

IMPLEMENTAÇÃO DE INFRAESTRUTURA COMO CÓDIGO PARA PROVISIONAMENTO E DEPLOY DE APLICAÇÕES

Jones Luis Noll¹, Fabrício Pretto²

Resumo: A evolução tecnológica dos últimos anos trouxe uma necessidade crescente por softwares, devendo estes, estarem sempre disponíveis, acessíveis e atualizados para seus usuários. Para atender as demandas dos sistemas de informação, a infraestrutura passou a ser construída através de código, tornando-se assim, dinâmica e escalável. Ao utilizar os conceitos de DevOps e Infraestrutura como Código, este trabalho teve como objetivo a criação automatizada de toda a infraestrutura necessária para execução do ciclo de desenvolvimento de um software, desde o registro da mudança até sua entrega. Foram realizadas a criação e a configuração das máquinas virtuais e contêineres necessários, a orquestração dos mesmos em um cluster Kubernetes, finalizando com o *deploy* da aplicação e do banco de dados. Como resultado, após a execução de todos os processos, um sistema de informação encontrou-se em execução na estrutura e disponível aos usuários. Foram analisados os tempos necessários para a criação automatizada da estrutura e os benefícios na adoção da Infraestrutura como Código.

Palavras-chave: Infraestrutura. DevOps. Provisionamento. Automação. Deploy.

1 INTRODUÇÃO

O mercado cada vez mais competitivo, tem colocado as empresas em uma corrida pelo lançamento de novas funcionalidades de software, disponíveis para diferentes dispositivos e sistemas operacionais, integrando-os a outros serviços. Isso não só aumentou a complexidade do processo de desenvolvimento de software, como também da infraestrutura, dificultando que novas versões sejam aplicadas em produção de maneira eficaz e eficiente.

Outro fator que contribui para a ineficiência na evolução do software, diz respeito às diferentes habilidades, vocabulários e experiências entre as equipes de operações e desenvolvimento de software. Além disso, tais equipes possuem objetivos diferentes, onde enquanto o desenvolvimento objetiva

1 Graduado em Sistemas de Informação - Univates - jnoll@universo.univates.br

2 Mestre em Ciência da Computação - Univates - fabricao.pretto@univates.br

a evolução do software, voltado ao lançamento de novas funcionalidades, as equipes de operações objetivam a estabilidade, sendo completamente resistentes a qualquer mudança, ocasionando conflito entre as equipes (BEYER et al., 2016).

O termo DevOps surgiu para resolver o conflito entre as equipes de desenvolvimento e operações, de modo que consigam trabalhar em conjunto, possuindo os mesmos objetivos e responsabilidades. A Amazon (2019), definiu DevOps como uma combinação de filosofias culturais, práticas e ferramentas que objetivam aumentar a capacidade de uma organização entregar software e serviços, aumentando assim a qualidade de atendimento aos seus clientes e a competitividade.

Entre os pilares da metodologia DevOps estão a Integração e Entrega Contínua, onde o software vive em constante evolução e novas funcionalidades são disponibilizadas rapidamente aos usuários. Vadapalli (2018) define Integração Contínua como o trabalho em conjunto de diversos desenvolvedores, integrando seus códigos com o auxílio de um repositório. Entre os benefícios, o autor cita o rápido acesso ao código mais recente e a redução de erros em códigos. A entrega contínua, é definida pelo autor como uma entrega rápida e automatizada de software repetitivamente, possuindo como principais benefícios a automação e a confiabilidade em que o mesmo é disponibilizado.

O DevOps utiliza muita automação, inclusive na infraestrutura, criando o conceito de *Infrastructure as Code* (IaC), que em tradução livre, significa Infraestrutura como Código. A Hewlett Packard Enterprise (2019) definiu que Infraestrutura como Código trata a infraestrutura da mesma forma como um software programável, onde toda a infraestrutura necessária para a execução de um software pode ser criada através da execução de um código previamente desenvolvido.

A infraestrutura como código ajuda a resolver problemas de escalabilidade e gerenciamento, tendo como principais benefícios: o controle de versão do código, podendo facilmente avançar ou retroceder para um estado conhecido; a escalabilidade, tornando possível duplicar ambientes de forma rápida e segura; e a recuperação de desastres, pelo fato da infraestrutura toda ser baseada em código, torna possível restaurar o ambiente através da execução do código existente em *backup* (NELSON-SMITH, 2013).

O objetivo geral deste trabalho foi implementar e avaliar o processo de um ambiente de homologação e produção de um produto de software, fazendo uso dos conceitos e técnicas de Infraestrutura como Código. Como objetivos específicos foram definidos e realizados: pesquisar e conhecer os conceitos de DevOps e IaC; modelar a arquitetura da infraestrutura a ser construída; implantar ferramentas que tornem possível o provisionamento, gerenciamento de configurações e *deploy* de servidores; manter o código da infraestrutura versionado; automatizar o processo de *deploy* de software; realizar a integração entre as ferramentas utilizadas; e analisar os resultados obtidos.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentadas as referências bibliográficas utilizadas para teorizar e fundamentar a temática deste trabalho.

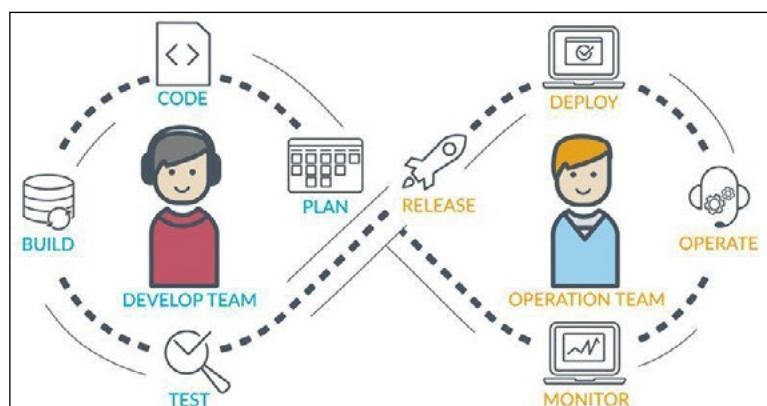
2.1 DevOps

Sharma (2014), define DevOps como a junção abreviada de desenvolvimento e operações, sendo basicamente uma metodologia de trabalho baseada no modelo ágil, onde todos os envolvidos trabalham para realizar a entrega do software de maneira contínua.

DevOps se caracteriza pelo envolvimento da Tecnologia da Informação (TI) em todas as fases do design e desenvolvimento de uma aplicação, substituindo interações humanas por forte automação, e utilizando de metodologias e ferramentas em atividades de operação (BEYER et al., 2016). Hüttermann (2012) acrescenta que DevOps cria uma cultura de compartilhamento, onde as pessoas compartilham ideias, processos e ferramentas. Na cultura DevOps, as pessoas estão sempre antes de processos e ferramentas, onde o software é feito por e para pessoas.

Um dos pilares da metodologia DevOps é a entrega contínua, que segundo Kim et al. (2018), é garantir que o código e a infraestrutura estejam alinhados, e em um estado implementável. Dessa forma, qualquer código pode passar para produção de forma segura. Para isso, é utilizado *build*, teste e integração contínuos.

Figura 1 - Ciclo DevOps



Fonte: Ventapane (2019, texto digital).

A Figura 1 mostra que o ciclo de vida DevOps também possui as fases do desenvolvimento de software, entretanto, com integração contínua entre as fases e muita automação de processos (HUMBLE; FARLEY, 2010).

2.2 Infraestrutura como código

Para Nelson-Smith (2013), a infraestrutura como código teve início em 2006, quando a Amazon Web Services lançou o *Elastic Compute Cloud* (EC2). A partir daquele momento, qualquer ideia para uma aplicação *web* pode ser implementada em questão de semanas, fazendo com que pequenas empresas tivessem um crescimento acelerado, e mesmo sendo lideradas por desenvolvedores, tiveram que se preocupar com rotinas de equipes de operações, como adicionar máquinas idênticas, fazer *backup*, escalar banco de dados, entre outras. Como estas equipes eram pequenas e tinham que administrar ambientes complexos, começaram a surgir as primeiras ferramentas de gestão de configuração.

Infraestrutura como código pode ser definida como o uso das práticas do desenvolvimento de software para a automação da infraestrutura, utilizando-se de rotinas consistentes e repetíveis, possibilitando a alteração de sistemas e suas configurações, através de sistemas com validações (MORRIS, 2016). Artač et al. (2017), definem infraestrutura como código como o uso de código fonte para projetos de infraestrutura, sendo que todo o conjunto de *scripts*, códigos de automação e configuração, modelos, dependências necessárias e parâmetros de configuração sejam expressados com a utilização do mesmo. Hüttermann (2012) acrescenta que o objetivo da infraestrutura como código é lidar com as suas configurações da mesma forma como se lida com o desenvolvimento de um sistema, escolhendo a linguagem ou ferramenta adequada para iniciar o desenvolvimento de uma solução que atenda às necessidades, que seja executável e que possa ser aplicada de forma eficiente e repetitiva.

Segundo a Instruct (2017), a infraestrutura como código é baseada na automação, sendo composta por três atividades principais:

- a) Gerência de configuração: utilização de ferramentas para manter a infraestrutura padronizada.
- a) Orquestração: consiste em um conjunto de instruções a serem executadas para cumprir um objetivo, em tempo real e com a possibilidade de ser paralela.
- b) Provisionamento: camada entre os recursos disponíveis e a necessidade, sendo responsável por criar novos ambientes, de maneira rápida e fácil.

Nelson-Smith (2013) cita benefícios que a infraestrutura como código traz, e que ajudam a resolver problemas de escalabilidade e gerenciamento:

- a) Repetibilidade: a construção da infraestrutura em uma linguagem de programação de alto nível a torna ordenada e repetível. O mesmo código deve produzir sempre a mesma saída, levando a certeza que seja possível recriar o ambiente;

- a) Automação: implementar aplicativos com uso de ferramentas maduras, escritas em linguagens modernas, faz com que o próprio fato de abstrair infraestruturas gere os benefícios da automação. Sato (2014) acrescenta que investir em automação resulta na execução mais rápida das tarefas, além de diminuir a possibilidade de erros humanos;
- b) Agilidade: o gerenciamento de código fonte e controle de versões trazem a possibilidade de avançar ou retroceder para um estado conhecido. Desta forma, somos capazes de realizar mudanças drásticas na topologia com facilidade, respondendo com velocidade às mudanças impostas pelo negócio.
- c) Escalabilidade: a repetibilidade e a automação trazem a possibilidade de aumentar a quantidade de servidores facilmente, principalmente quando o ambiente possui os benefícios do provisionamento rápido de hardware da nuvem.
- d) Reafirmação: todos os benefícios trazem segurança, principalmente pelo fato da arquitetura e o design da infraestrutura serem modelados, e não apenas implementados. Mas ter o código-fonte da infraestrutura significa ter uma base de conhecimento razoável, podendo visualizar como os sistemas funcionam e não correndo o risco de apenas um profissional ter o entendimento completo do ambiente;
- e) Recuperação de desastres: como toda a infraestrutura é escrita como código, caso ocorra um evento catastrófico que destrua todo o ambiente de produção, será possível facilmente restaurar o ambiente a partir do backup, reimplementando toda a infraestrutura. Além disso, a infraestrutura como código facilita procedimentos de testes de recuperação.

3 METODOLOGIA

Este trabalho pode ser considerado uma pesquisa aplicada, que conforme Marconi e Lakatos (2003), é dirigida a resolver problemas específicos, tendo como objetivo gerar conhecimento sobre o tema trabalhado para sua aplicação prática.

Os objetivos de pesquisa deste trabalho enquadram-se em exploratórios, que segundo Botelho e Cruz (2013), tem como um dos principais objetivos promover a familiaridade sobre o tema. Além disso, pode ser considerada uma pesquisa experimental.

Essencialmente, a pesquisa experimental consiste em determinar um objeto de estudo, selecionar as variáveis que seriam capazes

de influenciá-lo, definir as formas de controle e de observação dos efeitos que a variável produz no objeto (GIL, 2002, p. 47).

Inicialmente realizou-se uma pesquisa exploratória sobre infraestrutura como código e DevOps, reunindo informações sobre seus benefícios, práticas de adoção e ferramentas comumente utilizadas. Baseando-se no material coletado, iniciou-se a elaboração da proposta a ser desenvolvida, definindo o escopo do projeto e os objetivos que deveriam ser alcançados.

Com a fundamentação teórica concluída, iniciou-se o desenvolvimento da solução proposta. Primeiramente foram exploradas as ferramentas disponíveis em infraestrutura como código e definidas as que seriam utilizadas. Na sequência foi desenvolvida uma pequena aplicação a ser utilizada para testes e validação do projeto, que deveria ser executada por uma infraestrutura baseada em código.

Com a aplicação concluída, foi desenvolvida a infraestrutura para comportar e executar a aplicação, baseada apenas em código e utilizando de ferramentas de infraestrutura como código. A partir de então, foram analisados os tempos de construção e reconstrução da estrutura, bem como, os benefícios e os problemas encontrados em seu desenvolvimento.

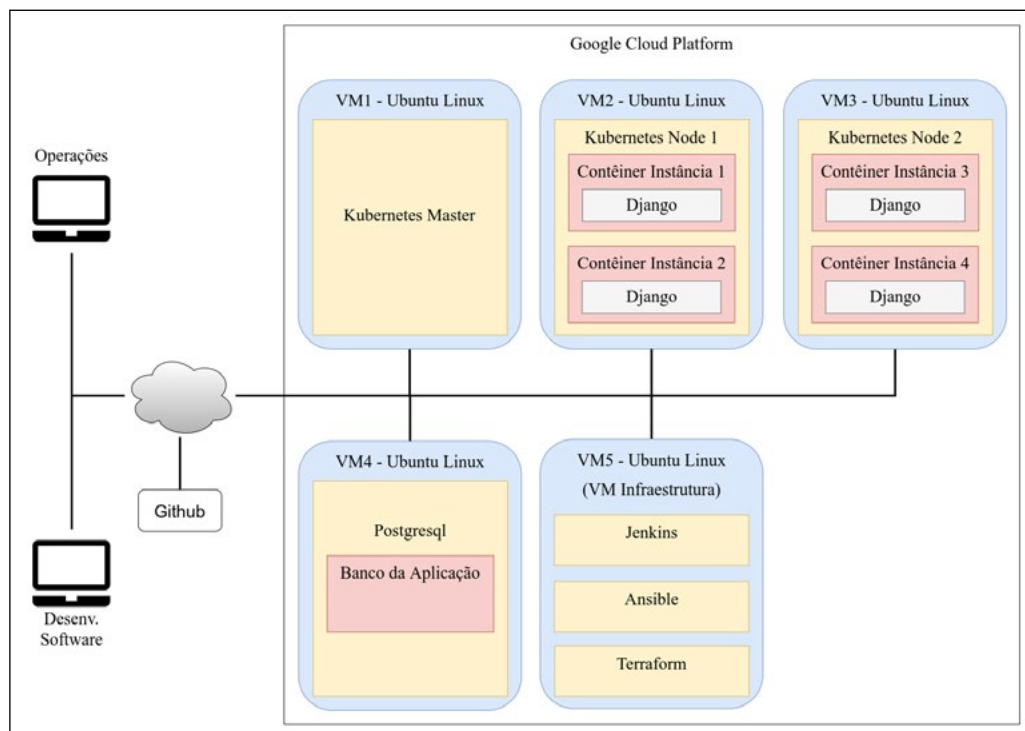
Para o desenvolvimento do projeto foram utilizadas ferramentas de IaC reconhecidas na área de TI por sua eficiência, com documentação ampla e de qualidade e priorizando ferramentas livres para uso. Entre estas ferramentas, estão: Terraform, Ansible, Jenkins, Docker, Django, Kubernetes, Postgresql e Google Cloud Platform. Destas, apenas a *Cloud* do Google requer pagamento para uso. Cada uma dessas ferramentas será especificada no Capítulo 4, abordando seu papel no cenário desenvolvido.

4 EXPERIMENTAÇÃO

Este trabalho teve como objetivo a criação de uma infraestrutura completa, desde o início, com todos os recursos necessários para execução de um software em um *cluster* de contêineres hospedado em *cloud*, com a utilização de infraestrutura como código.

Para concepção da solução foi planejada uma arquitetura (FIGURA 2) que segmenta a infraestrutura em cinco máquinas virtuais hospedadas em nuvem.

Figura 2 - Visão geral da arquitetura desenvolvida



Fonte: Elaborado pelo autor (2020).

Na Figura 2 é exibido o *cluster* Kubernetes composto por um nó de gerenciamento e dois nós de trabalho, representado pelas VM1, VM2 e VM3. No Kubernetes cada VM pertencente ao *cluster* é chamada de nó, podendo ser estes de trabalho, que executam os contêineres e as tarefas envolvidas com suas execuções, ou de gerenciamento, que gerenciam o *cluster* e os outros nós. Neste *cluster* são executados contêineres com o *framework* Django e os serviços necessários para a execução da aplicação. A VM4 é responsável por executar o banco de dados da aplicação. Já a VM5 é utilizada pela equipe de operações para criar o ambiente, possuindo as ferramentas necessárias para criar a estrutura e realizar o *deploy* da aplicação. Na arquitetura também é exibida a ferramenta Github, utilizada para armazenamento do código da aplicação e da infraestrutura.

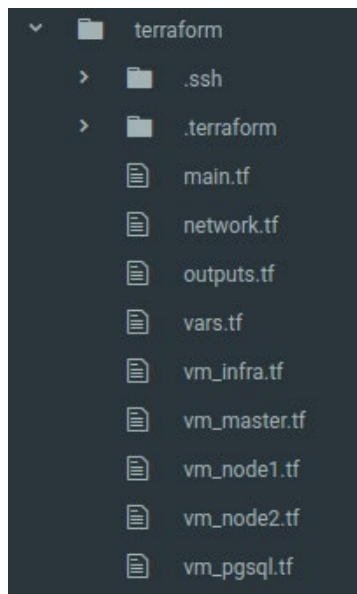
4.1 Provisionamento da infraestrutura em *cloud*

A criação das máquinas virtuais, configurações de rede e de *firewall* na plataforma de *cloud* do Google foram realizadas com a utilização da ferramenta Terraform. Estas configurações na ferramenta foram realizadas localmente no computador do autor. O código de cada configuração foi versionado para o

repositório Github, tornando-o acessível e executável de qualquer computador com acesso ao repositório.

A estrutura de arquivos criada na ferramenta é demonstrada na Figura 3, onde a nomenclatura dos arquivos foi definida pelo autor.

Figura 3 - Estrutura de arquivos do Terraform



Fonte: Elaborado pelo autor (2020).

O diretório `.ssh` possui as chaves instaladas pelo Terraform nas VMs criadas, para posteriormente permitir o acesso remoto para o usuário, administrador da rede ou outras ferramentas de IaC. Já o diretório `.terraform` possui arquivos de controle e binários utilizados pela ferramenta.

O arquivo `main.tf` (Figura 4) possui as configurações de conexão com o Google *Cloud*, como o apontamento para o arquivo com a credencial a ser utilizada, o nome do projeto e região geográfica do *Data Center* a ser utilizado. A autenticação com o Google *Cloud* é realizada através de uma conta de serviço criada anteriormente na plataforma, com permissões para criar e modificar os recursos do projeto.

Figura 4 - Terraform: arquivo main.tf

```
1 terraform {
2   backend "gcs" {
3     bucket     = "tcc-terraformstate"
4     prefix     = "tcc-infraascode"
5     credentials = "../uteis/Formatura-786e11b5710f.json"
6   }
7 }
8
9 provider "google" {
10  credentials = file("../uteis/Formatura-786e11b5710f.json")
11  project     = "formatura-249122"
12  region      = var.regiao
13 }
14
15 resource "random_id" "instance_id" {
16   byte_length = 8
17 }
```

Fonte: Elaborado pelo autor (2020).

No arquivo `network.tf` são descritas as configurações de rede e *firewall* que devem ser criadas pelo Terraform. No arquivo, além dos endereços IPs privados a serem utilizados, são definidas as portas que estarão permitidas e visíveis na internet. Já os arquivos `vm_infra.tf`, `vm_master.tf`, `vm_node1.tf`, `vm_node2.tf` e `vm_pgsql.tf` possuem as configurações de criação de VMs, como é exemplificado na Figura 5. Nestes arquivos são definidas, além de outras configurações, o nome da VM, o tipo de hardware que ela deve utilizar, nome da imagem no Google, tamanho do disco e, através das *tags* de *firewall*, quais portas devem estar disponíveis para a internet. Neste arquivo também são definidos os usuários do sistema operacional a serem criados e as chaves de SSH que serão utilizadas em seus acessos, contidas no diretório `.ssh` do Terraform.

Figura 5 - Terraform: arquivo vm_infra.tf

```
1 resource "google_compute_instance" "vminfra" {
2   name                = "${var.projeto}-infra-${var.env}"
3   machine_type        = var.tipo_vm
4   zone                = var.vm_infra_zona
5   allow_stopping_for_update = true
6   tags                = ["ssh", "jenkins"]
7   boot_disk {
8     initialize_params {
9       image = var.nome_imagem
10      size  = 20
11    }
12  }
13  metadata = {
14    ssh-keys = "noll:${file(".ssh/id_rsa.pub")} \nubuntu:${file(".ssh/id_rsa.pub")}
15    *        \nubuntu:${file(".ssh/id_rsa_jenkins.pub")}"
16  }
17  network_interface {
18    subnetwork = google_compute_subnetwork.private_subnet.name
19    access_config {
20    }
21  }
22 }
```

Fonte: Elaborado pelo autor (2020).

O arquivo outputs.tf possui informações que o Terraform deverá exibir após sua execução, como por exemplo, os endereços IPs das VMs criadas. Já no arquivo vars.tf, são definidas variáveis customizadas utilizadas em todos os arquivos do Terraform, facilitando uma eventual mudança ou até mesmo o reaproveitamento de código.

A criação da estrutura é antecedida de uma etapa inicial, onde é criada a VM 5, no Terraform denominada VM Infraestrutura, e todas as configurações de rede necessárias para o seu funcionamento. Posteriormente, com a instalação do Jenkins e de todas as ferramentas necessárias para o gerenciamento da infraestrutura, é ele que foi utilizado para disparar as rotinas de criação do restante da estrutura e *deploy* da aplicação. Este processo de execução parcial do Terraform foi realizado passando parâmetros para que ele crie apenas esta parte da infraestrutura em *cloud* (FIGURA 6).

Figura 6 - Terraform: criação da VM infraestrutura

```
[noll:~/GIT/tcc-infraascode/terraform]$ terraform apply -target=google_compute_instance.vminfra -target=google_compute_firewall.allow-jenkins -target=google_compute_firewall.allow-ssh
```

Fonte: Elaborado pelo autor (2020).

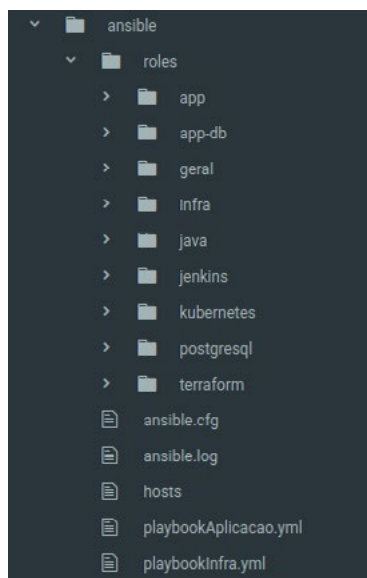
Após a execução do comando exibido na Figura 6, a VM Infraestrutura estará criada e em execução no Google *Cloud*, entretanto sem nenhuma configuração personalizada do Sistema Operacional e sem nenhum serviço em execução. Estas configurações ficam a cargo do Ansible.

4.2 Gerenciamento de configurações

O gerenciamento de configurações da estrutura é realizada pelo Ansible, ficando ele responsável pela atualização e configuração do sistema operacional, bem como, a instalação e configuração de aplicativos. Neste projeto o Ansible exerce um papel fundamental, sendo ele responsável por instalar e configurar toda a estrutura após a criação das VMs pelo Terraform, finalizando sua execução com a estrutura concluída e com a aplicação de homologação sendo acessível pela internet.

Um dos arquivos mais importantes do Ansible é o *playbook*, nele são definidas quais tarefas devem ser executadas, a ordem de execução e em qual host (computador hospedeiro alvo). A estrutura de arquivos utilizada neste projeto é demonstrada pela Figura 7, onde foram criados os arquivos `playbookAplicação.yml` (FIGURA 8) e `playbookInfra.yml`. Os *playbooks* foram separados para que, assim como o Terraform, seja disponibilizada primeiramente a VM Infraestrutura, onde após a mesma estar em execução e com o Jenkins configurado, a criação do restante da estrutura é realizada pelo Jenkins.

Figura 7 - Estrutura de arquivos do Ansible



Fonte: Elaborado pelo autor (2020).

Figura 8 - Ansible: *Playbook* Aplicação

```

1  ---
2  - name: Playbook Aplicação - Configurar Sistema Operacional
3    hosts: servers
4    roles:
5      - geral
6
7  - name: Playbook Aplicação - Configurar Postgresql
8    hosts: dbs
9    roles:
10     - postgresql
11     - app-db
12
13 - name: Playbook Aplicação - Configurar Cluster Kubernetes
14   hosts: kubernetes
15   roles:
16     - kubernetes
17
18 - name: Playbook Aplicação - Configurar a Aplicação
19   hosts: masters
20   roles:
21     - app
22   ...

```

Fonte: Elaborado pelo autor (2020).

Nos *playbooks* são configuradas as *roles* que devem ser executadas em cada *host* ou grupo de *hosts*. *Roles* são conjuntos de tarefas a serem executadas pelo Ansible, que podem ser criadas manualmente ou através do *download* de *roles* disponibilizadas pela comunidade no portal Ansible Galaxy (<https://galaxy.ansible.com/>). O Quadro 1 demonstra a função e a origem das *roles* utilizadas neste projeto.

Quadro 1 - Ansible: Roles utilizadas

Nome da Role	<i>Playbook</i>	Função	Origem
app	Aplicação	Realiza o <i>deploy</i> da aplicação no <i>cluster</i> Kubernetes.	Autor
app-db	Aplicação	Importa o último backup do Postgresql armazenado no Github.	Autor
geral	Aplicação e Infra	Configurações gerais e atualização de pacotes do sistema operacional.	Autor
infra	Infra	Instalação de todos os requisitos para a execução do Ansible, Docker e Django. Restore do último backup do Jenkins, importando todos os jobs criados anteriormente.	Autor

Nome da Role	Playbook	Função	Origem
java	Infra	Instalação do Java.	Ansible Galaxy
jenkins	Infra	Instalação do Jenkins e seus plugins.	Ansible Galaxy
kubernetes	Aplicação	Instalação e configuração do <i>cluster</i> Kubernetes.	Autor
postgresql	Aplicação	Instalação e configuração do Postgresql.	Ansible Galaxy
terraform	Infra	Instalação do binário do Terraform.	Autor

Fonte: Elaborado pelo autor (2020).

Uma *role* pode possuir diversas instruções em arquivos diferentes, chamadas de tarefas. Além disso, possui os diretórios *files* e *templates*, onde no primeiro são colocados arquivos que devem ser transmitidos para o sistema alvo, e no segundo *templates* de arquivos de configuração de aplicativos. A Figura 9 demonstra as tarefas da *role* App, que primeiramente configura a autenticação do *cluster* Kubernetes com o Google Cloud para que seja possível o *download* do contêiner da aplicação, e em seguida, realiza o *deploy* do mesmo e coloca a aplicação em execução.

Figura 9 - Ansible: Role App

```
1 ---
2 ### Chave para autenticao no repositorio do google
3 - name: Adicionar a chave do google
4   template: src=kubernetes.json dest=/home/ubuntu force=yes owner=ubuntu
5   group=ubuntu mode=0440
6
7 - name: Criar chave de autenticao
8   become_user: ubuntu
9   shell: 'kubectl create secret docker-registry gcr-json-key --docker-
10  server=eu.gcr.io --docker-username=_json_key --docker-password="$(cat ~/
11  kubernetes.json)" --docker-email= && touch /var/ansible/
12  cluster_gcpkey.txt'
13   args:
14     creates: /var/ansible/cluster_gcpkey.txt
15
16 - name: Patch serviceaccount
17   become_user: ubuntu
18   shell: 'kubectl patch serviceaccount default -p "{\"imagePullSecrets\":
19   [{"name\": \"gcr-json-key\"}]}" && touch /var/ansible/cluster_patchkey.txt'
20   args:
21     creates: /var/ansible/cluster_patchkey.txt
22
23 ### Subir a aplicacao
24 - name: Adicionando arquivo django.yaml
25   become_user: ubuntu
26   template: src=django.yaml dest=/home/ubuntu force=yes owner=ubuntu group=ubuntu
27   mode=0644
28
29 - name: Criar deployment
30   become_user: ubuntu
31   shell: kubectl apply -f django.yaml && touch /var/ansible/cluster_deployment.txt
32   args:
33     creates: /var/ansible/cluster_deployment.txt
34
35 - name: Criar service
36   become_user: ubuntu
37   shell: kubectl expose deployment django-deployment --type=LoadBalancer --port 80
38   --target-port 8000 --external-ip={{ansible_default_ipv4.address}} && touch /var/
39   ansible/cluster_service.txt
40   args:
41     creates: /var/ansible/cluster_service.txt
42 ...
```

Fonte: Elaborado pelo autor (2020).

Os endereços IPs e nomes dos alvos do Ansible são configurados no arquivo *hosts*, onde as máquinas são agrupadas de acordo com suas funções. Além disso, neste arquivo também estão algumas variáveis utilizadas pela ferramenta, como o método de conexão com os *hosts* e a versão do Python, utilizada pela ferramenta Ansible para execução das ações.

4.3 Automação com Jenkins

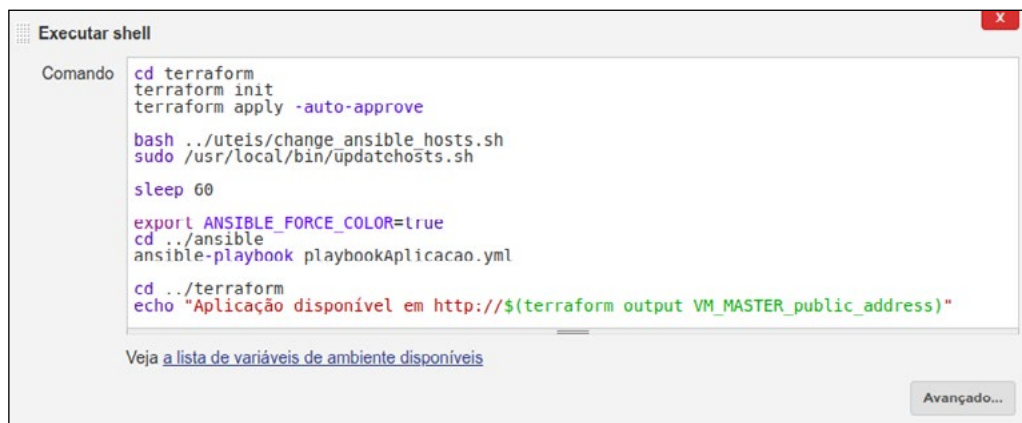
Neste projeto o Jenkins foi utilizado como um automatizador de rotinas, sendo ele responsável por criar e alterar toda a infraestrutura necessária para executar a aplicação, desde a criação das VMs com o Terraform, a configuração do ambiente com o Ansible e o *deploy* da aplicação.

Ao final da execução do *playbookInfra.yml* pelo Ansible, a VM Infraestrutura entra em execução com o Jenkins e todos os requisitos para gerenciar a estrutura e a aplicação. No processo de configuração do Ansible é restaurado o último *backup* do Jenkins, iniciando o mesmo já configurado com as rotinas necessárias:

- Atualizar Sistema de Notas: responsável por realizar o *download* da última versão da aplicação no Github, os testes que garantam o funcionamento da aplicação, as atualizações do banco de dados e a criação de um novo contêiner atualizado com a aplicação e o *rollout*³ do mesmo no *cluster* Kubernetes.
- Criar ou Atualizar Infraestrutura: é responsável pela criação ou atualização da infraestrutura da aplicação, criando e configurando as VMs e os softwares necessários. A rotina inicia com a execução do Terraform para a criação das VMs e na sequência executa dois scripts para atualizar o arquivo *hosts* do Ansible com os novos endereços IPs das VMs e também o arquivo */etc/hosts* da VM Infraestrutura. Na sequência é executado o *playbookAplicacao.yml* do Ansible que realiza a configuração de todo o ambiente, disponibilizando ao término a aplicação em execução. Os comandos de execução da rotina são exibidos na Figura 10.

³ *Rollout* quando relacionado a software, significa implementar um software ou uma atualização do mesmo.

Figura 10 - Jenkins: criar ou atualizar infraestrutura



```
Executar shell
Comando
cd terraform
terraform init
terraform apply -auto-approve

bash ../uteis/change_ansible_hosts.sh
sudo /usr/local/bin/üupdatehosts.sh

sleep 60

export ANSIBLE_FORCE_COLOR=true
cd ../ansible
ansible-playbook playbookAplicacao.yml

cd ../terraform
echo "Aplicação disponível em http://$(terraform output VM_MASTER_public_address)"

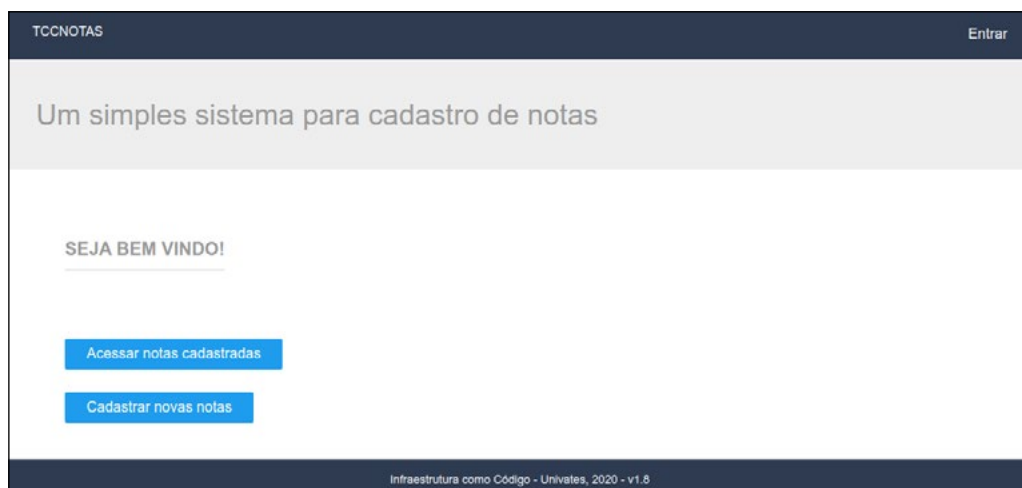
Veja a lista de variáveis de ambiente disponíveis
Avançado...
```

Fonte: Elaborado pelo autor (2020).

4.4 Aplicação para homologar infraestrutura

Para realizar testes e validar a infraestrutura desenvolvida, exibindo ao final da criação e configuração da mesma a execução de uma aplicação com acesso ao banco de dados, foi realizado o desenvolvimento de uma pequena aplicação de exemplo, possuindo como função o cadastro de notas de texto, com a ideia de lembretes, chamada TCCNOTAS (FIGURA 11).

Figura 11 - Aplicação: tela inicial



Fonte: Elaborado pelo autor (2020).

A aplicação é baseada em Python para Web, sendo utilizado para seu desenvolvimento o *framework* Django. Além de ser utilizada como a aplicação que valida a infraestrutura, estando em funcionamento após a execução de todas as ferramentas de IaC, ela também foi importante para demonstrar algumas práticas de DevOps, como os testes automatizados, o versionamento do banco de dados e o *deploy* automatizado e sem queda da aplicação.

Para realizar a configuração completa do ambiente e disponibilizar a aplicação em funcionamento, durante o processo de execução das ferramentas de IaC é restaurado o último *backup* válido do banco de dados. Para isso, foi utilizado como repositório do *backup* o Github, sendo o mesmo atualizado a cada 30 minutos quando a aplicação está em execução. É importante ressaltar que esta é uma aplicação com banco de dados mínimo, praticamente sem dados, onde a estratégia de *backup* e *restore* para aplicações maiores devem ser definidas de acordo com as necessidades e particularidades das mesmas.

4.5 Testes e validações

Para validar o funcionamento do experimento, foram realizados testes que executaram, durante 10 sessões, todos os scripts implementados. A média de tempo para criar toda a estrutura projetada foi de 18 minutos e 32 segundos. O tempo foi monitorado por um cronômetro, disparado no início do primeiro script e interrompido ao constatar que a criação da estrutura estava concluída. Este também poderia ser considerado o tempo necessário para duplicar a estrutura em uma estrutura de testes ou desenvolvimento, bastando apenas alterar algumas variáveis do Terraform e o apontamento de IPs no Ansible. O tempo de uma recuperação de desastres também fica muito próximo a este, adicionando a ele apenas as etapas de configuração de uma nova conta no Google *Cloud* e das permissões para conexão do Terraform, com tempo adicional, alterando variáveis do Terraform para outro provedor de *Cloud* suportado pela ferramenta.

Durante a realização do trabalho, foram realizados 110 *commits*⁴ na ferramenta de versionamento de código e a infraestrutura foi destruída e reconstruída 69 vezes no Google *Cloud*.

O Quadro 2 apresenta a quantidade de arquivos de código ou configuração gerados no trabalho, totalizando 151 arquivos divididos pelas ferramentas Terraform e Ansible, que são as ferramentas responsáveis pela criação e configuração de todo o ambiente. Arquivos utilizados por outras ferramentas, como por exemplo o Jenkins, estão incluídos em sua *role* correspondente.

4 Envio de atualização de código para a ferramenta de versionamento de código.

Quadro 2 - Número de arquivos gerados

Ferramenta	Função/Tarefa	Número de arquivos
Terraform	Configurações gerais	3
	Criar VMs em <i>Cloud</i>	8
	Configurações de rede e <i>firewall</i>	1
Ansible	Configurações gerais	4
	Role app	4
	Role app-db	6
	Role geral	4
	Role infra	9
	Role java	5
	Role jenkins	76
	Role kubernetes	5
	Role postgresql	23
	Role terraform	3
Total		151

Fonte: Elaborado pelo autor (2020).

Com as ferramentas de IaC foi possível realizar todos os procedimentos necessários para criar e configurar a infraestrutura proposta, desde a criação das VMs, a configuração do sistema operacional, a configuração de um *cluster* Kubernetes, e até mesmo, o *deploy* da aplicação. É notável o quanto a infraestrutura definida como código fica facilmente escalável, uma vez que, para replicar toda a estrutura a partir desse momento basta executar poucos comandos e aguardar um tempo inferior a 20 minutos. A mesma tarefa no modelo tradicional, instalando e configurando manualmente os serviços, representaria horas de trabalho. Além disso, resolve a dependência com os provedores de *Cloud*, uma vez que torna possível com poucas alterações de código o seu provisionamento em outra estrutura, sendo também considerado um ótimo processo de recuperação de desastres.

5 CONSIDERAÇÕES FINAIS

Cada vez mais a tecnologia faz parte do dia-a-dia das pessoas, onde a demanda por sistemas cresce de forma acelerada. Para atender esta demanda, foram criadas diversas metodologias de desenvolvimento de software com agilidade, entretanto a infraestrutura tradicional não acompanhou em mesmo ritmo. Com o uso de ferramentas de automação da infraestrutura, torna-se a mesma escalável e dinâmica, atendendo com velocidade as necessidades de suas aplicações.

Entre os diversos benefícios da adoção da infraestrutura como código, com este trabalho foi possível evidenciar a recuperação de desastres. Uma vez que a infraestrutura completa está em formato de código, basta alterar apontamos do provedor de *Cloud* para em alguns poucos minutos a estrutura estar em execução novamente em outros servidores. Isso também acaba trazendo a independência do provedor de *Cloud*, onde no cenário tradicional uma migração de provedor acaba sendo demorada e trabalhosa.

Ao utilizar a infraestrutura baseada em código, a documentação básica do ambiente é automática, uma vez que o próprio código também é documentação. Além disso, manter o código versionado garante que alterações mal sucedidas não causem um desastre, havendo sempre a possibilidade de retornar a infraestrutura para um estado conhecido e funcional. Para o sucesso de uma infraestrutura construída como código, é fundamental a repetitividade de execução das ferramentas de IaC, pois somente assim serão evitadas alterações de forma manual no ambiente, onde na próxima execução se não forem incluídas como código, serão removidas.

REFERÊNCIAS

AMAZON. **O que é DevOps?**. 2019. Disponível em: <<https://aws.amazon.com/pt/devops/what-is-devops/>>. Acesso em: 08 out. 2019.

ARTAČ, Matej; BOROVŠAK, Tadej; DI NITTO, Elizabetta; GUERRIERO, Michele; TAMBURRI, Damian A. DevOps: Introducing Infrastructure-as-Code. In: International Conference on Software Engineering Companion, 39., 24 ago. 2017. **Anais...** Buenos Aires: IEEEExplore, 2017. Disponível em: <<https://ieeexplore.ieee.org/abstract/document/7965401>>. Acesso em: 02 out. 2019.

BEYER, Betsy; JONES, Chris; PETOFF, Jennifer; MURPHY, Niall R. **Engenharia de Confiabilidade do Google**: Como o Google administra seus sistemas de produção. São Paulo: Novatec, 2016.

BOTELHO, Joacy M.; CRUZ, Vilma A. G. **Metodologia Científica**. São Paulo: Pearson Education do Brasil, 2013.

GIL, Antonio C. **Como Elaborar Projetos de Pesquisa**. 6. ed. São Paulo: Atlas, 2002.

HEWLETT PACKARD ENTERPRISE. **O que é Infraestrutura como Código?**. 2019. Disponível em: <<https://www.hpe.com/br/pt/what-is/infrastructure-as-code.html>>. Acesso em: 12 de out. 2019.

HUMBLE, Jez; FARLEY, David. **Continuous Delivery**: Reliable software releases through build, test, and deployment automation. Boston: Pearson Education, 2010.

HÜTTERMANN, Michael. **DevOps for Developers**: Integrate development and operations, the agile way. New York: Apress, 2012.

INSTRUCT. **Guia definitivo de Infraestrutura Ágil**. 2017. Disponível em: <<https://instruct.com.br/wp-content/uploads/2017/05/guia-completo-infra-agil.pdf>>. Acesso em: 15 out. 2019.

KIM, Gene; HUMBLE, Jez; DEBOIS, Patrick; WILLIS, John. **Manual de DevOps: Como obter agilidade, confiabilidade e segurança em organizações tecnológicas**. Rio de Janeiro: Alta Books, 2018.

MARCONI, Marina A.; LAKATOS, Eva M. **Fundamentos de metodologia científica**. 5. ed. São Paulo: Atlas, 2003.

NELSON-SMITH, Stephen. **Test-Driven Infrastructure with Chef: Bring behavior-driven development to infrastructure as code**. Sebastopol: O'Reilly Media, 2013.

SATO, Danilo. **DevOps: Na prática: Entrega de software confiável e automatizada**. São Paulo: Casa do Código, 2014.

SHARMA, Sanjeev. **DevOps For Dummies: IBM Limited Edition**. New Jersey: John Wiley & Sons, 2014.

VADAPALLI, Sricharan. **DevOps: Continuous Delivery, Integration, and Deployment with DevOps: Rapid Learning Solution**. Birmingham: Packt, 2018.

VENTAPANE, Dennis. **Cultura DevOps: entenda o que é quais os seus benefícios. Profissionais TI**, 2019. Disponível em: <<https://www.profissionaisiti.com.br/2019/06/cultura-devops-entenda-o-que-e-quais-os-seus-beneficios/>>. Acesso em: 06 out. 2019.