Infraestrutura como Código: A prova de conceito para o melhor resultado em uma pipeline de desenvolvimento de software

Denis B. Citadin[1] **e Arthur F. Lorenzon**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Caixa Postal 15.064 – 91.501-970 – Porto Alegre – RS – Brazil

deniscitadin@gmail.com, aflorenzon@inf.ufrgs.br

Resumo. *O objetivo deste trabalho é analisar as ferramentas de infraestrutura como código, apresentando suas funcionalidades e comparando-as entre si. Não somente, como também, trazer novas perspectivas através de um olhar crítico, definições e práticas que possam contribuir com implantações de softwares mais atraentes, ágeis, simples e robustas.*

1. Introdução

A popularização da computação em nuvem tem revolucionado a abordagem de gerenciamento dos recursos computacionais, simplificando a implementação de infraestruturas em larga escala na Internet. Contudo, com o crescimento acelerado desse setor, surgiram desafios significativos relacionados ao gerenciamento desses ambientes complexos, devido ao aumento no número de equipes de desenvolvimento de softwares. Neste sentido, a prática de Infraestrutura como código (*Infrastructure as Code* - IaC) emergiu como uma solução eficaz para esses desafios, simplificando a gestão de infraestruturas de nuvem ao permitir que administradores e desenvolvedores definam e controlem recursos de computação por meio de arquivos de configuração. Baseado nessas transformações da indústria, o surgimento de diversas ferramentas diferentes e a competição da indústria entre elas, foi algo natural e esperado, devido aos avanços tecnológicos. Assim, neste trabalho, examinaremos as diversas ferramentas e os seus respectivos *frameworks*, *addons* e *plugins*, que surgiram em prol de cobrir falhas das mesmas, com a finalidade de indicar a melhor sugestão para a construção de *pipelines* inteligentes de infraestrutura como código.

2. Referencial Teórico

O desenvolvimento de arquivos de configuração em prol da transformação de grandes infraestruturas não se resume apenas a documentar infraestruturas em forma de código. Se esse fosse o único objetivo, poderíamos utilizar linguagens de programação convencionais para fazer chamadas de API diretamente à infraestrutura alvo. No entanto, o princípio fundamental é o gerenciamento completo do estado da infraestrutura, estabelecendo uma conexão entre identificadores únicos e os objetos declarados nos arquivos de configuração. Esse aspecto permite que o planejamento, implementação e até mesmo a destruição de blocos de recursos sejam simplificados por meio desse controle. Além disso, há outras vantagens significativas, como a possibilidade do compartilhamento universal com as equipes de desenvolvimento. Com base na abordagem de infraestrutura

como código, a facilidade de manter uma estrutura não redundante (princípio DRY – *Don't repeat yourself*) [Morris 2020] se torna mais viável e atraente, contribuindo para a simplificação no projeto de arquiteturas, tornando o processo de desenvolvimento mais eficiente e coeso.

2.1. Principais Ferramentas

Terraform [Brikman 2022] é considerada a principal ferramenta e com o maior número de *softwares* agregadores para serem adicionados a um *pipeline*. Sua popularidade vem da ampla compatibilidade com diversas tecnologias e do suporte de uma extensa comunidade. Dificilmente, um engenheiro de software, encontrará alguma tecnologia de mercado que não possa ser produzida com Terraform, seja na nuvem ou no *on-premisses*, assim como *switches*, roteadores de rede e até mesmo computação embarcada. Além do Terraform, existem diversas outras ferramentas populares no mercado, como SaltStack, Chef, Puppet e Pulumi, cada uma contribuindo com suas peculiaridades para a automação e gestão de infraestruturas. Adicionalmente, há ferramentas proprietárias oferecidas pelos provedores de serviços em nuvem, tais como o Cloudformation da AWS, o Azure Resource Manager do Azure e Google Cloud Deployment Manager do Google Cloud. A principal diferença dessas ferramentas em relação ao Terraform é que elas são restritas à criação e gestão de recursos dentro de seus respectivos ambientes de nuvem, limitando sua aplicabilidade a um único provedor.

3. Metodologia e Resultados Esperados

A proposta deste trabalho de mestrado consiste em demonstrar os melhores cenários de desenvolvimento de infraestrutura como código em uma *pipeline*, utilizando-se das ferramentas com maior compatibilidade entre os grandes *players* de mercado, o menor esforço de convergência positiva em implantações e manutenibilidade de código mais simples, para cada cenário de teste. Para utilização dos cenários de testes citados, isso inclui, a utilização de ambientes de desenvolvimento construídos com *Kubernetes* ou totalmente *Serverless*, ambientes em nuvem privada, ambientes *on-premisses*, desenvolvimento e controle de bancos de dados por infraestrutura como código e demais outros recursos que possam ser manipulados por ferramentas de *IaC*. Este estudo pretende abordar os melhores resultados em implantações de arquiteturas comumente utilizadas no mercado de desenvolvimento, priorizando segurança, viabilidade financeira e padronizações de implantações de arquiteturas modernas [Beyer et al. 2016], utilizando *IaC* como base. Além de compará-los entre si, demonstrando vantagens, desvantagens e enumerando as melhores combinações de ferramentas para os ambientes citados anteriormente. Entre as métricas que serão utilizadas, estão: as melhores compatibilidades da ferramenta de *IaC* com a tecnologia suportada na pipeline, a velocidade de implantação, a facilidade de escrita de código, bem como, a maior compatibilidade de desenvolvimento por código.

References

Beyer, B., Jones, C., Petoff, J., and Murphy, N. R. (2016). *Site reliability engineering: How Google runs production systems*. " O'Reilly Media, Inc.".

Brikman, Y. (2022). *Terraform: Up and Running*. " O'Reilly Media, Inc.".

Morris, K. (2020). *Infrastructure as code*. O'Reilly Media.