SOUZAEAD
Revista Acadêmica Digital

ISSN 2595-5934

PERIODICIDADE
MENSAL

JUN 2026 EDIÇÃO Nº98

IDIOMAS
PORTUGUÊS E INGLÊS

QUALIS B3

CAPES

www.souzaeadrevistaacademica.com.br

SOUZAEAD
Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

DESAFIOS NA ALTA DISPONIBILIDADE E ORQUESTRAÇÃO DE APLICAÇÕES EM INFRAESTRUTURAS DE NUVEM: UMA PROPOSTA BASEADA EM KUBERNETES E TERRAFORM

CHALLENGES IN HIGH AVAILABILITY AND APPLICATION ORCHESTRATION IN CLOUD INFRASTRUCTURES: A PROPOSAL BASED ON KUBERNETES AND TERRAFORM

RESENDE, Igor Peli[1]

RESUMO

A crescente complexidade dos sistemas distribuídos tem ampliado os desafios relacionados à padronização, manutenção e confiabilidade da infraestrutura. Nesse contexto, este trabalho apresenta uma proposta de arquitetura baseada em Infraestrutura como Código, utilizando Terraform e Kubernetes, com foco na automação e na reprodutibilidade de ambientes computacionais. A abordagem foi avaliada por meio de uma prova de conceito, na qual a infraestrutura e os serviços foram definidos de forma declarativa e orquestrados em um cluster Kubernetes. Os resultados obtidos indicam que a integração entre as ferramentas contribui para a redução de configurações manuais, melhora a organização da infraestrutura e facilita sua evolução. Apesar das limitações inerentes ao ambiente de testes, a proposta mostrou-se consistente como base para arquiteturas mais escaláveis.

Palavras-chave: Infraestrutura como Código. Kubernetes. Terraform. Computação em Nuvem. Alta Disponibilidade.

ABSTRACT

The increasing complexity of distributed systems has intensified challenges related to infrastructure standardization, maintenance, and reliability. In this context, this paper presents an architecture proposal based on Infrastructure as Code using Terraform and Kubernetes, aiming to automate and improve the reproducibility of computational environments. The approach was evaluated through a proof of concept, in which infrastructure and services were defined declaratively and orchestrated within a Kubernetes cluster. The results indicate that the integration of these tools reduces manual configuration efforts, improves infrastructure organization, and supports its evolution. Despite the limitations of the experimental environment, the proposal proved to be a consistent foundation for more scalable architectures.

[1] Graduação em Ciência da Computação pela Universidade de Vila Velha (UVV), Brasil. E-mail: igor.peli.resende@gmail.com

SOUZAEAD Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

Keywords: Infrastructure as Code. Kubernetes. Terraform. Cloud Computing. High Availability.

1 INTRODUÇÃO

1.1 Arquiteturas de sistemas distribuídos

Tradicionalmente, sistemas distribuídos eram desenvolvidos a partir de arquiteturas monolíticas, caracterizadas por forte acoplamento entre seus componentes. Esse modelo dificulta o crescimento do sistema, bem como sua manutenção e escalabilidade, uma vez que alterações ou aumentos de carga tendem a exigir intervenções em toda a aplicação (BASS et al., 2013).

Com o avanço tecnológico, a maior disseminação de ferramentas auxiliares e o aumento do conhecimento técnico levaram à adoção de arquiteturas mais desacopladas, como arquiteturas em camadas e baseadas em microsserviços. Esse modelo traz maior flexibilidade para evoluir o sistema, permitindo isolar responsabilidades e reduzir impactos em mudanças (NEWMAN, 2021).

Arquiteturas desacopladas possibilitam uma manutenção mais simples e eficiente, pois as intervenções podem ser realizadas em pontos específicos do sistema, sem a necessidade de afetar todo o conjunto. Além disso, quando apenas uma parte da aplicação precisa lidar com maior carga, é possível escalar somente esse componente, evitando desperdício de recursos computacionais (FOWLER, LEWIS, 2014).

1.2 Desafios na gestão de infraestrutura

Apesar da ampla disponibilidade de tecnologias e ferramentas modernas, ainda persiste, em muitos ambientes, a ocorrência de problemas recorrentes, tais como configurações manuais, ausência de padronização e dificuldades em garantir níveis adequados de disponibilidade dos serviços (LAUDON, LAUDON, 2017). Esses fatores aumentam a complexidade operacional e elevam custos de manutenção.

Nesse contexto, práticas associadas à automação e à padronização de ambientes tornam-se fundamentais. No entanto, mesmo com a adoção de abordagens DevOps, desafios ligados à reprodutibilidade, governança e confiabilidade de

SOUZAEAD Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

ambientes ainda são observados (KIM et al., 2016), reforçando a necessidade de soluções mais estruturadas e escaláveis.

1.3 Ferramentas para automação e padronização

É nesse ponto que entram ferramentas como o Kubernetes e o Terraform. Kubernetes é uma plataforma open-source para gerenciar workloads e serviços em contêineres, baseada em configuração declarativa e automação, oferecendo mecanismos que apoiam alta disponibilidade, como autorrecuperarão (self-healing), balanceamento e escalabilidade (KUBERNETES AUTHORS, 2024).

Complementarmente, o Terraform adota o conceito de Infraestrutura como Código (IaC), permitindo definir e provisionar infraestrutura por meio de arquivos declarativos, reduzindo a dependência de configurações manuais e contribuindo para consistência e repetibilidade entre ambientes (HASHICORP, 2024).

1.4 Objetivos

Diante desse contexto, o objetivo geral deste artigo é propor e analisar uma arquitetura baseada em Infraestrutura como Código, utilizando as ferramentas Terraform e Kubernetes, com foco na automação, padronização e reprodutibilidade de ambientes computacionais.

Como objetivos específicos, busca-se:

- Apresentar os desafios recorrentes associados à configuração manual e à gestão tradicional de infraestrutura;
- Demonstrar, por meio de uma prova de conceito, a integração entre Terraform e Kubernetes na definição e execução de ambientes;
- Analisar, de forma qualitativa, os benefícios e limitações da arquitetura proposta no que se refere à organização, manutenção e evolução da infraestrutura.

SOUZAEAD
Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

2 DESENVOLVIMENTO

2.1 Contexto e desafios em sistemas de informação

2.1.1 Complexidade das arquiteturas modernas

Com a evolução das arquiteturas de software, aumentou significativamente a complexidade envolvida na estruturação e implementação de sistemas de informação. Diferentemente dos modelos tradicionais, nos quais a aplicação era desenvolvida como um único bloco de código, as arquiteturas atuais exigem uma visão de longo prazo, considerando aspectos como modularização, comunicação entre serviços e segurança individual de cada componente, o que amplia os desafios de desenvolvimento e operação (FOWLER; LEWIS, 2014).

2.1.2 Dependência de configurações manuais

Em muitos ambientes, ainda há forte dependência de intervenções humanas na configuração e manutenção da infraestrutura. Esse cenário contribui para a ocorrência de erros operacionais e períodos de indisponibilidade, especialmente em contextos que exigem conformidade com normas e políticas de compliance. Além disso, a existência de múltiplos ambientes, como desenvolvimento, homologação e produção, tende a aumentar a complexidade e a probabilidade de inconsistências entre configurações (KIM et al., 2016).

2.1.3 Centralização do conhecimento

Outro desafio recorrente está relacionado à concentração de conhecimento em um número restrito de profissionais. Quando apenas um pequeno grupo domina o funcionamento do sistema e de sua infraestrutura, surgem dificuldades na integração de novos membros à equipe, além de riscos associados à dependência excessiva de pessoas específicas. Esse cenário impacta diretamente a continuidade operacional e o processo de capacitação (FORSGREN et al., 2018).

2.1.4 Governança e rastreabilidade

Adicionalmente, a ausência de documentação adequada e de práticas de governança dificulta o controle de versões, auditorias e rastreabilidade das mudanças

realizadas no ambiente. Configurações executadas de forma manual, muitas vezes, não são registradas, fazendo com que o conhecimento sobre o estado do sistema fique restrito a quem realizou a intervenção. Esses fatores comprometem a confiabilidade dos sistemas e limitam sua manutenção, customização, escalabilidade e evolução, resultando em ambientes pouco padronizados e de difícil gestão (BURNS, 2018).

2.2 Proposta de arquitetura baseada em infraestrutura como código

2.2.1 Papel do Terraform na arquitetura

O Terraform tem como papel gerenciar, versionar e administrar a arquitetura de forma declarativa, por meio do conceito de Infraestrutura como Código (IaC), permitindo que toda a infraestrutura seja definida por arquivos de configuração (MORRIS, 2016; HASHICORP, 2024).

Por meio do código, toda a infraestrutura passa a ficar disponível para análise, facilitando a compreensão do que foi criado e como os recursos estão organizados. Essa abordagem permite modificar ou remover componentes de maneira mais rápida e segura, garantindo que alterações sejam realizadas de forma controlada, sem deixar resíduos de configurações anteriores.

A linguagem declarativa adotada pelo Terraform funciona a partir da comparação entre o estado desejado, definido nos arquivos de configuração, e o estado atual da infraestrutura. Esse estado declarado atua como uma fonte da verdade, a partir da qual o Terraform identifica quais mudanças precisam ser aplicadas para convergir o ambiente (BURNS, 2018).

Por esse motivo, é importante que o arquivo de estado seja tratado de forma segura, uma vez que ele representa o registro fiel da infraestrutura existente. Esses mecanismos contribuem diretamente para a rastreabilidade das mudanças e para a governança do ambiente (MORRIS, 2016).

SOUZAEAD Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

2.2.2 Papel do Kubernetes na execução dos serviços

O Kubernetes atua como um orquestrador dos recursos computacionais da aplicação, permitindo definir regras e comportamentos esperados para os serviços em execução (BURNS et al., 2019).

A partir dessas definições, a plataforma passa a gerenciar automaticamente situações como falhas, reinicializações e atualizações, reduzindo a necessidade de intervenções constantes por parte das equipes e evitando a dependência de monitoramento manual contínuo, conforme descrito na documentação oficial da plataforma (KUBERNETES AUTHORS, 2024).

Na arquitetura proposta, o Kubernetes organiza os recursos por meio de namespaces, que em ambientes profissionais costumam representar diferentes contextos, como desenvolvimento, homologação e produção (KUBERNETES AUTHORS, 2024).

Os Deployments são responsáveis por gerenciar o ciclo de vida das aplicações, garantindo que o número desejado de instâncias esteja sempre em execução e possibilitando recuperação rápida em caso de falhas. Já os Services permitem a comunicação entre os componentes internos da aplicação e a exposição controlada para acesso externo, contribuindo para a padronização e previsibilidade da arquitetura (BURNS et al., 2019).

2.2.3 Integração entre Terraform e Kubernetes

A integração entre Terraform e Kubernetes é um ponto central da arquitetura proposta. Por meio do provedor Kubernetes, o Terraform passa a se comunicar diretamente com o cluster, permitindo que os recursos do ambiente sejam definidos e gerenciados por código (WEAVEWORKS, 2021).

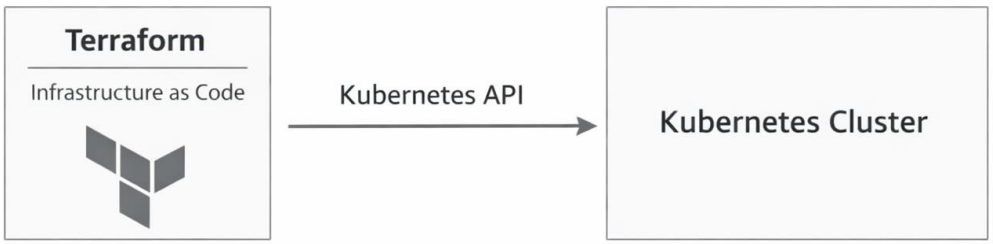
Essa integração possibilita a criação e a manutenção da infraestrutura do Kubernetes de forma padronizada e reproduzível, característica essencial em ambientes modernos de computação em nuvem (MORRIS, 2016).

Conforme ilustrado na Figura 1, o Terraform atua como ponto central de definição da infraestrutura, aplicando configurações por meio da API do Kubernetes.

SOUZAEAD
Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

Com essa abordagem, o Terraform é responsável por declarar o estado desejado dos recursos, enquanto o Kubernetes atua na aplicação e na manutenção desse estado em tempo de execução. Essa separação de responsabilidades permite explorar o melhor das duas ferramentas, combinando a rastreabilidade e o versionamento do IaC com os mecanismos nativos de orquestração e automação do Kubernetes.

Figura 1 - Visão geral do uso do Terraform como Infraestrutura como Código para criação e gerenciamento de recursos em um cluster Kubernetes.



Fonte: Próprio autor.

2.3 Metodologia

Este estudo caracteriza-se como uma pesquisa aplicada, de natureza qualitativa e exploratória, baseada na implementação de uma prova de conceito. O ambiente foi construído a partir da integração entre Terraform e Kubernetes, com o objetivo de avaliar a viabilidade dessa abordagem na definição e no gerenciamento de infraestrutura por meio de código.

A solução foi implementada em um cluster Kubernetes de nó único, adequado para fins acadêmicos e de validação conceitual. A análise concentrou-se na observação do comportamento da infraestrutura implantada, considerando aspectos relacionados à organização dos recursos, versionamento da configuração e mecanismos de orquestração oferecidos pelo Kubernetes.

SOUZAEAD
Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

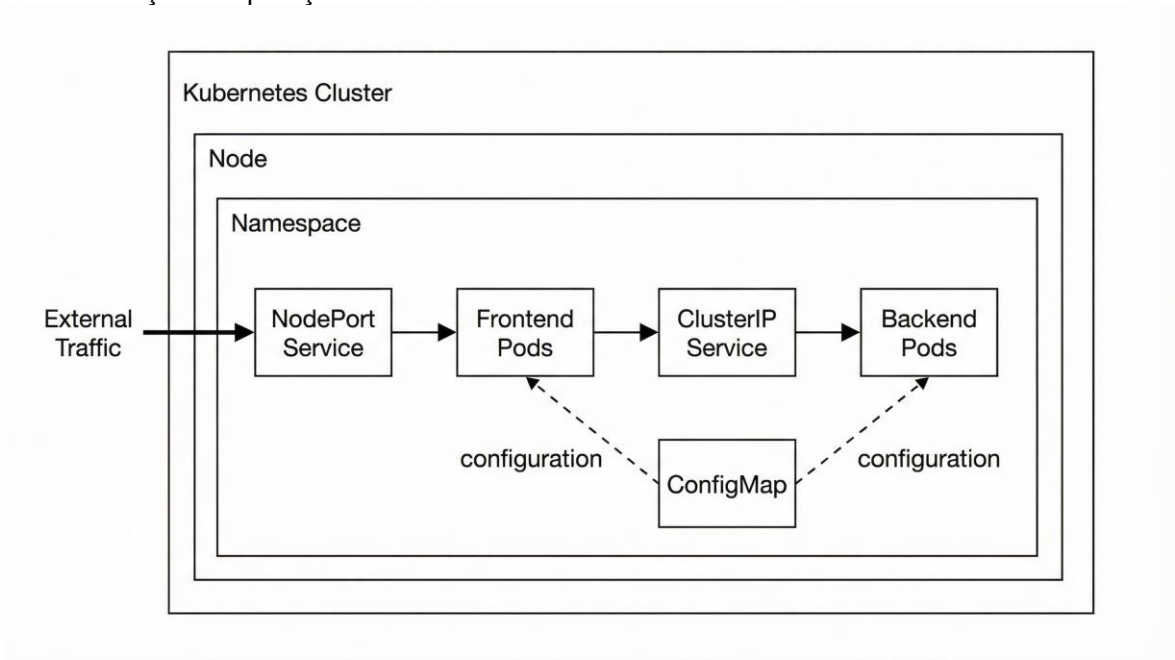
2.4 Arquitetura da Solução

2.4.1 Visão geral da arquitetura

A solução é apresentada como uma prova de conceito, na qual todos os componentes são executados em um único nó. A organização dos recursos e o fluxo de comunicação da solução são apresentados na Figura 2.

O cluster Kubernetes foi implementado de forma local na máquina de desenvolvimento, servindo como ambiente de validação da proposta arquitetural apresentada.

Figura 2 - Arquitetura interna do cluster Kubernetes, destacando a organização dos recursos e o fluxo de comunicação da aplicação.



Fonte: Próprio autor.

2.4.2 Organização dos recursos no namespace

Conforme ilustrado na figura, todos os recursos da aplicação estão organizados dentro de um único namespace. Esse recurso atua como um mecanismo lógico de isolamento, permitindo a separação e a organização dos componentes dentro do cluster.

Dentro do namespace, os recursos são definidos de forma explícita e padronizada. O ConfigMap é utilizado para o armazenamento de parâmetros de

configuração no formato chave-valor, sendo associado aos pods de frontend e backend. Esse recurso não participa diretamente do fluxo de comunicação, tendo como finalidade a separação entre configuração e código da aplicação.

2.4.3 Fluxo de comunicação

O fluxo de comunicação do cluster segue um caminho bem definido. Todo o tráfego externo ao cluster é direcionado inicialmente para um Service do tipo NodePort, que atua como ponto de entrada da aplicação. Esse serviço é responsável por receber as requisições externas e distribuí-las para os pods do frontend.

Após o processamento inicial realizado pelos pods de frontend, as requisições são encaminhadas para um Service do tipo ClusterIP. Esse serviço é utilizado exclusivamente para comunicação interna no cluster, abstraindo o acesso aos pods de backend e realizando o balanceamento do tráfego entre eles. Dessa forma, a comunicação entre frontend e backend ocorre de maneira padronizada e controlada (KUBERNETES AUTHORS, 2024).

2.4.4 Escopo e limitações do ambiente

A arquitetura apresentada foi desenvolvida com foco em uma demonstração conceitual, sendo executada em um cluster Kubernetes com apenas um único nó. Essa configuração impõe limitações em relação à alta disponibilidade em nível de infraestrutura, uma vez que a falha do nó compromete todo o ambiente. Em cenários reais de produção, recomenda-se a utilização de múltiplos nós, com o objetivo de aumentar a disponibilidade dos serviços.

Ainda assim, o modelo adotado permite que a solução seja reproduzida em ambientes mais complexos, mantendo o mesmo modelo de organização dos recursos e o fluxo de comunicação apresentado, sem a necessidade de alterações conceituais na arquitetura.

SOUZAEAD Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

2.5 Resultados e discussão

2.5.1 Padronização e reprodutibilidade da infraestrutura

A implementação da prova de conceito possibilitou analisar, na prática, a utilização conjunta do Terraform e do Kubernetes no gerenciamento de uma infraestrutura baseada em código. Um dos principais resultados observados foi a padronização do ambiente, uma vez que todos os recursos passaram a ser definidos de forma declarativa, reduzindo a necessidade de configurações manuais e diminuindo inconsistências entre diferentes implantações.

Esse resultado está de acordo com a literatura, que aponta a Infraestrutura como Código como uma prática essencial para aumentar a confiabilidade e reduzir erros operacionais em ambientes de nuvem (MORRIS, 2016; KIM et al., 2016).

O uso do Terraform como ferramenta de Infraestrutura como Código mostrou-se essencial para garantir a reprodutibilidade da solução. A partir dos arquivos de configuração, a infraestrutura pode ser recriada de maneira previsível, permitindo o versionamento das mudanças e facilitando a compreensão do estado do ambiente ao longo do tempo.

2.5.2 Orquestração e estabilidade dos serviços

Em relação à execução dos serviços, o Kubernetes apresentou bom desempenho na orquestração dos componentes da aplicação. Funcionalidades como reinicialização automática de pods e balanceamento interno de requisições contribuíram para maior estabilidade operacional, mesmo em um cluster de nó único.

Esse comportamento também é descrito na literatura, que destaca o Kubernetes como uma solução eficiente para aumentar a resiliência e a disponibilidade de aplicações distribuídas (BURNS et al., 2019).

2.5.3 Limitações e considerações sobre escalabilidade

Apesar dos resultados positivos, algumas limitações devem ser consideradas. A execução da solução em um ambiente composto por apenas um nó limita a

avaliação prática de alta disponibilidade em nível de infraestrutura, uma vez que a indisponibilidade do nó compromete todo o sistema.

Estudos sobre arquiteturas distribuídas indicam que a alta disponibilidade depende da utilização de múltiplos nós, evitando pontos únicos de falha e aumentando a resiliência do sistema (NEWMAN, 2021).

2.5.4 Síntese dos resultados

De modo geral, os resultados obtidos indicam que a arquitetura proposta atende aos objetivos definidos, evidenciando que a integração entre Terraform e Kubernetes constitui uma abordagem adequada para a automação, padronização e evolução de ambientes computacionais.

Esse resultado está alinhado com estudos que destacam a importância da combinação entre práticas de DevOps, Infraestrutura como Código e orquestração de contêineres para lidar com a complexidade dos sistemas modernos (FORSGREN et al., 2018).

3 CONSIDERAÇÕES FINAIS

3.1 Limitações da proposta

Embora a solução tenha sido implementada em um cluster Kubernetes de nó único, os resultados obtidos indicam que o modelo arquitetural proposto é consistente e pode ser expandido para cenários mais complexos. As limitações identificadas estão relacionadas principalmente ao escopo do ambiente de testes e não comprometem a aplicabilidade da abordagem em contextos reais de produção.

3.2 Trabalhos futuros

Como trabalhos futuros, sugere-se a evolução da arquitetura para clusters Kubernetes multinó, a integração com pipelines de integração e entrega contínua (CI/CD) e a realização de avaliações quantitativas de desempenho e disponibilidade. Essas extensões possibilitariam uma análise mais aprofundada dos benefícios da abordagem em ambientes produtivos.

SOUZAEAD Revista Acadêmica Digital
ISSN 2595-5934
JUN 2026 EDIÇÃO Nº98

3.3 Disponibilização do código

Todo o código desenvolvido neste trabalho encontra-se disponível em um repositório público no GitHub, podendo ser acessado em: https://github.com/IgorPeli/Article_Terraform-Kubernets.

3.4 Considerações finais

Conclui-se, portanto, que a utilização conjunta do Terraform e do Kubernetes representa uma alternativa viável e eficiente para a automação e o gerenciamento de infraestrutura, sendo aplicável tanto em contextos acadêmicos quanto profissionais.

REFERÊNCIAS

BASS, L.; CLEMENTS, P.; KAZMAN, R. Software Architecture in Practice. 3. ed. Addison-Wesley, 2013.

BURNS, B. Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services. O'Reilly Media, 2018.

FORSGREN, N.; HUMBLE, J.; KIM, G. Accelerate: The Science of Lean Software and DevOps. IT Revolution Press, 2018.

FOWLER, M.; LEWIS, J. Microservices: a definition of this new architectural term. Disponível em: https://martinfowler.com/articles/microservices.html. Acesso em: 15 jan. 2026.

MORRIS, K. Infrastructure as Code. O'Reilly Media, 2016.

KIM, G.; HUMBLE, J.; DEBOIS, P.; WILLIS, J. The DevOps Handbook. IT Revolution Press, 2016.

KUBERNETES AUTHORS. Concepts Overview. Disponível em: https://kubernetes.io/docs/concepts/overview/. Acesso em: 15 jan. 2026.

LAUDON, K. C.; LAUDON, J. P. Sistemas de Informação Gerenciais. Pearson Education, 2017.

NEWMAN, S. Building Microservices: Designing Fine-Grained Systems. 2. ed. O'Reilly Media, 2021.

ZHANG, Q.; CHEN, M. Cloud computing: state-of-the-art and research challenges. Journal of Internet Services and Applications, 2020.

WEAVEWORKS. GitOps: Operations by Pull Request. 2021. Disponível em: https://medium.com/weaveworks/gitops-operations-by-pull-request-14e8b659b058

HASHICORP. Terraform documentation. 2024. Disponível em: https://developer.hashicorp.com/terraform/docs. Acesso em: 15 jan. 2026.