Associação Propagadora Esdeva
Centro Universitário Academia – UniAcademia
Curso de Bacharelado em Sistemas de Informação
Trabalho de Conclusão de Curso – Artigo

DESENVOLVIMENTO DE TEMPLATES TERRAFORM PARA GITOPS: PROVISIONAMENTO DE MULTIPLATAFORMAS NA NUVEM COM CONTROLE DE VERSÃO E INTEGRAÇÃO CONTÍNUA

João Pedro Guedes Presto[1]
Centro Universitário Academia - UniAcademia, Juiz de Fora, MG
Romualdo Monteiro de Resende Costa[2]
Centro Universitário Academia - UniAcademia, Juiz de Fora, MG

Linha de Pesquisa: Redes de Computadores e Sistemas Distribuídos

RESUMO

Devido à evolução contínua e à adoção generalizada de recursos em nuvem, torna-se imperativo automatizar os processos de provisionamento e versionamento de infraestrutura. Com o objetivo de simplificar o provisionamento de recursos na nuvem, este trabalho propõe o desenvolvimento de Templates Terraform. Esses modelos foram categorizados e criados para duas das principais plataformas de nuvem: *Microsoft Azure* e *Amazon Web Services*, conforme indicado por pesquisas de mercado. Para aumentar a autonomia de provisionamento, foram estabelecidos fluxos de trabalho por meio do GitHub Actions, que dependem de revisão e aprovação do código antes de realizar o provisionamento. Todo o trabalho foi devidamente versionado no repositório GitHub.

Palavras-chave: Templates. *Terraform. GitHub Actions. GitHub*.

1 INTRODUÇÃO

A computação em nuvem surgiu no MIT em 1961, introduzida por *John MacCharty* como um modelo de negócios de utilidade pública (Amaral, 2022). Desde então, a computação em nuvem tem sido amplamente adotada devido a várias

[1] Discente do Curso de Sistemas de Informação do Centro Universitário Academia – UniAcademia.
[2] Docente do Curso de Sistemas de Informação do Centro Universitário Academia - UniAcademia

justificativas, como a facilidade na aquisição de recursos, economia de escala obtida pela consolidação de servidores e benefícios relacionados à redução de custos com infraestrutura e pessoal. A nuvem oferece recursos virtualizados "inesgotáveis" acessados como serviços pelos clientes, permitindo o crescimento elástico dos sistemas (Moraes, 2021).

A fim de evitar a dependência de um único provedor de nuvem e aproveitar ao máximo os benefícios da computação em nuvem, Moraes (2021) destaca a utilização de múltiplas nuvens. Dessa forma, uma aplicação pode ser implantada em vários provedores que melhor atendam às necessidades dos usuários e aplicativos.

À medida que a infraestrutura cresce, surge a necessidade de usar um método para gerenciá-la e provisioná-la (Patni *et al.* 2020). A Infraestrutura como Código (IaC) é uma forma de gerenciar os componentes de infraestrutura, tratando-os como código-fonte, aplicando os mesmos princípios e práticas de engenharia de software (Modi, 2021). Isso garante que a infraestrutura se torne parte integrante das práticas de engenharia, seguindo os mesmos princípios e processos de desenvolvimento de aplicativos.

Dentre as principais ferramentas de IaC, destaca-se o Terraform, uma solução multiplataforma e multi-clouds para provisionar e gerenciar recursos de TI (Modi, 2021). Segundo *Terraform* (2023), a ferramenta permite definir os recursos de forma legível em arquivos de configuração que podem ser versionadas, reutilizáveis e compartilhados.

Segundo López-Viana *et al*. (2022), uma metodologia que é capaz de unificar o uso de IaC juntamente com o repositório git é o GitOps, que foi criado pela empresa *Weaveworks* em 2017. Sendo considerado um *framework* operacional usado para automatizar a infraestrutura.

Considerando toda a evolução nos conceitos relacionados à infraestrutura em nuvem, o objetivo deste trabalho é propor técnicas que visam simplificar o processo de provisionamento em nuvem. Para isto, são desenvolvidos templates para a plataforma Terraform com o objetivo de facilitar o versionamento da Infraestrutura em nuvem e automatizar o processo de provisionamento com a integração contínua. Através dos templates, os usuários poderão escolher, de forma simultânea ou seletiva, quais recursos desejam utilizar das seguintes plataformas: *Amazon Web Services (AWS) e Microsoft Azure (Azure)*. Entre as opções de escolha, serão

levantadas aquelas mais comuns e, eventualmente, novas opções poderão ser adicionadas aos templates para uso futuro pelos usuários.

O artigo ainda abordará o uso do *GitHub* para o versionamento do código e na automação da integração contínua (IC), o *GitHub Actions*[3]. Essas ferramentas, juntamente com o Terraform, fazem parte dos conceitos do *GitOps*.

A próxima seção descreve os conceitos utilizados para o desenvolvimento da proposta do trabalho (Referencial Teórico). A terceira seção discute sobre as tecnologias utilizadas no desenvolvimento. Por fim, na quarta, e última seção, são apresentadas considerações finais sobre o trabalho, bem como eventuais trabalhos futuros.

2 REFERENCIAL TEÓRICO

Nesta Seção são descritos os conceitos que fundamentam este trabalho, com destaque em Computação em Nuvem (Seção 2.1), Infraestrutura com Código (Seção 2.2), a ferramenta Terraform (Seção 2.3), a linguagem *Hashicorp Configuration Language* (*HCL*) (Seção 2.4) e, por fim, o *GitHub* (Seção 2.5).

2.1 COMPUTAÇÃO EM NUVEM

A computação em nuvem surgiu como uma solução para fornecer recursos virtuais acessíveis, abrangendo hardware, software, plataformas de desenvolvimento e serviços (Nascimento, 2020). Esse modelo permite a aquisição sob demanda, otimizando o uso por meio do pagamento pela demanda (Singh *et al.*, 2011).

Diferentemente da computação realizada com servidores locais, através do uso de serviços na nuvem, não são necessários vultosos investimentos iniciais para o estabelecimento do serviço que, muitas vezes, são realizados em um cenário de incerteza, onde não é possível precisar a real demanda pelo serviço. Nesse caso, corre-se o risco da realização de investimentos desnecessários ou, ainda, que fiquem abaixo do necessário, no caso do rápido crescimento da demanda pelo serviço oferecido. A utilização de servidores locais exige, também, todo o

[3] https://github.com/features/actions

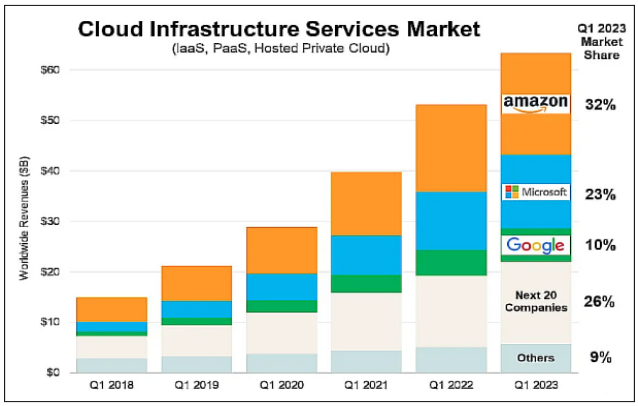
conhecimento de aquisição dos recursos, bem como da sua configuração e manutenção.

Além da flexibilidade e escalabilidade oferecidas pelo armazenamento em nuvem, os clientes também se beneficiam ao pagar apenas pelo serviço utilizado como, por exemplo, pela quantidade de dados armazenados, em um serviço dessa natureza. Também é possível, no geral, o acesso aos serviços de qualquer região geográfica com acesso à rede ou à Internet do provedor de serviços em nuvem (Singh *et al*., 2011).

Evidentemente, o custo sobre demanda pode, ao longo do tempo, ser uma estratégia mais cara do que o custo realizado com o investimento inicial de servidores locais, associados ao custo de manutenção da infraestrutura adquirida. Esse cenário, no entanto, é apenas uma das considerações que devem ser realizadas, por ocasião da decisão de adotar ou não a computação em nuvem. Outras questões que devem ser mensuradas, incluem, por exemplo, questões relacionadas à segurança dos ativos que, quando colocados na nuvem, permitem que a gestão do risco possa ser realizada através da terceirização para empresas especializadas que oferecem o serviço de infraestrutura.

Por tudo isto, é esperado uma crescente adoção da computação em nuvem como estratégia de oferecimento de serviços pelas empresas. Esta crescente é demostrada na Figura 1 (CloudZero (2023), que mostra como os gastos com infraestrutura em nuvem vêm apresentando uma evolução ao longo dos anos de 2018 até 2023. Também na Figura 1 é destacado, nos quatro primeiros meses de 2023, a participação da *Amazon*, com 32% do mercado, seguindo da *Microsoft* com 23% e, em seguida, da *Google*, com participação estimada de 10% do mercado.

Figura 1: Participação no mercado de serviços de infraestrutura em nuvem



Fonte: CloudZero (2023)

Com a crescente adoção da computação em nuvem, surgiram estratégias para torná-la ainda mais flexível e presente nas organizações. O conceito de "*multi-clouds*", também conhecido como "*interclouds*" ou "nuvem-de-nuvens", indica que a computação em nuvem não se limita a uma única nuvem (Amaral, 2022). A utilização de várias nuvens permite solucionar problemas relacionados aos padrões de uso de serviço e implantação personalizados em um aplicativo em nuvem, garantindo escalabilidade, disponibilidade de recursos e recuperação de desastres (Moraes, 2021).

A estratégia de nuvem de nuvens também aborda questões como a distribuição geográfica, baixa latência de acesso, conformidade legal e regulatória, eficiência de custos, economia de energia, interoperabilidade e redução da dependência de um único fornecedor. A utilização de várias nuvens é a solução ideal para atender às demandas de consumidores dispersos geograficamente, que necessitam de baixo tempo de resposta e, portanto, da capacidade de usar simultaneamente várias nuvens (Moraes, 2021).

Segundo a empresa de pesquisa Gartner (2022), as empresas líderes do mercado de plataformas de nuvem, a partir de junho de 2022, são a *Amazon Web Services* (*AWS*), a *Microsoft Azure* (*Azure*) e a *Google Cloud Platform* (*GCP*). Com destaque para as duas primeiras que, portanto, foram escolhidas para a realização deste trabalho.

2.2 INFRAESTRUTURA COMO CÓDIGO

De acordo com Cruz (2020), Infraestrutura como código (IaC) é uma abordagem de gerenciamento e provisionamento de configuração que utiliza arquivos de configuração escritos em uma linguagem de programação simples e declarativa, em vez de configurações manuais realizadas tanto em ambientes físicos quanto virtuais.

A proposta de utilização de uma linguagem de especificação declarativa é que, através de especificações simples, isto é, sem o conhecimento de estruturas complexas relacionadas à programação, o especialista possa descrever procedimentos de configuração que, posteriormente, poderão, inclusive, serem replicados. Outra vantagem importante é que as especificações permanecem registradas para posterior verificação, coisa que não é possível nas configurações

manuais. Dessa forma, torna-se possível, inclusive, a análise posterior da configuração realizada para eventuais correções de falhas ou aprimoramento de especificações relacionadas às configurações.

Seguindo as ideias de Brikman (2019), o IaC oferece facilidade no processo de implantação de infraestrutura, tornando-o mais rápido e eficiente. Além disso, ele torna a infraestrutura mais acessível para consulta, uma vez que é representada por meio de código. A possibilidade de controlar a versão da infraestrutura é outro ponto positivo, permitindo a rastreabilidade e o gerenciamento de mudanças. Com o IaC, é possível validar a infraestrutura antes de implementá-la, garantindo maior segurança e qualidade. Além disso, o reuso de código possibilita a criação de ambientes a partir de bases já existentes, simplificando o processo. Por fim, a automação de processos repetitivos de infraestrutura é viabilizada pelo uso do IaC.

2.3 TERRAFORM

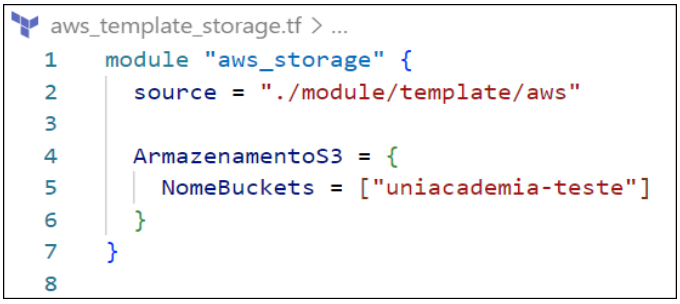
O *Terraform*[4] é uma ferramenta de Infraestrutura como Código que pode ser utilizada para descrever e provisionar recursos na nuvem. Com o uso de arquivos de configuração legíveis, o *Terraform* permite controlar a criação, atualização e destruição de diversos componentes, desde recursos de baixo nível, como computação, armazenamento e redes, até componentes de alto nível, como entradas DNS (*Domain Name Services*, usada na tradução de nome na Internet) (Terraform, 2022).

De acordo com Brikman (2019), a linguagem de configuração do *Terraform*, conhecida como *HashiCorp Configuration Language (HCL)*, utilizada para escrever o código em arquivos com extensão "*.tf*", é declarativa, ou seja, tem como objetivo descrever a infraestrutura desejada, enquanto o Terraform se encarrega de determinar como criá-la. A flexibilidade do Terraform permite a criação de infraestrutura em várias plataformas, ou "provedores", como *AWS*, *Azure*, *Google Cloud*, entre outros.

[4] www.terraform.io

A Figura 2 apresenta, como exemplo, um trecho de código utilizado no template para provisionar um *Bucket S3*[5] com o nome “uniacademia-teste”. Um *Bucket* é uma estrutura de armazenamento de dados, um container de objetos.

Figura 2: Exemplo de Utilização do template

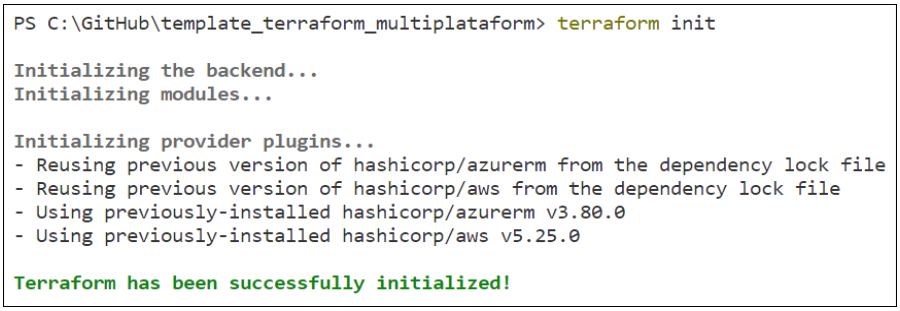


Fonte: Elaboração Própria

Modi (2021) e Terraform (2023) destacam os principais comandos do *Terraform*: *init* (a), *plan* (b) e *apply* (c). O comando *init*, uma abreviação de "inicialização", é utilizado para configurar o contexto e o ambiente do Terraform. Ao executar o *init*, o Terraform realiza o download das dependências necessárias com suas versões adequadas no diretório local de trabalho, inicializa o arquivo de estado e auxilia no gerenciamento do ambiente, utilizando módulos e provedores.

A Figura 3 apresenta, como exemplo, o resultado da execução do comando *init*. É possível notar a inicialização das dependências de cada provedor, destacando que os plugins foram baixados, a *Azure* com a versão 5.25.0 e a *AWS* com 3.80.0, também evidencia a inicialização dos módulos do template e, por fim, confirma que obteve sucesso ao iniciar.

Figura 3: Comando *terraform init*



Fonte: Elaboração Própria

[5] https://aws.amazon.com/pt/s3

A execução do comando *plan*, por sua vez, gera um plano de execução que revela as diferenças entre a configuração presente nos arquivos e o estado atual do sistema. Essa etapa é fundamental para entender quais alterações serão realizadas no ambiente de destino.

A Figura 4 ilustra o uso do comando '*plan*'. Como exemplo, considera-se o planejamento para o recurso *Bucket S3*, cujo preenchimento do template foi apresentado na Figura 2. O nome do recurso, argumento mostrado anteriormente, é visível, enquanto os demais argumentos são adquiridos somente após a criação do recurso. Além disso, no final da Figura 4, nota-se que há a adição de apenas um recurso.

Figura 4: Comando *terraform plan*

```
PS C:\GitHub\template_terraform_multiplataform> terraform plan
                                [...]
Terraform will perform the following actions:

  # module.aws_storage.module.simple_storage_service["uniacademia-teste"].aws_s3_bucket.s3 will be created
  + resource "aws_s3_bucket" "s3" {
      + acceleration_status         = (known after apply)
      + bucket                      = "uniacademia-teste"
      + bucket_domain_name          = (known after apply)
      + tags                        = {
          + "Name" = "uniacademia-teste"
        }
      + tags_all                    = {
          + "Name" = "uniacademia-teste"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Fonte: Elaboração Própria

Por fim, o comando *apply*, similar ao *plan*, segue as mesmas regras mencionadas anteriormente em relação aos parâmetros. Além disso, valores de variáveis também podem ser passados como parte do comando *apply*. Ao executar o *apply*, ocorrem alterações na infraestrutura da nuvem remota e no arquivo de estado.

A Figura 5 apresenta, como exemplo, o resultado do comando '*apply*' para o provisionamento de um *Bucket S3*. Inicia-se o processo com a reiteração do planejamento mencionado anteriormente, antes de avançar para a criação efetiva do recurso. Nota-se que a criação do *Bucket* demorou 5 segundos (5s), e o identificador (id) coincide com o nome que foi previamente inserido no template.

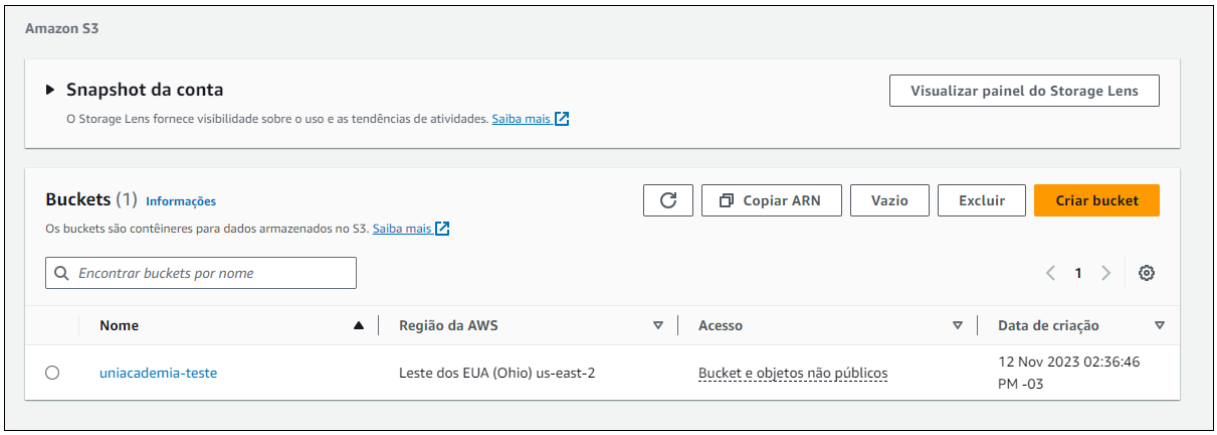
Figura 5: Provisionamento do Azure Container Storage

```
PS C:\GitHub\template_terraform_multiplataform> terraform apply
Acquiring state lock. This may take a few moments...

Terraform used the selected providers to generate the following execution plan
  + create
                    [...]
      + tags                          = {
          + "Name" = "uniacademia-teste"
        }
      + tags_all                      = {
          + "Name" = "uniacademia-teste"
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.
module.aws_storage.module.simple_storage_service["uniacademia-teste"].aws_s3_b
ucket.s3: Creating...
module.aws_storage.module.simple_storage_service["uniacademia-teste"].aws_s3_b
ucket.s3: Creation complete after 5s [id=uniacademia-teste]
Releasing state lock. This may take a few moments...

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Fonte: Elaboração Própria

Por fim, na Figura 6 pode ser observado o recurso que foi criado na *Amazon Web Service*. O *Bucket* foi provisionado corretamente com o nome “uniacademia-teste” e está localizado na região Leste dos EUA (Ohio) *us-east-2*. Este seria o resultado da execução dos comandos mencionados na ordem apresentada. Assim, para realizar esta tarefa, e outras, de forma automatizada e parametrizada, serão utilizados os templates apresentados na próxima seção.

Figura 6: Visualização do Bucket



Fonte: Elaboração Própria

2.4 *HASHICORP CONFIGURATION LANGUAGE (HCL)*

Conforme abordado por Modi (2021), o *HCL* desempenha um papel fundamental ao fornecer uma maneira clara e legível de descrever o que está sendo provisionado. Dessa forma, o template desenvolvido foi escrito em *HCL*, seguindo as diretrizes e práticas recomendadas.

Seguindo a orientação da Hashicorp (2023), a sintaxe do *HCL* foi explorada para criar blocos que representassem os diferentes componentes e recursos a serem provisionados no ambiente. Os argumentos foram utilizados para atribuir valores personalizados aos elementos do template, permitindo a configuração adequada de cada recurso.

Além disso, as expressões do *HCL* foram empregadas para representar valores literais e para referenciar e combinar outros valores necessários na definição do template. Esse aspecto foi essencial para garantir a flexibilidade e adaptabilidade do template em diferentes cenários de provisionamento. Estas expressões foram empregadas nas validações dos argumentos dos recursos, que trazem restrições a respeito de como provisionar.

A Figura 7 exemplifica validações aplicadas para a criação de bancos no *Azure SQL Database*[6]. Neste exemplo, são apresentadas duas restrições: a primeira refere-se à detecção de valores duplicados, enquanto a segunda diz respeito à limitação da quantidade de caracteres nos nomes. Estas validações asseguram que as regras pré-definidas sejam seguidas durante o processo de planejamento e criação de recursos.

[6] https://azure.microsoft.com/pt-br/products/azure-sql/database

Figura 7: Expressões usadas na validação do Azure Sql Database

```
390   validation {
391     condition = length(var.ServidorMSSql.NomeBancos) == 0 || length(
392       distinct(var.ServidorMSSql["NomeBancos"])
393       ) == length(
394       var.ServidorMSSql["NomeBancos"]
395     )
396     error_message = "A lista de bancos contém valores duplicados."
397   }
398
399   validation {
400     condition = length(var.ServidorMSSql.NomeBancos) == 0 || alltrue([
401       for name in var.ServidorMSSql.NomeBancos :
402       length(name) >= 1 && length(name) <= 128
403     ])
404     error_message = "Os nomes dos Bancos SQL devem ter entre 1 e 128 caracteres."
405   }
```

Fonte: Elaboração Própria

A Figura 7 apresenta, como exemplo, as validações para *Azure Sql Database*. A primeira validação, apresentada no intervalo entre as linhas 390 a 397, verifica se a lista de nomes de bancos fornecida (*var.ServidorMSSql.NomeBancos*) não contém valores duplicados. Isso é assegurado pela condição que compara o tamanho da lista original com o tamanho da lista após remover duplicatas. Caso a condição não seja atendida, uma mensagem de erro é gerada indicando a presença de valores duplicados na lista de bancos.

A segunda validação, apresentada entre as linhas 399 a 405, garante que os nomes dos bancos SQL estejam dentro dos limites aceitáveis, entre 1 e 128 caracteres. A condição utiliza a função "alltrue" para verificar se todas as entradas na lista de nomes de bancos atendem à condição estabelecida pelo loop "for". Caso algum nome de banco não cumpra o requisito de comprimento, a mensagem de erro correspondente é gerada, indicando que os nomes dos bancos SQL devem ter entre 1 e 128 caracteres. Essas validações garantem a integridade e a consistência dos dados durante o provisionamento do banco de dados no ambiente Azure por meio do Terraform.

2.5 *GITHUB*

O *GitHub* é uma plataforma online colaborativa que oferece ferramentas para facilitar o desenvolvimento distribuído. Ela combina recursos convencionais, como hospedagem gratuita e sistema de controle de versão, com elementos sociais, proporcionando um ambiente integrado para equipes de desenvolvimento colaborarem eficientemente. O núcleo do *GitHub* é representado pelo *Git*, um sistema de controle de versão descentralizado responsável por gerenciar e armazenar as revisões dos repositórios (Cosentino *et al.*, 2017).

De acordo com Kinsman *et al*. (2021), *GitHub Actions* é uma ferramenta orientada a eventos fornecida pela plataforma *GitHub* para automatizar fluxos de trabalho de integração e entrega continua (*CI/CD)*. As ações podem executar uma série de comandos após a execução de um evento específico. Um evento é uma atividade que dispara um fluxo de trabalho. Os fluxos de trabalhos são definidos no diretório. *github/workflows* e os arquivos usam a sintaxe *YAML*, com extensão *yml* ou *yaml*.

3. DESENVOLVIMENTO

Nesta seção é abordada a construção dos templates para o uso com a ferramenta *Terraform*. É apresentada a categorização dos recursos de computação em nuvem de cada plataforma (*AWS* e *Azure*) na Seção 3.1, bem como a organização dos Templates na Seção 3.2. Por fim, é apresentado o template de integração e entrega continua (*CI/CD)* para automatizar o processo de provisionamento utilizando o *GitHub Actions*.

3.1 RECURSOS CLOUD PARA OS TEMPLATES

No desenvolvimento deste template foram utilizadas as plataformas *Amazon Web Services* (*AWS*) e *Microsoft Azure*, em razão da importância dessas plataformas em soluções no mercado, como já mencionado. A *AWS* é conhecida por suas operações globais e por atender a uma variedade de clientes, desde startups até grandes empresas. Já a Azure oferece operações geograficamente

diversificadas e é especialmente adequada para organizações que utilizam soluções baseadas em tecnologias da *Microsoft*.

Na implementação deste template foram explorados diversos recursos da *AWS*, incluindo *Elastic Compute Cloud, Autoscaling Group, App Runner, Beanstalk Application, Simple Storage Service, RDS Database e Athena Database*. Esses recursos permitem a criação de ambientes escaláveis e o armazenamento de dados de forma eficiente. Os recursos citados anteriormente são detalhados na Tabela 1, que inclui todos os recursos que fazem parte do modelo de templates proposto neste trabalho.

Tabela 1: Recursos Amazon Web Services

Recurso	Descrição	Documentação
Elastic Compute Cloud (EC2)	É um serviço de nuvem que oferece instâncias virtuais escaláveis para hospedar aplicativos e serviços.	https://aws.amazon.com/pt/ec2/
Autoscaling Group	É um conjunto de instâncias EC2.	https://docs.aws.amazon.com/pt_br/autoscaling/ec2/userguide/auto-scaling-groups.html
App Runner	É um serviço de aplicações de contêineres.	https://aws.amazon.com/pt/apprunner/
Beanstalk Application	É um serviço para implementar e escalar aplicações e serviços da Web.	https://aws.amazon.com/pt/elasticbeanstalk/
Simple Storage Service (S3)	É um serviço de armazenamento de objetos.	https://aws.amazon.com/pt/s3/
RDS Database	É uma coleção de serviços gerenciados que facilita a configuração, operação e escalabilidade de bancos de dados na nuvem.	https://aws.amazon.com/pt/rds/
Athena Database	É um serviço de análise interativo e sem servidor criado em frameworks de código aberto, com suporte a formatos de tabela e arquivo abertos.	https://aws.amazon.com/pt/athena/

Fonte: Elaboração própria

No caso do *Azure*, foram utilizados recursos como *Azure Virtual Machine, Azure Virtual Machine Scale Sets, Azure App Service, Azure Functions, Azure SQL Database, Azure Database for MariaDB, Azure Database for MySQL, Azure Database for PostgreSQL e Azure Container Storage*. Esses recursos oferecem flexibilidade na configuração de máquinas virtuais, o suporte a diversos tipos de bancos de dados e a possibilidade de hospedar aplicativos da Web. Os recursos anteriormente citados são apresentados com detalhes na Tabela 2.

Tabela 2: Recursos Microsoft Azure

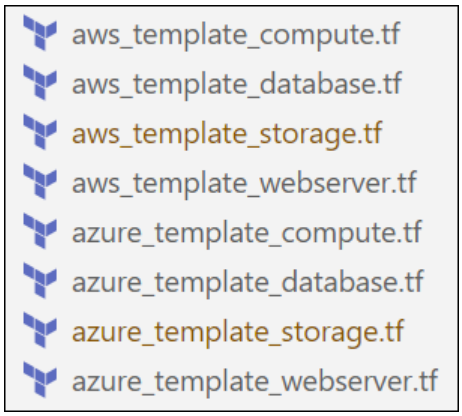
Recurso	Descrição	Documentação
Azure Virtual Machine	São instâncias de serviço de imagem que oferecem recursos de computação escalonáveis e sob demanda com preço baseado no uso.	https://azure.microsoft.com/pt-br/products/virtual-machines
Azure Virtual Machine Scale Sets	Permitem criar e gerenciar um grupo de VMs com balanceamento de carga.	https://learn.microsoft.com/pt-br/azure/virtual-machine-scale-sets/overview
Azure App Service	É um serviço com base em HTTP para hospedagem de aplicativos Web, APIs REST e back-ends móveis.	https://learn.microsoft.com/pt-br/azure/app-service/overview
Azure Functions	É uma plataforma de computação sem servidor controlada por eventos que ajuda você a desenvolver com mais eficiência usando a linguagem de programação de sua escolha.	https://azure.microsoft.com/pt-br/products/functions
Azure SQL Database	É um serviço de banco de dados relacional da Microsoft.	https://azure.microsoft.com/pt-br/products/azure-sql/database
Azure Database for MariaDB	É um serviço de banco de dados relacional da MariaDB.	https://azure.microsoft.com/pt-br/products/mariadb
Azure Database for MySQL	É um serviço de banco de dados relacional do MySql.	https://azure.microsoft.com/pt-br/products/mysql
Azure Database for PostgreSQL	É um serviço de banco de dados relacional do PostgreSQL.	https://azure.microsoft.com/pt-br/products/postgresql
Azure Container Storage	É uma solução de armazenamento de objetos da Microsoft para a nuvem.	https://learn.microsoft.com/pt-br/azure/storage/blobs/storage-blobs-introduction

Fonte: Elaboração própria

3.2 ORGANIZAÇÃO DOS TEMPLATES

Os templates desta proposta são divididos em quatro categorias de recursos: Compute (a), Database (b), Storage (c) e WebServer (d). A Categoria Compute engloba recursos responsáveis por fornecer capacidade de processamento, como Azure Máquinas Virtuais e Elastic Compute Cloud (EC2). Na categoria Database concentra-se a gestão de recursos de banco de dados, como RDS Database e Azure SQL Database. Já a Storage trata dos recursos de armazenamento, como Amazon S3 e Azure Storage Container. Por fim, a Web Server está focada na implantação e gerenciamento de aplicativos e serviços Web, como Elastic Beanstalk e Azure App Service. A Figura 8 demonstra a organização dos arquivos templates que estão na raiz do repositório (GitHub), os quatro primeiros para a AWS e quatro últimos para a AZURE.

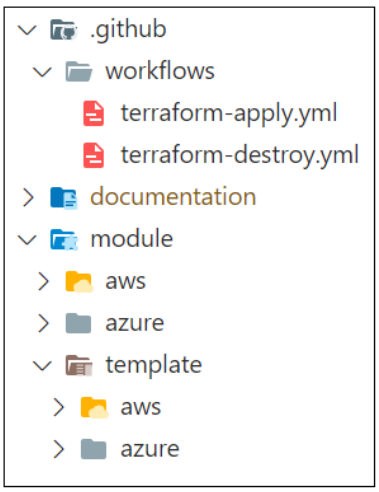
Figura 8: Disposição dos template



Fonte: Elaboração Própria

Além dos arquivos citados anteriormente, estes templates estão organizados em três diretórios, que são: "*.github/workflows*" (a), "*documentation*" (b) e "*module*" (c). O diretório "*.github/workflows*" foi usado para armazenar arquivos de configurações do *GitHub Actions*: *'terraform-apply.yml' e 'terraform-destroy.yml'*. O diretório "*documentation*" é destinado aos arquivos que descrevem como provisionar os recursos, incluindo seus detalhes. Finalmente, o diretório o "*module*" foi utilizado para armazenar os módulos Terraform. A Figura 9 apresenta a disposição dos diretórios.

Figura 9: Organização dos diretórios



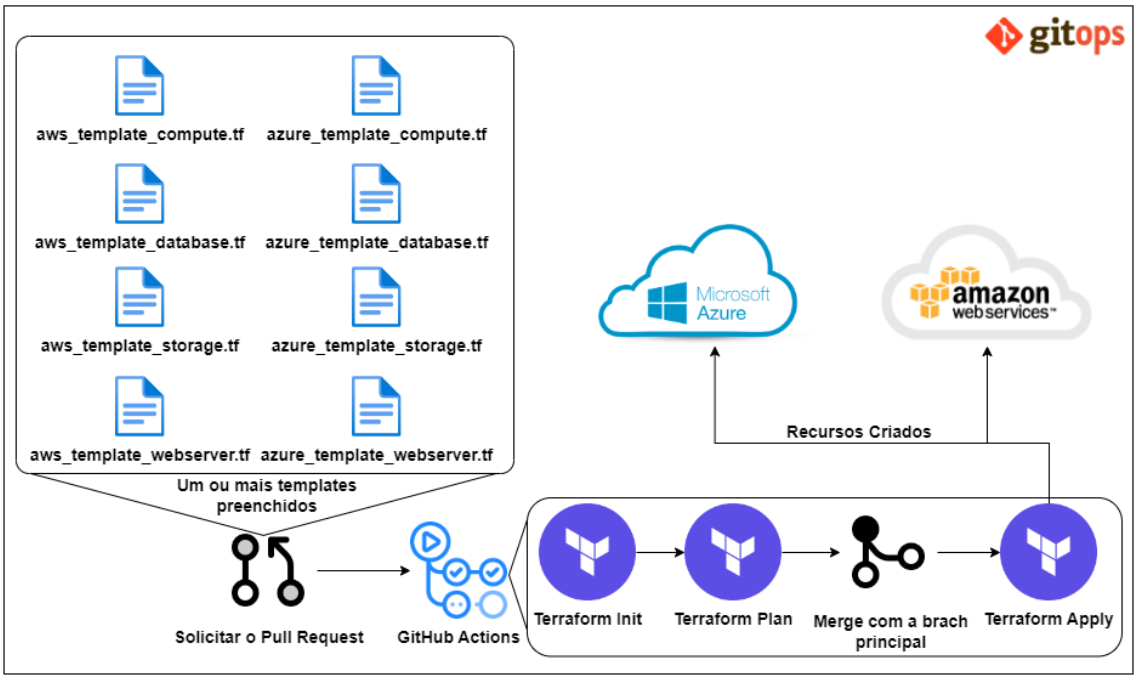
Fonte: Elaboração Própria

3.3 UTILIZAÇÃO DOS TEMPLATES

Os templates desenvolvidos neste projeto têm como característica principal de facilitar a utilização da ferramenta *Terraform* e automatizar o processo de provisionamento de recursos em múltiplas nuvens. Na Figura 10, é apresentado o modelo do processo para a utilização desses templates. O ciclo inicia com a seleção dos recursos desejados, seguido pela criação de um *Pull Request* (*PR*) para a *branch* principal. Neste momento, é desencadeado um evento no *GitHub Actions* com a abertura do *PR*, iniciando duas tarefas: a primeira inicializar o *Terraform* (*terraform init*) e a segura é responsável pelo planejamento dos recursos (*terraform plan*).

A próxima etapa envolve a revisão e aprovação do *PR*, seguida pelo merge com a *branch* principal. Uma vez aprovado, inicia o evento de fechamento do *PR*, dispara a tarefa responsável por executar o comando "*terraform apply*", conforme detalhado na Seção 2.3. Por fim, os recursos são provisionados.

Figura 10: Processo de utilização dos Templates



Fonte: Elaboração Própria

Os templates desenvolvidos nesse repositório tem objetivo simplificar o processo de provisionamento. Os usuários finais têm a opção de escolher entre os

templates disponíveis. Toda documentação necessária está incluída para orientar sobre os recursos e o preenchimento dos templates.

Já o processo de provisionar os recursos de forma automática foi possível através do *GitHub Actions* possibilitando um processo autônomo. O processo de construção só é concluído através da revisão e aprovação do *PR*. A Figura 11 ilustra o arquivo '*terraform-apply.yml*' mencionado na Seção anterior.

Figura 11: Tarefas no GitHub Actions

.github > workflows > terraform-apply.yml

```
1   name: 'Terraform-Apply'
2
3   on:
4     pull_request:
5       branches:
6         - main
7       types:
8         - opened
9         - closed
10              [...]
14  jobs:
15    terraform:
16      name: 'Terraform'
17      runs-on: windows-latest
18
19      defaults:
20        run:
21          shell: bash
22
23      steps:
              [...]
32      - name: Terraform Init
33        run: terraform init
34
35      - name: Terraform Plan
36        run: terraform plan
37
38      - name: Terraform Apply
39        if: github.event.pull_request.merged == true
40        run: terraform apply -auto-approve
```

Fonte: Elaboração Própria

No conteúdo do arquivo, apresentado na Figura 11, é apresentado, entre nas linhas 3 a 9 o modo de disparo que ocorre com *PR*, aberto ou fechado, para a *branch* '*main'*. Nas linhas 14 a 22, é apresentado que o fluxo está com o nome 'terraform', e que será executado em numa máquina *Windows* do *GitHub Actions*. Nas linhas 23 a 40 são apresentados os passos de execução do (*CI/CD*), com

destaque para linha 39 onde encontra-se uma condição que só será executada quando houve um merge (fechando do *PR*).

4 CONSIDERAÇÕES FINAIS

Este trabalho abordou os conceitos sobre Infraestrutura como Código com ênfase no provisionamento de infraestrutura em múltiplas nuvens e abordou os conceitos a respeito da utilização do *GitHub Actions* para automatizar os comandos do *Terraform*, onde foram desenvolvidos vários templates para provisionar alguns recursos da *Microsoft Azure* e *Amazon Web Services*.

O desenvolvimento dos Templates foi feito através de categorização dos recursos, facilitando assim a localização e organização do código. A escolha das plataformas se deu por conta de pesquisas da utilização de mercado e popularização delas.

A principal limitação no desenvolvimento foi por conta do acesso as plataformas utilizadas, tendo em vista que essas plataformas, na sua maioria, têm um prazo curto de utilização gratuita, esse aspecto dificulta o aprendizado sobre os recursos. Há também limitações em relações ao provisionamento dos recursos, seja por questão de disponibilidade em certas regiões como também limitações em hardware dos provedores cloud. E por fim, há limitações no terraform por não contemplar todos os recursos na nuvem.

Para os trabalhos futuros, os templates poderão completar outras plataformas cloud, como o *Google Cloud Plataform* (*GCP*) a fim de aumentar as possibilidades de escolha pelo usuário. Como também utilizar outros (*CI/CD*), por exemplo o *Jenkins*. Outro tema importante, que poderá ser abordado futuramente com os templates, é sobre a criação de usuários e credenciais a fim de limitar acesso aos recursos cloud.

Todos os templates assim como os arquivos desse diretório estão disponibilizados no repositório do GitHub no seguinte endereço: https://github.com/joaopedro2017/template_terraform_multiplataform.

ABSTRACT

Due to the continuous evolution and widespread adoption of cloud resources, it becomes imperative to automate infrastructure provisioning and versioning processes. In order to simplify resource provisioning in the cloud, this work proposes the development of Terraform Templates. These templates have been categorized and created for two of the major cloud platforms: Microsoft Azure and Amazon Web Services, as indicated by market research. To enhance provisioning autonomy, workflows have been established through GitHub Actions, which depend on code review and approval before carrying out provisioning. The entire work has been properly versioned in the GitHub repository.

Keywords: *Templates. Terraform. GitHub Actions. GitHub.*

REFERÊNCIAS

AMARAL, Lucas Barros. **Adoção de Multi-Nuvem e Nuvem Híbrida nas Organizações**. Faculdade de Tecnologia de São Paulo, São Paulo, 2022. Disponível em: <http://ric.cps.sp.gov.br/bitstream/123456789/10544/1/ads_2022_1_lucasbarrosamaral_adocaodemultinuvemenuvemhibrida.pdf>. Acesso em: 21/08/2023.

BRIKMAN, Yevgeniy. **Terraform: Up and Running Writing Infrastructure as Code (2nd. ed.)**. 2019. O'Reilly Media, Inc.

COSENTINO, Valerio. IZQUIERDO, Javier L. Cánovas. CABOT, Jordi. **A Systematic Mapping Study of Software Development With GitHub**. IEEE Access, vol. 5, pp. 7173-7192, 2017. Disponível em: <https://ieeexplore.ieee.org/abstract/document/7887704> Acesso em: 13/11/2023

CLOUDZERO. **Cloud Computing Market Size And Key Insights You Need To Know In 2023**. Disponível em: <https://www.cloudzero.com/blog/cloud-computing-market-size/>. Acesso em: 31/10/2023.

GARTNER. **Quadrante Mágico para infraestrutura em nuvem e serviços de plataforma**. Disponível em <https://www.gartner.com/technology/media-products/reprints/AWS/1-2AOZQARL-PTB.html>. Acesso em: 15/08/2023.

HASHICORP. **Terraform Language Documentation**. Disponível em: <https://developer.hashicorp.com/terraform/language>. Acesso em: 08/03/2023.

KINSMAN, Timothy. WESSEL, Mairieli. GEROSA, Marco A. TREUDE, Christoph. **How Do Software Developers Use GitHub Actions to Automate Their Workflows?**. IEEE/ACM 18th International Conference on Mining Software

Repositories (MSR), 2021, pp. 420-431. Disponível em: <https://arxiv.org/pdf/2103.12224.pdf>. Acesso em: 07/10/2023

LÓPEZ-VIANA, Ramón. DÍAS, Jessica. PÉREZ. Jorge E. **Continuous Deployment in IoT Edge Computing: A GitOps implementation**. 17th Iberian Conference on Information Systems and Technologies (CISTI), Madrid, Spain, 2022, pp. 1-6. Disponível em: <https://ieeexplore.ieee.org/document/9820108>. Acesso em: 20/09/2023.

MODI, Ritesh. **Deep-Dive Terraform on Azure: Automated Delivery and Deployment of Azure Solutions (1st ed.)**. 2021. Apress.

MORAES, Pedro Augusto Alcântara Ribeiro. **Uma Análise de Desempenho Multi-Cloud de Uma Aplicação Baseada em Microsserviços**. Universidade Federal do Piauí (UFPI), Picos, 2021. Disponível em: <https://ufpi.br/arquivos_download/arquivos/PICOS/Not%C3%ADcias/PICOS_2022/Biblioteca/2021/Sistemas_de_Informa%C3%A7%C3%A3o_2021/PEDRO_AUGUSTO_ALC%C3%82NTARA_RIBEIRO_MORAES.pdf>. Acesso em: 18/08/2023.

NASCIMENTO, Gustavo Henrique Alvim. **Computação em Nuvem com a Plataforma Microsoft Azure**. PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS, GOIÂNIA, GO, 2020. Disponível em: <https://repositorio.pucgoias.edu.br/jspui/bitstream/123456789/1070/1/TCC%20-%20Gustavo%20Henrique%20%28vers%C3%A3o%20final%29.pdf>. Acesso em: 19/09/2023.

PATNI, Jagdish Chandra. BANERJEE, Souradeep. TIWARI, Devyanshi. **Infrastructure as a Code (IaC) to Software Defined Infrastructure using Azure Resource Manager (ARM)**. International Conference on Computational Performance Evaluation (ComPE), Shillong, India, 2020, pp. 575-578. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9200030>. Acesso em 21/08/2023.

SINGH, Yashaswi. KANDAH, Farah. ZHANG, Weiyi. **A secured cost-effective multi-cloud storage in cloud computing**. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Shanghai, China, 2011, pp. 619-624. Disponível em: <https://ieeexplore.ieee.org/abstract/document/5928887>. Acesso em: 22/08/2023.

TERRAFORM. **What is Terraform?**. Disponível em: <https://developer.hashicorp.com/terraform/intro>. Acesso em: 08/03/2023.