, destaca-se a concepção do conceito de "atenção", que permite que modelos identifiquem partes relevantes de uma sequência de entrada [Alammar and Grootendorst 2024]. A implementação de modelos com atenção abriu caminho para o desenvolvimento de aplicações de IA mais eficientes em tarefas como tradução automática e sumarização de textos, onde entender a relação entre elementos de uma sequência (por exemplo, palavras) é crucial para a geração de resultados precisos.

A adoção de atenção em modelos de PLN foi um passo crucial para o desenvolvimento de LLMs [Alammar and Grootendorst 2024], isto é, modelos de IA tipicamente treinados com grandes volumes de dados textuais, visando a captura de padrões como a probabilidade de ocorrência de palavras em determinadas sequências. Diversos LLMs têm ganhado popularidade, como GPT [Achiam et al. 2023] (do inglês: *Generative Pre-trained Transformer*) e Gemini [Team et al. 2023]. Tais modelos têm sido aplicados em uma ampla gama de tarefas, dentre elas a geração e correção de códigos-fonte para cenários como automação de processos de TI.

3. Trabalhos Relacionados

Esta seção apresenta alguns estudos relacionados à automação de tarefas ligadas ao provisionamento de infraestruturas de TI usando LLMs. Além disso, apresenta-se um paralelo entre os esforços existentes e o escopo deste trabalho.

Kwon et al. [Kwon et al. 2023] exploram a aplicabilidade de LLMs no reparo automático de Playbooks do Ansible em infraestruturas Edge-Cloud. Os autores explicam que a automação bem-sucedida de componentes de infraestrutura em ambientes Edge-Cloud é crucial, especialmente dados os requisitos estritos das aplicações e a natureza distribuída e heterogênea dos servidores. Os autores exploram dois LLMs, GPT e Bard, no processo de revisão de 58 Playbooks. Os resultados obtidos indicam o potencial de LLMs na correção de Playbooks, com 70% dos casos avaliados apresentando respostas úteis. Diferente de Kwon et al. [Kwon et al. 2023], que não apresenta exemplos de

correções para os LLMs, o trabalho atual usa exemplos para melhorar a eficácia dos modelos na geração de Playbooks Ansible.

Na mesma linha de pesquisa, Low et al. [Low et al. 2024] investigam a correção automática de arquivos de configuração do Terraform, uma ferramenta de IaC amplamente utilizada para provisionamento de recursos de infraestrutura. Os autores focam no uso do modelo GPT-4, proposto pela empresa OpenAI. Ao invés de usarem o modelo em sua forma padrão, os autores propõem algumas práticas para adaptar o modelo às particularidades dos arquivos de configuração do Terraform. Primeiramente, o conteúdo dos arquivos de configuração é dividido em blocos, em vista das limitações de tamanho de contexto do LLM. Além disso, os autores adotam uma abordagem de interação com o LLM baseada em exemplos, alimentando o modelo com detalhes sobre problemas de configuração para facilitar a geração de correções precisas. Os resultados obtidos são promissores, com o LLM sendo capaz de corrigir 87,4% dos problemas testados. O trabalho atual possui um foco diferente do de Low et al. [Low et al. 2024]. Enquanto Low et al. [Low et al. 2024] focam em arquivos de configuração do Terraform, o presente trabalho visa a geração de Playbooks Ansible.

Por outro lado, Sahoo et al. [Sahoo et al. 2024a] apresentam o Ansible Lightspeed, um serviço baseado em um LLM proprietário chamado Watsonx Code Assistant. Em linhas gerais, o Ansible Lightspeed visa simplificar a geração de Playbooks Ansible a partir de descrições textuais. Os autores conduziram uma avaliação qualitativa com 10696 usuários, analisando a retenção dos usuários, suas edições nos Playbooks sugeridos e a percepção sobre as sugestões geradas. Os resultados obtidos indicam uma taxa de retenção de 13,66% no trigésimo dia, mostrando que o serviço conseguiu engajar efetivamente seus usuários. Além disso, 49,08% das sugestões geradas pelo Ansible Lightspeed foram aceitas sem edições críticas pelos usuários. O presente trabalho é complementar ao de Sahoo et al. [Sahoo et al. 2024a], no sentido de visar explorar modelos proprietários e de código aberto para a geração de Playbooks Ansible.

Sarda et al. [Sarda et al. 2024] propõem o uso de LLMs para geração de códigos Ansible para remediação de problemas em aplicações distribuídas baseadas na arquitetura de microsserviços. Os autores exploram o uso de dois LLMs: um proprietário (GPT-4) e outro de código aberto (LLaMa-2-70B). Os autores explicam que a escolha do modelo a ser utilizado depende do contexto da aplicação, especialmente em relação à sensibilidade dos dados envolvidos. Isto é, enquanto o GPT-4 é capaz de gerar bons resultados, ele pode não ser apropriado quando as entradas contêm dados confidenciais; por outro lado, modelos de código aberto, como o LLaMa-2-70B, podem ser implantados em ambientes locais, evitando a exposição de dados sensíveis. Os autores demonstram, através de experimentos, que os LLMs são capazes de gerar Playbooks Ansible eficientes na resolução de problemas em ambientes de microsserviços, com taxa média de correção de 98,86%. Ao contrário de Sarda et al. [Sarda et al. 2024], este trabalho não se limita somente à automatização do provisionamento de microsserviços.

Conforme discutido nessa seção, vários pesquisadores têm focado no uso de LLMs para automatizar tarefas de operações de TI. Diferente das abordagens existentes, o presente trabalho visa a implementação de um sistema que gera Playbooks Ansible a partir de descrições textuais em Português ou Inglês, bem como identifica problemas em Playbooks existentes e propõe correções. Além disso, ao invés de focar em um único

modelo de LLM, o trabalho atual propõe a avaliação de diferentes modelos. Por fim, explora-se uma abordagem baseada na passagem de exemplos para os modelos, visando aprimorar a eficácia das sugestões geradas.

4. Solução Proposta

Esta Seção apresenta a solução proposta para o uso de LLMs na geração de arquivos de configuração de infraestrutura com Ansible. Especificamente, a Seção 4.1 indica os objetivos do trabalho, a Seção 4.2 detalha a arquitetura preliminar da solução proposta, e por fim a Seção 4.3 discute os próximos passos da pesquisa.

4.1. Objetivos

Este trabalho visa criar um sistema que utiliza LLMs para criação e refatoração de arquivos de configuração do Ansible, com foco em reduzir a complexidade de infraestruturas amparadas por processos de automação. Como objetivos específicos, destacam-se:

1. Adotar técnicas de Engenharia de Prompt [Sahoo et al. 2024b] para otimização de LLMs em tarefas relacionadas à IaC.
2. Identificar combinações otimizadas de parâmetros de LLMs para geração de códigos de configuração de infraestrutura.
3. Avaliar LLMs com diferentes arquiteturas e tamanhos para identificar o impacto desses fatores em tarefas de IaC.

A seção seguinte detalha os resultados obtidos na primeira etapa da pesquisa, com foco na definição preliminar da solução proposta para o uso de LLMs na geração de arquivos de configuração de infraestrutura com Ansible.

4.2. Descrição da Solução

O sistema foi implementado em Python[3], linguagem escolhida por sua simplicidade e ampla disponibilidade de bibliotecas de IA. Como LLMs demandam alto poder computacional, requisito que muitas vezes está além da capacidade de computadores domésticos comuns, optou-se por utilizar uma Interface de Programação de Aplicações (*Application Programming Interface* — API) da plataforma HuggingFace[4]. Essa abordagem permite que a solução proposta utilize modelos de larga escala sem a necessidade de infraestrutura local robusta. No entanto, é importante observar que o uso da API do HuggingFace requer uma conta e solicitação de licença na plataforma.

Para adaptar o LLM à tarefa de gerar arquivos de configuração para o Ansible, utilizou-se a técnica de *Few Shot Learning* [Wang et al. 2020], que consiste em fornecer um contexto inicial e exemplos (*few shots*) junto às solicitações feitas ao modelo. Essa abordagem foi escolhida por não exigir treinamento adicional, reduzindo o custo computacional e o número de requisições à API, ao mesmo tempo que possibilita a adaptação do modelo às instruções específicas demandadas pelo sistema.

O sistema foi projetado inicialmente com uma interface de linha de comando com menus interativos que orientam o usuário em todas as etapas do processo. O fluxo do sistema, apresentado na Figura 2, funciona da seguinte forma:

[3]`https://www.python.org/`
[4]`https://huggingface.co/`

1. **Seleção do modelo:** Inicialmente, o usuário escolhe qual LLM será utilizado.
2. **Menu principal:** Em seguida, é exibido um menu com as opções de:
    (a) **Gerar um arquivo de configuração:** O sistema solicita uma descrição em linguagem natural, que será utilizada para criar o arquivo.
    (b) **Realizar o upload de um arquivo existente:** O usuário deve inserir o caminho completo do arquivo. Após o upload, o sistema apresenta uma análise detalhada do arquivo.
3. **Processamento do LLM:** Após a interação inicial, o sistema processa a entrada fornecida utilizando o modelo selecionado e apresenta os resultados no terminal.
4. **Feedback e ajustes:** O usuário tem a opção de revisar e refazer a descrição, caso queira alterar o resultado gerado. Se optar por modificar, será redirecionado ao passo de inserção de descrição.
5. **Salvar ou finalizar:**
    (a) Se o usuário estiver satisfeito com o resultado, pode optar por salvar o arquivo, informando o diretório onde o mesmo deve ser armazenado.
    (b) Caso o usuário decida não salvar, o programa será encerrado.

A arquitetura do sistema foi projetada com componentes independentes, permitindo uma separação clara de responsabilidades. O sistema foi concebido para facilitar a interação do usuário com LLMs e automatizar a criação, validação e manipulação de arquivos de configuração do Ansible. Para isso, são adotadas práticas de orientação a objetos, com classes e métodos que representam entidades e operações do sistema, permitindo a reutilização de código e sua extensão de forma organizada.

Até o momento, foram realizados alguns testes preliminares com a solução proposta. Os resultados foram satisfatórios, com o sistema gerando Playbooks Ansible válidos e coerentes com tarefas simples descritas em linguagem natural (por exemplo, provisionamento de servidores web). No entanto, perceberam-se limitações de escalabilidade da solução em cenários onde várias iterações para refinamento do arquivo de configuração são necessárias, sugerindo a necessidade de melhorias no uso de informações contextuais e na adoção de exemplos mais refinados para os modelos.

4.3. Próximos Passos

Para que os objetivos estabelecidos na Seção 4.1 sejam alcançados, os próximos passos deste trabalho serão organizados em quatro etapas:

1. **Revisão de Literatura:** Esta fase consistirá em um estudo mais aprofundado sobre soluções e abordagens existentes para a geração de arquivos de configuração de soluções de IaC (em especial, Ansible) usando LLMs. Pretende-se realizar uma busca tanto em fontes acadêmicas quanto em fontes de literatura cinzenta. Um subproduto desta fase será a definição de tarefas de automação que servirão como base para os testes de desempenho dos LLMs.
2. **Análise de Sensibilidade:** Esta fase consistirá em uma avaliação de LLMs com variações de parâmetros para identificar a influência de tais parâmetros na geração de arquivos de configuração de infraestrutura, bem como a identificação de combinações otimizadas de parâmetros para os cenários de teste definidos.

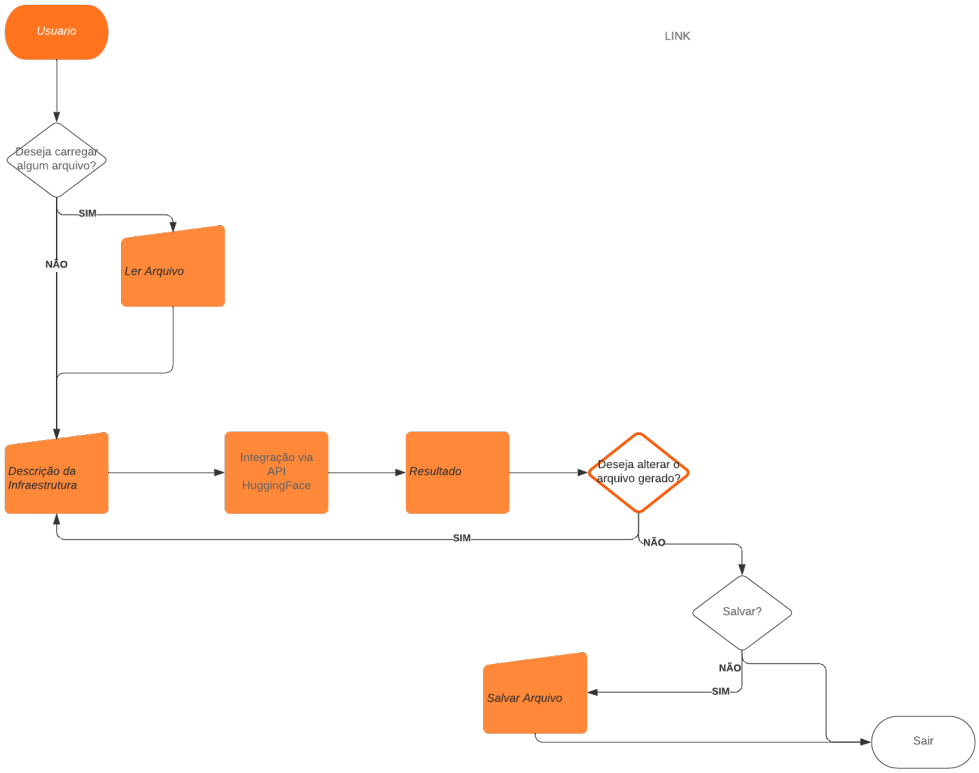

Figura 2. Fluxograma da solução proposta.

3. **Engenharia de Prompt:** Esta fase consistirá na criação de prompts para a interação com os LLMs, visando, por exemplo, mitigar eventos como a geração de códigos inválidos.
4. **Avaliação e Integração:** Esta fase consistirá na avaliação dos resultados obtidos nas fases anteriores, com o intuito de identificar modelos com maior aderência aos cenários de teste e, consequentemente, maior potencial de aplicação prática. Com base nos resultados obtidos, os melhores modelos serão integrados a um sistema que permita a interação com o usuário final.
5. **Relatório:** Esta fase consistirá na redação do relatório final do trabalho. Esta fase potencialmente compreenderá também a submissão de um artigo apresentando a solução para um evento científico.

É importante observar que mais de uma fase poderá ser executada de forma simultânea, dependendo da complexidade e relação entre as atividades. A distribuição temporal das atividades futuras é apresentada na Tabela 1.

Tabela 1. Cronograma das atividades para o restante da pesquisa.

Atividade	**Mês**						
	Jan/2025	**Fev/2025**	**Mar/2025**	**Abr/2025**	**Mai/2025**	**Jun/2025**	**Jul/2025**
Revisão de Literatura	x	x					
Análise de Sensibilidade		x	x				
Engenharia de Prompt				x			
Avaliação e Integração				x	x	x	
Relatório				x	x	x	x

5. Considerações Finais

A automação e a gestão eficiente de infraestruturas são pilares fundamentais no cenário atual de tecnologia, onde a escalabilidade e a agilidade são indispensáveis. A prática de IaC tornou-se uma solução robusta para padronizar, replicar e gerenciar ambientes complexos. No entanto, a criação manual de arquivos de configuração, especialmente em formatos como YAML, apresenta desafios significativos, como erros humanos, repetitividade e dificuldades para escalar processos.

Neste contexto, este trabalho propõe o uso de LLMs para automatizar a geração de Playbooks do Ansible. Foram realizados testes preliminares com um protótipo escrito em Python, e os resultados obtidos indicam o potencial da solução no processo de simplificação do uso de ferramentas de IaC. Como próximos passos, pretende-se (i) estudar mais profundamente as soluções existentes no âmbito de IaC e LLMs, (ii) avaliar diferentes arquiteturas e tamanhos de LLMs para identificar o impacto desses fatores na geração de Playbooks, (iii) criar *prompts* para interação com os LLMs, e (iv) integrar os modelos mais promissores a um sistema que permita a interação com o usuário final.

Referências

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Alammar, J. and Grootendorst, M. (2024). *Hands-on large language models: Language understanding and generation*. O'Reilly Media, Sebastopol, CA.

Bass, L. (2017). The software architect and devops. *IEEE Software*, 35(1):8–10.

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A., et al. (2018). A manifesto for future generation cloud computing: Research directions for the next decade. *ACM Computing Surveys*, 51(5):1–38.

Chowdhary, K. and Chowdhary, K. (2020). Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649.

Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). Devops. *IEEE software*, 33(3):94–100.

Guerriero, M., Garriga, M., Tamburri, D. A., and Palomba, F. (2019). Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *International conference on software maintenance and evolution*, pages 580–589. IEEE.

Kwon, S., Lee, S., Kim, T., Ryu, D., and Baik, J. (2023). Exploring llm-based automated repairing of ansible script in edge-cloud infrastructures. *Journal of Web Engineering*, 22(6):889–912.

Low, E., Cheh, C., and Chen, B. (2024). Repairing infrastructure-as-code using large language models. In *2024 IEEE Secure Development Conference (SecDev)*, pages 20–27. IEEE.

Morris, K. (2016). *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, Inc., 1st edition.

Rahman, A., Mahdavi-Hezaveh, R., and Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77.

Raiaan, M. A. K., Mukta, M. S. H., Fatema, K., Fahad, N. M., Sakib, S., Mim, M. M. J., Ahmad, J., Ali, M. E., and Azam, S. (2024). A review on large language models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access*.

Sahoo, P., Pujar, S., Nalawade, G., Genhardt, R., Mandel, L., and Buratti, L. (2024a). Ansible lightspeed: A code generation service for it automation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, pages 2148–2158.

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., and Chadha, A. (2024b). A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*.

Sarda, K., Namrud, Z., Litoiu, M., Shwartz, L., and Watts, I. (2024). Leveraging large language models for the auto-remediation of microservice applications: An experimental study. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*, pages 358–369.

Satyanarayanan, M., Klas, G., Silva, M., and Mangiante, S. (2019). The seminal role of edge-native applications. In *International Conference on Edge Computing*, pages 33–40. IEEE.

Shahin, M., Babar, M. A., and Zhu, L. (2017). Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

Team, G., Anil, R., Borgeaud, S., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., Millican, K., et al. (2023). Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Wang, Y., Yao, Q., Kwok, J. T., and Ni, L. M. (2020). Generalizing from a few examples: A survey on few-shot learning. *ACM Computing Surveys*, 53(3):1–34.

Zhu, L., Bass, L., and Champlin-Scharff, G. (2016). Devops and its practices. *IEEE Software*, 33(3):32–34.