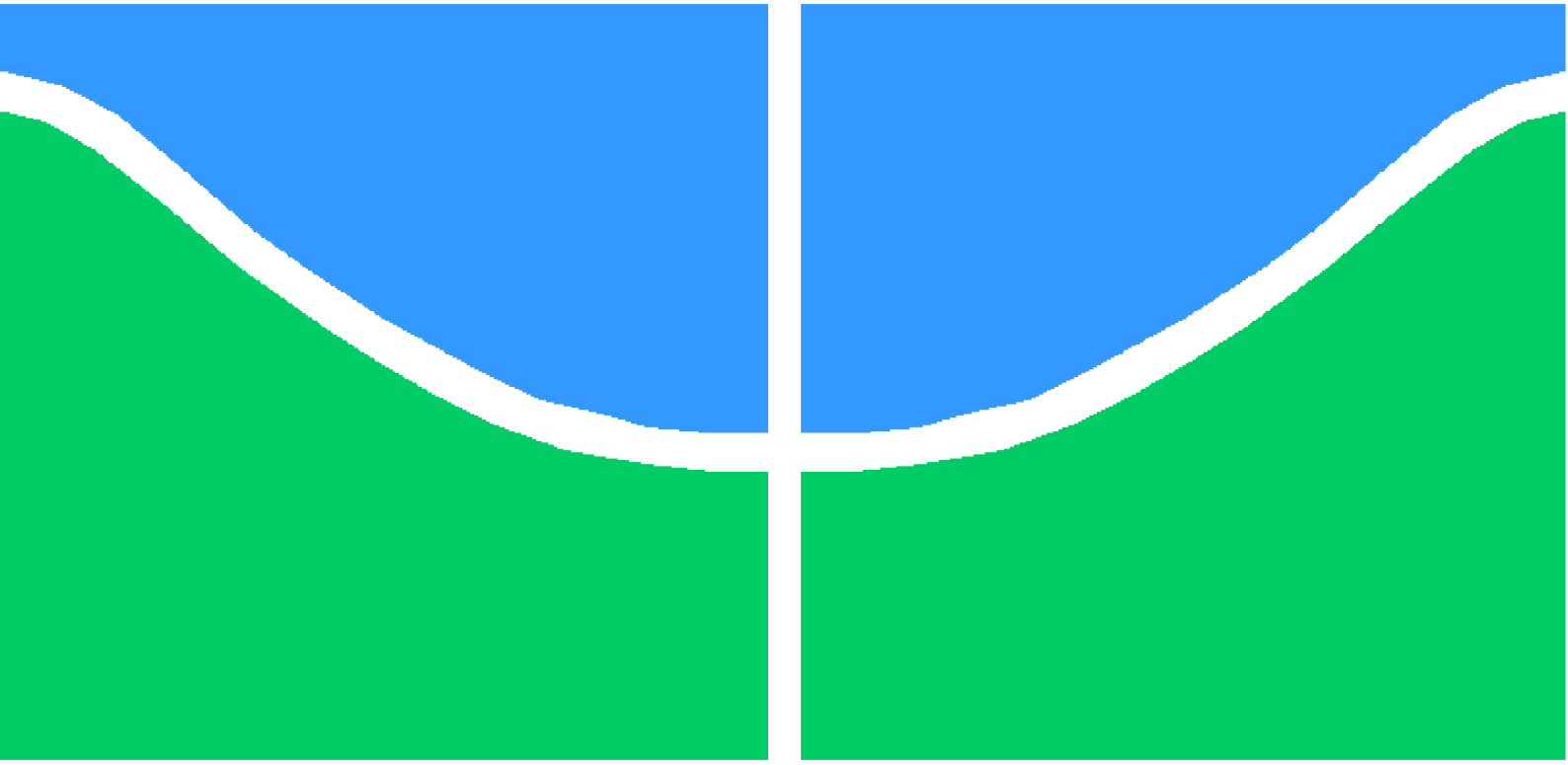


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Conceitos Práticos de IaC para Engenheiros de Software

Autor: Filipe Coelho Hilário Barcelos
Igor Araújo de Sousa

Orientador: Dra. Carla Rocha

Brasília, DF

2021

Filipe Coelho Hilário Barcelos
Igor Araújo de Sousa

Conceitos Práticos de IaC para Engenheiros de Software

Monografia submetida ao curso de graduação em (Engenharia de Software) da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em (Engenharia de Software).

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Dra. Carla Rocha

Brasília, DF
2021

Filipe Coelho Hilário Barcelos
Igor Araújo de Sousa
Conceitos Práticos de IaC para Engenheiros de Software/ Filipe Coelho Hilário Barcelos
Igor Araújo de Sousa . – Brasília, DF, 2021-
52 p. : il. (algumas color.) ; 30 cm.

Orientador: Dra. Carla Rocha

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2021.

1. Palavra-chave01. 2. Palavra-chave02. I. Dra. Carla Rocha. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Conceitos Práticos de IaC para Engenheiros de Software

CDU 02:141:005.6

Resumo

Infrastructure as Code (*IaC*) consiste, de forma resumida, na utilização de uma linguagem de codificação descritiva de alto nível para automatizar o provisionamento de uma infraestrutura em Tecnologia da Informação (TI). Pensando nisso, esse trabalho se apresenta no modelo de pesquisa-ação com objetivo de apresentar uma forma de suprir a necessidade que engenheiros de *software* de nível júnior possuem de aprender conceitos e práticas da cultura *DevOps* e, mais especificamente, *IaC*. Para atingir a meta, foi realizada a criação de uma base de conhecimentos sobre o processo de configuração de uma infraestrutura de TI para hospedagem de aplicações através de um tutorial prático.

Palavras-chave: *Infrastructure as Code. DevOps. Deploy. Clouds.*

Abstract

Infrastructure as Code (IaC) is, in short, the use of a high-level descriptive coding language to automate the provisioning of Information Technology (IT) infrastructure. Come to think of it, this work is presented in the action research model with the objective of feed the need that junior-level software engineers have to learn concepts and practices from DevOps culture and, more specifically, IaC. To achieve the goal, a knowledge base was created on the process of configuring an IT infrastructure for hosting applications through a practical tutorial.

Key-words: Infrastructure as Code. DevOps. Deploy. Clouds.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

1.1 Problema de Pesquisa

Para o desenvolvimento de *software* no contexto atual é demandado dos desenvolvedores obterem conhecimentos em relação à configuração de ambientes de desenvolvimento e um entendimento sobre o processo de implantação, conhecido em inglês como *deploy*, do código do sistema nos ambientes de desenvolvimento, homologação e produção. Essa necessidade se dá pelo fato de que houve um avanço na forma de hospedagem de sistemas, desde servidores próprios das organizações, até a contratação de serviços de armazenamento em nuvem, as chamadas *clouds*, de terceiros.

A evolução nas práticas de *deploy* utilizadas ocorre devido ao avanço da cultura *DevOps* nas organizações. *DevOps* surgiu como um movimento cultural no qual os time de desenvolvimento, teste e operações colaboram para entregar *software* de maneira contínua e efetiva e se mostrou como uma boa oportunidade para as organizações que adotam ganharem mercado em relação aos concorrentes e construírem aplicações melhores (SONI, 2016). O objetivo principal é reduzir o tempo entre todo o processo de desenvolvimento, teste e *deploy*, além de obter um *feedback*/monitoramento da aplicação em ambiente de produção.

No contexto de *DevOps* existe a utilização do conceito de Infraestrutura como Código ou, em inglês, *Infrastructure as Code* (*IaC*), que consiste na escrita de código para descrever as especificações referentes ao processo de *deploy* de uma aplicação para utilização em conjunto com *containers* orquestrados em infraestruturas como as de *clouds* (ARTAC et al., 2017). Assim, *IaC* promove a utilização de práticas e técnicas de desenvolvimento de *software* por meio da criação de *scripts* e automação de código de configuração para incluir as dependências e parâmetros necessários para o provisionamento e manutenção de uma determinada infraestrutura de TI.

Essa forma de organização possibilita a descentralização de responsabilidade da configuração e gerenciamento da infraestrutura necessária, no qual é possível remover a dependência de um operador específico e permitir que parte do processo de *deploy* seja gerido por desenvolvedores utilizando-se de conhecimentos e técnicas de *IaC* ou até mesmo não precisar de uma pessoa específica para esse gerenciamento. Além disso, o desenvolvimento se beneficia da automatização gerada ao possibilitar que os desenvolvedores não precisem gastar tanto tempo em gestão e suporte de diferentes ambientes configurados manualmente e individualmente.

1.2 Justificativa

O desenvolvimento deste trabalho visa responder a seguinte questão: *Como introduzir conceitos práticos de IaC para engenheiros de software de nível júnior?*

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo deste trabalho é a criação de uma base de conhecimentos sobre o processo de configuração de infraestrutura para armazenamento de aplicações de *software* por meio de um tutorial prático com base nos conceitos e princípios da cultura *DevOps* com foco em *IaC*.

1.3.2 Objetivo Específicos

Para atingir o objetivo geral, foram definidos os seguintes objetivos específicos:

- Identificar práticas mais atuais da cultura *DevOps* e *IaC*.
- Apresentar conceitos e técnicas de gerenciamento e configuração de infraestrutura por meio de *IaC* em exemplos simples e práticos.
- Criar um tutorial prático para a iniciação de desenvolvedores no uso de ferramentas do ecossistema de *IaC*.

1.4 Descrição dos Capítulos

Este documento está dividido nas seguintes seções:

- **Fundamentação Teórica**, que aborda parte do contexto histórico, conceitos e ferramentas relacionados à cultura *DevOps* e *IaC*.
- **Proposta**, onde é fornecido o detalhamento da proposta de trabalho de realização de um tutorial com foco em *IaC*.
- **Tutorial**, onde são demonstrados o planejamento, execução e resultados obtidos após a implementação e validação do tutorial realizado.
- **Conclusão**, que demonstra o que foi alcançado com a execução do trabalho em relação aos objetivos previamente definidos.

2 Fundamentação Teórica

2.1 *DevOps*

Organizações que desenvolvem soluções intensivas em *software* com caráter inovador ou organizações em ambientes com restrições significativas de *time-to-market*, ou seja, o tempo gasto entre o desenvolvimento do programa elaborado e o momento de anunciá-lo aos clientes, enfrentam o desafio de se adaptar constantemente às imprevisíveis mudanças para que possam alcançar seus objetivos de negócio. Nesse cenário, essas organizações precisam adotar práticas durante o ciclo de vida do *software* que garantam a entrega rápida e de qualidade das frequentes demandas de seus clientes.

DevOps se trata de um esforço colaborativo e multidisciplinar dentro de uma organização para automatizar a entrega contínua de novas versões de *software* com garantia de precisão e confiabilidade (LEITE et al., 2019). Além dessa definição acadêmica é possível entender *DevOps* como uma profissão remunerada no cenário atual do mercado de Tecnologia da Informação.

DevOps pode ser compreendido como uma evolução do manifesto ágil, propondo incrementar um conjunto de práticas ágeis para proporcionar entrega iterativa de *software* em curtos ciclos com efetividade (LEITE et al., 2019). Sendo assim, é uma evolução que possibilita a entrega de código em maior quantidade mantendo a estabilidade necessária.

De acordo com (RAJKUMAR et al., 2016) os Indicadores de Performance Chave (IPC) da aplicação de *DevOps* não se restringem à entrega contínua e redução do tempo de falha. *DevOps* tem a finalidade de alinhar os objetivos dos times de Desenvolvimento e Operações para a obtenção de resultados compartilhados com foco em confiabilidade. Entregas mais rápidas possibilitam o aumento da receita e fidelidade dos clientes.

Em relação ao desempenho, as métricas são estabelecidas levando em consideração o *deploy*, que é o termo em inglês para a fase de implantação do sistema no ambiente de produção, e o termo *commit*, que se refere a um comando de ferramentas de controle de versão de código, como o Git, onde ocorre o salvamento do estado atual do que está sendo desenvolvido. Assim, a performance de entrega é uma combinação de três métricas, que são: a frequência de *deploys*, tempo entre o *commit* ao repositório e o *deploy*, além do tempo médio de recuperação (FORSGREN; HUMBLE; KIM, 2018). Para viabilizar a implantação de *DevOps* em uma organização (RAJKUMAR et al., 2016) diz que é importante haver um conjunto correto de ferramentas utilizadas aliado a um fluxo de trabalho automatizado. Para que isso ocorre algumas estratégias a serem seguidas são:

- Definir IPCs para reconhecer padrões de processos críticos;
- Segurança automatizada e *pipelines* de *deploy*;
- Estratégias de minimização de esforço manual em testes possibilitando o desenvolvimento;
- Plano de otimização para tolerância a falhas repentinas e sistema incremental e *software* escalável.

Além disso, (RAJKUMAR et al., 2016) propõe a seguinte arquitetura de entrega contínua usando *DevOps*:

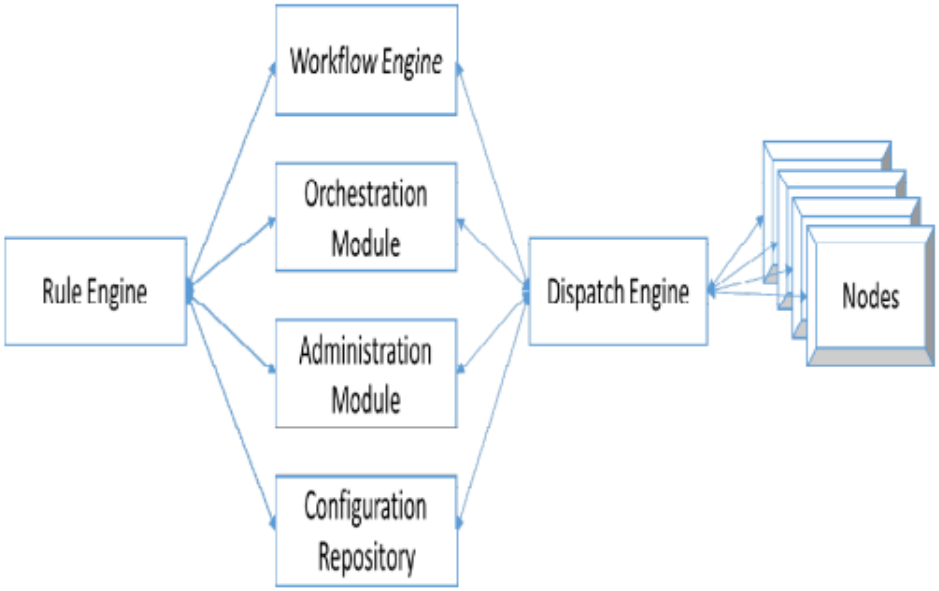


Figura 1 – Arquitetura de Entrega Contínua *DevOps*, por (RAJKUMAR et al., 2016)

Essa arquitetura foi planejada pensando em atingir velocidade e estabilidade durante os processos de implementação e *deploy* de *software*. Para isso, ela se divide nos seguintes componentes:

- **Mecanismo de regras (*Rule Engine*)** - Governa cada um dos subcomponentes definindo regras para: restringir o fluxo de processos do mecanismo de *workflow* para vários produtos e configurações de ambiente; orquestração de trabalhos através de múltiplas camadas; políticas de acesso e administração de direitos para repositório de configuração e entradas de código; e, por fim, direitos e autorização para o mecanismo de despacho.
- **Mecanismo de *Workflow* (*Workflow Engine*)** - Permite a sequência do processo para gerenciar durante o *deploy* e configurações de ambiente para diferentes produtos e camadas.

- **Orquestração e Módulo de Administração (*Orchestration and Administration Module*)** - O Módulo de orquestração permitirá a execução paralela de diferentes camadas remotas e o Módulo de Administração irá permitir usuários a monitorarem e controlarem atividades através de cada módulo.

- **Repositório de Configuração (*Configuration Repository*)**: Armazenará todos os arquivos referentes à configuração realizada.

- **Mecanismo de Despacho (*Dispatch Engine*)**: Permitirá que as tarefas sejam executadas corretamente em cada camada, além da possibilidade de *deploy* específico em uma camada.

De acordo com (AMAZON, 2021b), as principais práticas de *DevOps* são:

- Integração contínua, conhecida em inglês como *Continuous Integration* (CI)
- Entrega contínua, conhecida em inglês como *Continuous Deploy* (CD)
- Microsserviços
- *IaC*
- Monitoramento e registro em *log*
- Comunicação e Colaboração

2.2 Containers

Antes do advento da cultura *DevOps*, grande parte da função dos profissionais encarregados da infraestrutura era provisionar máquinas virtuais que possuíssem configurações semelhantes para que, assim, os testes efetuados no ambiente de homologação fossem equivalentes aos de produção. No entanto, nem sempre era possível obter essa igualdade nos ambientes, de modo que as divergências eram refletidas por meio de instabilidades em produção.

O *container* é uma forma de isolar os processos do Sistema Operacional (SO) (DOCKER, 2021c). Ele existe com o foco em empacotar as aplicações e criar um ambiente homogêneo. A grande vantagem da utilização de *containers* é, principalmente, o tamanho das imagens, pois quando comparado às máquinas virtuais, que normalmente necessitam da instalação de um SO, verifica-se que o *container* compartilha o *Kernel* da máquina que está sendo executado. Isso fica visível quando comparamos as aplicações contêinerizadas e as aplicações que utilizam máquinas virtuais como mostra a Figura 4.2.1.

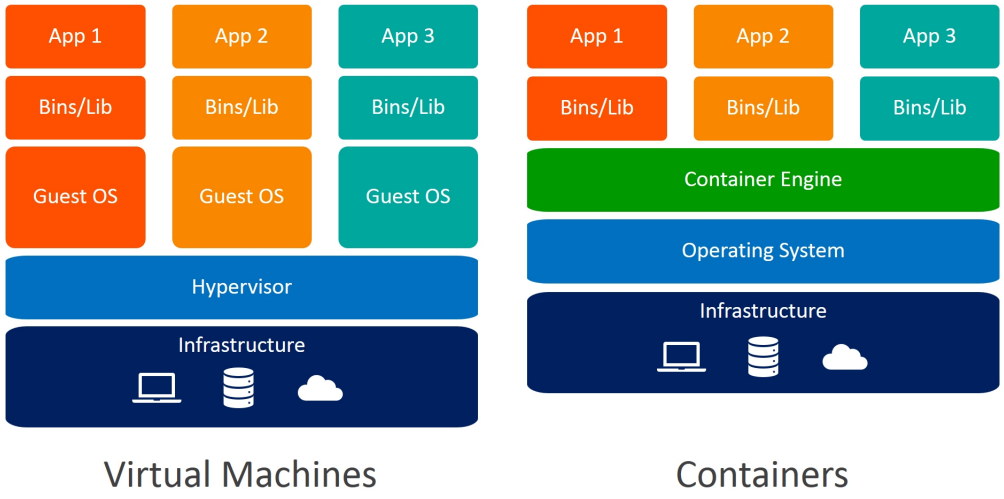


Figura 2 – Comparação entre *Containers* e *Virtual Machines* (Máquinas Virtuais) (ORACLE, 2021)

Como o *container* compartilha o *Kernel*, as únicas estruturas de dados que estão dentro dele são as bibliotecas e executáveis fazendo, assim, com que o *container* tenha somente as dependências necessárias.

2.3 Armazenamento em Nuvem (Clouds)

Os serviços de armazenamento em nuvem (*clouds*) são provedores de recursos de computação fornecidos sob demanda por meio da *internet* e com definição de preço conforme o uso (AMAZON, 2021a). Assim, *clouds* fornecem capacidade computacional, armazenamento e banco de dados removendo, assim, a necessidade de uso de *datacenters* e servidores físicos próprios pelas companhias.

As *clouds* foram importantes para o crescimento da cultura *DevOps*. Era muito comum empresas terem seus próprios *datacenters* com uma equipe dedicada somente para a manutenção e gerenciamento de servidores e infraestrutura no geral. No entanto, o custo ficava muito alto para provisionar e gerenciar toda essa estrutura e por isso as *clouds* foram ganhando força e hoje costumam ser as escolhas de grandes empresas.

Em relação ao funcionamento, *clouds* oferecem um serviço onde o responsável precisa somente definir os requisitos das máquinas e gerenciar, não sendo necessário se preocupar com picos de energia, instabilidade de rede, nem mesmo com manutenção, pois todas essas questões são garantidas pelo provedor da *cloud* escolhida (REDHAT, 2021). Além do fator monetário, a escalabilidade de serviços em *cloud* é bem maior comparada a de um *datacenter* local, pois os recursos podem ser alocados dinamicamente até mesmo entre diferentes regiões, fazendo, assim, com que a aplicação tenha alta disponibilidade e uma maior escalabilidade.

2.4 Infraestrutura Como Código

2.4.1 Eras do *deploy*

A imagem abaixo ilustra a passagem entre diferentes tipos de realização do *deploy* de aplicações:

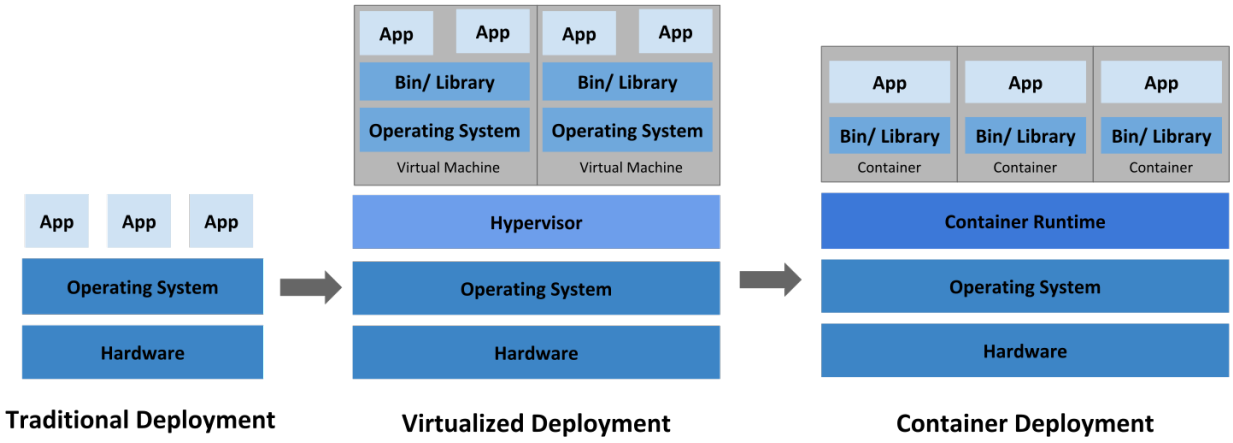


Figura 3 – Passagem entre as eras do *deploy* (KUBERNETES, 2021)

Podemos identificar a mudança entre a forma de realização do *deploy* de aplicações de *software* separando-as nas seguintes eras:

- **Era do *deploy* tradicional**: Inicialmente as organizações executavam aplicações utilizando servidores físicos próprios, o que impossibilitava o estabelecimento de limites de alocação de recursos das máquinas e ocasionava problemas de alocação dos mesmos. Assim, quando ocorria uma situação onde mais de um ambiente de produção ocupava o mesmo servidor poderia haver maior consumo de recursos por parte de uma única aplicação, o que tornava as demais menos performáticas. Para que não houvesse esse problema, a única solução era reservar um servidor por aplicação. Essa solução causava sobras de recursos por servidor, além de ser muito cara para manutenção de vários servidores físicos pelas organizações.
- **Era do *deploy* por virtualização**: Surgiu como solução para a impossibilidade de alocar múltiplas aplicações em um mesmo servidor físico com gerenciamento dos recursos utilizados. Essa solução utiliza Virtualização para que diferentes aplicações sejam isoladas em *Virtual Machines* (VMs), o que possibilita adição e atualização de aplicações, redução de custos de *hardware*, ao alocar apenas o necessário para cada VM, e obtenham, também, um maior nível de segurança, visto que em um único servidor podem haver várias VMs rodando aplicações que não são capazes de acessar informações livremente umas das outras. Nessa solução, cada VM é uma máquina completa com componentes e um SO rodando sobre um *hardware* virtualizado.

- **Era do *deploy* por meio de *containers***: Em comparação às VMs, *containers* são considerados mais leves por possibilitarem o isolamente de ambientes utilizando um único SO. Assim como as VMs, *containers* possuem propriedades de compartilhamento de CPU, memória, processamento e outros. Porém, por serem desacoplados de uma infraestrutura, eles podem ser portados em provedores de nuvem e diferentes distribuições de SOs, de modo que não demandam da organização a manutenção de uma infraestrutura de servidores própria, o que reduz custos.

2.4.2 O que é?

Ao conhecer as diferentes eras de realização do *deploy* de aplicações, é possível verificar a evolução dessa etapa no processo de entrega de *software*. Mesmo ao chegar até a utilização de *containers* para manutenção de ambientes com configurações similares ou idênticas, ainda existem questões em relação a configuração da infraestrutura em diferentes ambientes. Pensando nisso, Infraestrutura como Código ou *Infrastructure as Code (IaC)* utiliza uma linguagem de codificação descritiva de alto nível para automatizar o provisionamento de infraestrutura de TI (IBM, 2021). Essa automatização possibilita que desenvolvedores não precisem provisionar e gerenciar servidores manualmente, operar sistemas, conexões de banco de dados, armazenamento, além de outros elementos da infraestrutura sempre que precisam desenvolver, testar e realizar o *deploy* de aplicações de *software*. *IaC* também é uma prática essencial do *DevOps* sendo indispensável para um ciclo de vida de entrega de *software* em ritmo acelerado. Existem várias ferramentas que facilitam a utilização de *IaC*, como Chef, Puppet, Amazon Cloudformation, Saltstack, além de Ansible e Terraform (IBM, 2021) que são ferramentas de automatização declarativa.

De acordo com (GUERRIERO et al., 2019), *IaC* promove a gestão do conhecimento e experiência por meio de *scripts* de código de infraestrutura reutilizáveis ao invés de reservar essa tarefa para a realização manual de administradores de sistema tornando-a mais lenta, demorada, propensa a grande esforço e a erros.

A utilização de *IaC* é uma evolução surgida para resolver problemas de diferença de configuração nas *pipelines* de *deploy* dos diferentes ambientes (MICROSOFT, 2021). Sem *IaC*, as equipes precisam manter configurações individuais para cada ambiente, o que, a longo prazo, faz com que cada diferente configuração se torne muito particular e exclusiva, não podendo ser reproduzida de forma automática. Essa diferença e inconsistência entre as configurações dos ambientes pode resultar em problemas durante o *deploy*, além de dificultar a manutenção da infraestrutura, já que, neste caso, envolve processos manuais, tornando o processo mais propenso a erros.

Um importante benefício trazido pela utilização de *IaC* é a possibilidade da realização de testes de aplicações desde o início do desenvolvimento em diferentes ambientes

montados pelas equipes de *DevOps* (MICROSOFT, 2021). Esses ambientes de teste podem ser provisionados sob demanda, de forma confiável e de modo dinâmico sendo construídos quando necessário e destruídos em seguida pelas próprias plataformas de *clouds* com base nas definições de *IaC*.

A abordagem preferida na utilização de *IaC* é por meio de arquivos de definição declarativos, onde se especifica o que um determinando ambiente necessita e não necessariamente como. Não existe uma sintaxe padrão utilizada em *IaC* declarativa, de modo que as diferentes plataformas podem fornecer suporte a diferentes formatos de arquivo, como YAML, JSON e XML. Assim, a decisão de utilizar uma determinada sintaxe fica sujeita a qual plataforma será utilizada como destino da infraestrura.

2.5 Ferramentas do Ecossistema de Infraestrutura como Código

Para a realização de processos automatizados de *deploy* de aplicações existem diversas ferramentas auxiliares onde se empregam o uso de conceitos relacionados à *DevOps* e *IaC*, além do uso de *containers* portados em serviços de *clouds*. Dentre essas ferramentas, daremos destaque ao uso de Docker, Kubernetes, Terraform e Google Cloud Platform.

2.5.1 Docker

2.5.1.1 Descrição

Docker é uma plataforma para desenvolvimento, envio e execução de aplicações (DOCKER, 2021a). Essa plataforma permite separar aplicações da infraestrutura para que seja possível gerenciá-la da mesma forma que as aplicações e, com isso, entregar *software* mais rapidamente.

A plataforma Docker permite o uso de *containers* para o empacotamento e e execução de aplicações em ambientes isolados, o que possibilita a execução de múltiplos ambientes em um único *host* (DOCKER, 2021a). O objetivo do Docker é fornecer ferramentas e uma plataforma capaz de gerenciar o ciclo de vida dos *containers* de forma que os usuários:

- Desenvolvam aplicações e os devidos componentes de suporte usando *containers*;
- Usem *containers* como meio de distribuição e teste das aplicações;
- Realizem o *deploy* de aplicações no ambiente de produção, de modo que seja em um único *container* ou um serviço de orquestração de *containers*, com garantia de funcionamento igual em infraestrutras locais, *clouds* ou de forma híbrida.

2.5.1.2 Arquitetura Utilizada

A arquitetura que o Docker utiliza é baseada em cliente-servidor, ou *client-server* em inglês, onde o Docker *client* se comunica com o Docker *daemon*, que faz o trabalho pesado de construir, executar e distribuir os *containers*. A arquitetura possui os seguintes elementos:

- **Docker *daemon* (*dockerd*)**: escuta as solicitações da API do Docker e gerencia imagens, *containers*, *networks* e volumes, sendo que um *daemon* também pode se conectar com outros *daemons* para gerenciar serviços.
- **Docker *client* (*docker*)**: é a principal forma de interação dos usuários com o Docker, sendo utilizado em comando como *docker run* onde o *client* envia para o *daemon* para que ele os execute. O comando *docker* utiliza a API do Docker e esse cliente pode se comuncar com mais de um *daemon*.
- **Docker *registries***: Armazena imagens do Docker. Um *registry* que todos podem utilizar é o Docker Hub. Além disso, o Docker é configurado para buscar imagens no Docker Hub por padrão. Os comandos *docker pull* e *docker run*, por exemplo, buscam as imagens do *registry* configurado, seja ele o Docker Hub ou um privado. Da mesma forma o comando *docker push* envia a imagem mencionada para o *registry* configurado.
- **Docker *objects***: Imagens, *containers*, *networks*, volumes e *plugins*, por exemplo, são objetos utilizados pelo Docker.

Os elementos da arquitetura do Docker conversam entre si conforme a imagem abaixo:

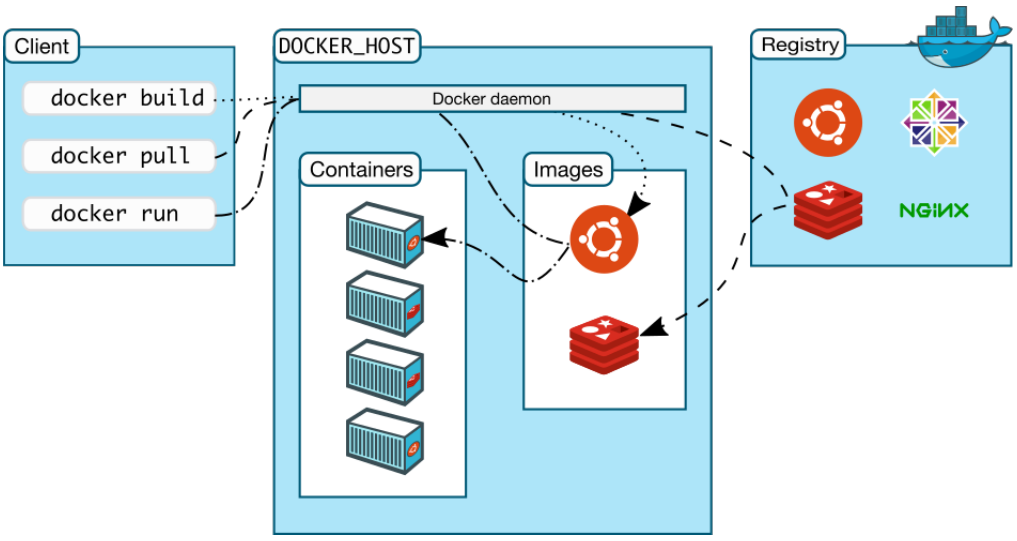


Figura 4 – Arquitetura do Docker (DOCKER, 2021a)

2.5.1.3 Dockerfile

O Dockerfile é um documento de texto que que contém todos os comandos que o usuário do Docker pode utilizar na linha de comando para montar uma imagem (DOCKER, 2021b). É possivel para o Docker construir imagens automaticamente à partir de instruções escritas em um Dockerfile. Assim, utilizando o comando *docker build* é possível criar uma *build* automatizada que executa diversos comandos em sequência na linha de comando.

2.5.1.4 Docker Compose

Docker Compose é uma ferramenta desenvolvida para facilitar a definição e compartilhamento de aplicações que utilizam múltiplos *containers* (DOCKER, 2021d). Com essa ferramenta é possível criar um arquivo YAML para definir serviços e com apenas um comando subir ou derrubar todos os serviços geridos.

A principal vantagem apresentada no uso do Compose é a possibilidade de definir toda a pilha, ou *stack*, da aplicação em um único arquivo, de modo que esse arquivo é mantido na raiz do repositório do projeto e passível de controle de versão. Assim, se torna muito fácil liberar o acesso para contribuição de outros desenvolvedores no projeto visto que basta apenas clonar o repositório e inicializar a aplicação por meio do Compose.

2.5.2 Kubernetes

2.5.2.1 Descrição

Kubernetes é uma plataforma *open-source* para gerenciamento de serviços que utilizam *containers*, o que facilita a configuração declarativa e automatização (KUBERNETES, 2021). Dessa forma, pode funcionar como um orquestrador de *containers* utilizados no processo de *deploy* de ambientes de aplicações.

2.5.2.2 Características

O papel do Kubernetes surge quando verifica-se a necessidade de gerenciar o uso de *containers* que executam uma aplicação em produção e existe a necessidade de que não haja tempo de inatividade do sistema em caso de queda ou mal funcionamento de algum deles. Caso algum *container* caia é necessário que outro com as mesmas funções inicie em seu lugar. Pensando nesse tipo de situação, o Kubernetes fornece um *framework* para executar sistemas distribuídos de forma resiliente (KUBERNETES, 2021). Com ele é possível cuidar do dimensionamento, tolerância a falhas, configurar diferentes padrões de *deploy*, entre outras funções. Dentre suas principais características, destacam-se:

- **Descobrimento de serviço e balanceamento de carga**: O kubernetes pode expor um *container* usando o DNS ou o IP dele e, se o tráfego for muito alto, é capaz de balancear a carga e distribuir o tŕafego de rede garantindo um *deploy* estável.

- **Orquestração do armazenamento**: È possível montar um sistema de armazenamento de diferentes formas, como armazenamento local, por provedores de nuvem ou outras opções.

- **Automatização de *rollouts* e *rollbacks***: É possível descrever o estado desejado para os *containers* implantados, de modo que eles mudem o estado atual para o desejado em uma taxa controlada. Assim, é possível automatizar a criação de *containers* para o *deploy* removendo os containers existentes e criando novos mantendo os mesmos recursos.

- **Empacotamento binário automático**: Pode ser fornecido um *cluster* de nós para executar tarefas que usam *containers*, assim, fornecendo a quantidade de CPU e memória RAM necessária para cada *container*. O Kubernetes é capaz de encaixar esses *containers* nos respectivos nós para garantir o melhor uso dos recursos alocados.

- **Autocorreção**: O Kuberentes pode reiniciar, substituir e encerrar *containers* que não respondem corretamente as checagens definidas. Além disso, não anuncia *containers* aos clientes antes de estarem prontos para uso.

- **Segredos e gerenciamento de configuração**: com Kubernetes é possível armazenar e gerenciar dados sensíveis, como senhas, OAuth *tokens* e chaves SSH. Também é possível realizar o *deploy* e atualizar informações secretas e configurações das aplicações sem refazer as imagens dos *containers* e sem expor os segredos na configuração.

É importante destacar que o Kubernetes não funciona como um PaaS (Plataforma Como Serviço) tradicional, ou seja, aquela que fornece um ambiente de desenvolvimento e implantação completo usando nuvem, com recursos que permitem a hospedagem desde aplicações mais simples a aplicações mais sofisticadas. Essa diferença se dá justamente pelo fato de o Kubernetes funcionar a nível de *container* e não a nível de *hardware*. O Kubernetes possui alguns recursos tradicionais de PaaS, como *deploy*, escalonamento, balanceamento de carga, além de integração com soluções de *logging*, monitoramento e alerta, porém, são opcionais proporcionando flexibilidade. Dito isso, o Kubernetes:

- não limita os tipos de aplicações suportadas, de modo que qualquer aplicação que possa ser executada em um *container* não deve ter problemas de ser executada no Kubernetes.

- não realiza *deploy* nem *build* da aplicação. Esses processos de CI/CD são determinados a parte de acordo com as preferências dos usuários.
- não fornece serviços em nível de aplicação, como *middlewares*, esteruturas de processamento de dados, bancos de dados, caches e nem sistemas de armazenamento de *clusters*. Porém, esses componentes podem ser executados pelo Kubernetes ou por aplicações executadas no Kubernetes.
- não dita soluções de *logging*, monitoramento ou alerta, mas fornece integrações para tal.
- não fornece ou exige uma linguagem específica de declaração.
- não fornece ou adota sistemas abrangentes de configuração de máquinas, manutenção, gerenciamente e nem autocorreção.
- o Kubernetes não é um mero sistema de orquestração, de modo que não importa qual o fluxo a ser seguido do início ao fim do processo, mas sim um meio de integrar diferentes processos de controle independentes e combináveis para que o fluxo definido seja seguido.

2.5.3 Terraform

O Terraform é uma ferramenta *open source* focada em *IaC* que facilita a criação, manutenção, migração e versionamento da infraestrutura (TERRAFORM, 2021). A dificuldade de se criar uma infraestrutura modular hoje é menor que no passado, levando em consideração a dificuldade da criação de *resources* idênticos em provedores diferentes. O Terraform possui muitas vantagens, de modo que a a principal é a forma como ele se integra com a maioria das plataformas provedoras de *cloud* públicas que existem no mercado, como Amazon Web Services (AWS), Google Cloud Platform e Azure.

Essa ferramenta funciona através de uma linguagem descritiva criada pela HashiCorp, que é a empresa responsável por sua manutenção, e o fato dela ser de uma empresa diferente dos maiores *players* de *cloud* do mercado é uma grande vantagem, pois faz com que a ferramenta seja agnóstica quando o assunto é infraestrutura, gerando assim uma quantidade de *resources* equivalentes entre os provedores de *cloud*.

Um dos maiores problemas de se lidar com *IaC* é que em algumas ocasiões o arquivo declarativo que gera a infraestrutura pode ter algum erro e impossibilitar o funcionamento da aplicação do código no provedor de *cloud*. No entanto, o Terraform trabalha com um esquema de planos e aplicações, fazendo, assim, com que antes de aplicar a infraestrutura é possível verificar se o código possui algum erro de implementação e se ela irá ser aplicada da forma que foi escrita.

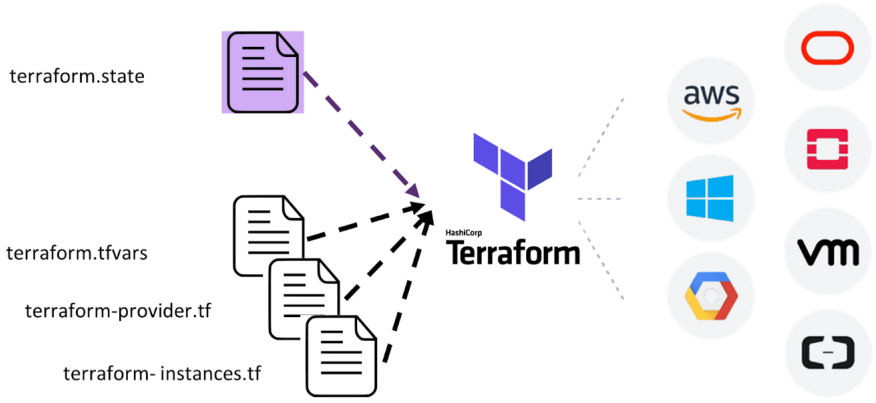


Figura 5 – Aplicação do terraform na prática (LUZ, 2021)

Como é possível verificar na Figura 5, todas as definições de autenticação, *providers* e *resources* são gerenciados pelo Terraform, o que facilita muito o encapsulamento de cada etapa de desenvolvimento da infra, pois tudo está centralizado e, assim, grande parte do trabalho de provisionamento da infraestrutura, que antes era totalmente manual, passa a ser automatizado e replicado para outros ambientes e *clouds*.

2.5.4 Google Cloud Platform

O Google Cloud Platform, ou simplesmente Google Cloud, é um provedor de *cloud* criado para facilitar a entrega de recursos de infraestrutura em *cloud* para os clientes, possibilitando, assim, que o usuário não precise de nada físico para que consiga prover uma infraestrutura. Todas as ferramentas do Google Cloud são cobradas por uso de computação em nuvem e abstraem muitos conceitos para que o usuário consiga ter toda sua infraestrutura sem necessariamente ter um time somente focado nisso. Uma grande vantagem do Google Cloud é a disponibilidade em várias partes do mundo, o que torna possível modificar a região da infraestrutura de forma fácil e rápida para diminuir a latência.

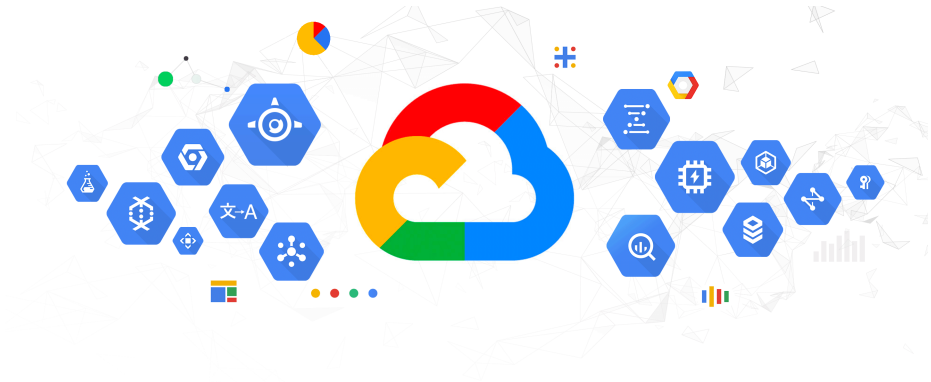


Figura 6 – Recursos disponíveis Google Cloud (GOOGLECLOUD, 2021)

Como é possível verificar na figura acima, existem vários recursos diferentes para cada contexto dentro do Google Cloud, como o Kubernetes Engine, Compute Engine, Cloud Functions e vários outros, cada um com sua respectiva finalidade.

3 Proposta

Em cursos de graduação de engenharia de *software*, além de profissionais de nível júnior no mercado de trabalho, normalmente não há capacitação para que se possa haver a compreensão de como se trabalhar com *IaC* e aplicações com *deploy* automatizado. A evolução da hospedagem de sistemas em serviços de *cloud* com o auxílio de ferramentas para gerenciamento dos parâmetros desses servidores remotos vem diminuindo a necessidade de profissionais exclusivamente dedicados à administração de aplicações em servidores próprios das organizações. Dessa forma, há o aumento da necessidade de que desenvolvedores se habilitem a não somente codificar, mas também lidar com essa configuração que demanda esses conhecimentos relativos à *IaC*.

Será realizado um tutorial prático, em formato *Hands On*, que seja capaz de guiar desenvolvedores no aprendizado de conceitos e ferramentas para habilitá-los a trabalhar com *IaC* para o gerenciamento, configuração e evolução de soluções de armazenamento de aplicações hospedadas em *clouds* próprias ou de terceiros. O tutorial realizado será dividido em várias etapas hospedadas em um repositório de uma organização na plataforma Github. Nesse capítulo iremos apresentar a metodologia da pesquisa.

3.1 Metodologia

Este trabalho se baseia no modelo de pesquisa-ação (WOHLIN; RUNESON, 2021), onde foi identificada a necessidade de uma forma em que desenvolvedores possam se adequar à atualização do mercado de *software* em relação à configuração e gerenciamento da infraestrutura utilizada. Para isso, foi idealizada uma forma de tutorial no modelo *Hands On*, onde o objetivo é que os desenvolvedores possam aprender conceitos e ferramentas referentes à *IaC* por meio de atividades práticas. Assim, o processo seguido para a realização do trabalho foi: pesquisa de conceitos e ferramentas, testes com as ferramentas, planejamento e execução do tutorial e validação com usuários.

3.1.1 Revisão Bibliográfica

Para identificar os conceitos e ferramentas utilizadas foi realizada uma revisão bibliográfica com foco de pesquisa na plataforma IEEE Xplore e nas documentações das ferramentas Docker, Terraform e Google Cloud.

3.1.2 Pesquisa de Conceitos e Ferramentas

As pesquisas para o desenvolvimento deste trabalho ocorreram com a busca de artigos científicos que corroborassem com a problemática apresentada, além de um referencial teórico amplamente baseado na documentação das ferramentas que foram introduzidas no desenvolvimento do tutorial, onde essas documentações apresentam não apenas a forma de utilizá-las, mas também, conceitos teóricos referentes aos tópicos de *DevOps*, *IaC*, *containers* e *clouds*.

Para obter conhecimento sobre as ferramentas selecionadas para o tutorial foram realizados testes práticos utilizando-as. Esses testes foram hospedados em repositórios no Github. O objetivo desses testes é avaliar qualitativamente as ferramentas, a documentação, o apoio da comunidade na resolução de problemas e a adoção de tais ferramentas pela comunidade de software. Com isso, avaliamos a viabilidade e complexidade de adoção de tais ferramentas por profissionais juniores de TI, que é o objetivo desse trabalho. A restrição é que todas as ferramentas testadas fossem *Open Source*.

Após garantir a viabilidade de adoção da ferramenta, criamos um tutorial prático, de um projeto piloto, que utiliza a(s) ferramenta(s) em um contexto real simples. Foi documentado o repositório dos conhecimentos e ferramentas ensinados no tutorial. Após a documentação do repositório ele foi preenchido com o conteúdo necessário para que os usuários possam executar o tutorial corretamente de acordo com o planejado. Todo o material produzido foi disponibilizado com licença livre em um repositório do Github.

3.1.3 Validação com Usuários

Para avaliar a qualidade do material proposto de forma qualitativa foram colhidos os *feedbacks* de usuários que executaram o tutorial com o objetivo de verificar a utilidade real para desenvolvedores e engenheiros de *software* de nível júnior. As análises das experiências dos usuários foram colhidas para apontar pontos de melhoria do tutorial e propiciar a evolução do material.

4 Tutorial

4.1 Definição do escopo

4.1.1 Público Alvo

O desenvolvimento do tutorial foi focado em engenheiros/desenvolvedores de *software* de nível júnior. Dessa forma, para um melhor aproveitamente do conteúdo disponibilizado é recomendável que quem utilize o tutorial tenha habilidades de:

- Lógica de programação;
- Codificação utilizando um *framework* em alguma linguagem de programação;
- Utilização de ferramentas de versionamento de código, mais especificamente o Git;
- Execução de comandos em linha de comando, independentemente do sistema operacional utilizado;
- Conhecimento sobre estruturas e o funcionamente de plataformas provedoras de nuvem (*clouds*).

4.1.2 Escolha das ferramentas

A escolha de cada uma das ferramentas foi baseada em critérios de maior popularidade entre a comunidade, facilidade de aprendizado e custo. Considerando as ferramentas específicas de *IaC*, as opções de busca foram restritas às mais utilizadas com base na seguinte tabela:

Tool	#	**Usage %**
Docker	26	59.0%
Ansible	23	52.2%
Vagrant	19	43.1%
Kubernetes	18	40.9%
Chef	16	36.3%
Terraform	15	34.1%
Puppet	13	29.5%
Apache Brooklyn	9	20.0%
Packer	9	20.0%
CloudFormation	9	20.0%
TOSCA	8	18.2%
Salt	6	13.6%
Shell scripts	4	09.0%
Cloudify	3	06.8%
Octopus Deploy	1	02.3%
Azure DevOps	1	02.3%

Tabela 1 – Comparação de procura sobre as ferramentas na pesquisa de (GUERRIERO et al., 2019)

Em relação às áreas de versionamento de código e provedores de nuvem, como não foi encontrado um artigo com um estudo sobre as ferramentas mais utilizadas, as opções foram restritas às 3 plataformas/ferramentas com maior popularidade com base nos dados do (STACKOVERFLOW, 2021).

4.1.3 Adoção de ferramentas

A adoção das ferramentas utilizadas foi feita com uma procura das 3 ferramentas mais populares no (STACKOVERFLOW, 2021), caso existissem, de cada categoria. No caso de ferramentas de *IaC* houve uma pré-seleção com base em 4.1.2. A partir das ferramentas encontradas, foi feita a seleção final de acordo com o seguinte processo:

- Verificação da adoção da comunidade em relação à ferramenta;
- Verificação dos critérios específicos para cada categoria de ferramenta, que, no geral, são: ser de Código Aberto (*Open Source*), acessibilidade para usuários sem conta, documentação explicativa, incentivo de créditos para aprendizado e suporte para o Google Cloud. Lembrando que esses são todos os critérios, mas não necessariamente foram utilizados em todas as categorias de ferramentas.

De acordo com esse processo, foi escolhida, para cada categoria desejada, a ferramenta que obteve os melhores resultados considerando todas as etapas de verificação.

Antes de desenvolvermos o projeto foi necessário entender, mesmo que teoricamente, como cada uma das ferramentas funcionam e, para isso, verificamos a documentação de cada ferramenta para entender minimamente sua aplicação e funcionamento.

4.1.4 Plataforma de versionamento de código

Existem muitas plataformas que utilizam Git para versionamento de código e, dentre elas, temos GitHub, Gitlab e Bitbucket. Para esse tipo de ferramenta, foi verificada a popularidade das plataformas no (STACKOVERFLOW, 2021) para pré-seleção. Assim, é possível verificar a maior adesão ao Github, com base na porcentagem de perguntas realizadas na plataforma, como mostrado na figura a seguir:

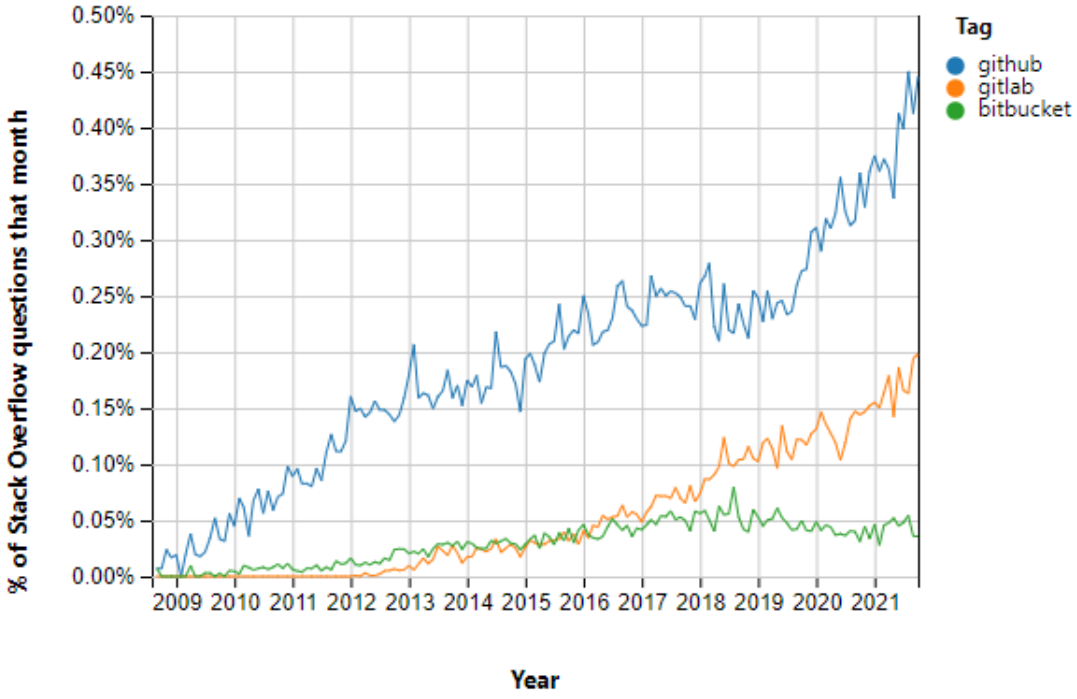


Figura 7 – Comparação de procura sobre as ferramentas de versionamento de código no (STACKOVERFLOW, 2021)

A comparação entre as plataformas pré-selecionadas pode ser vista na tabela a seguir:

Critérios	**Github**	**Gitlab**	**Bitbucket**
Maior adoção da comunidade	x	-	-
Open Source	-	x	-
Acessibilidade para usuários sem conta	x	-	-
Documentação explicativa	x	x	-

Tabela 2 – Comparação das plataformas de versionamento de código em relação aos critérios adotados

Com base nos critérios adotados, para o armazenamento do repositório foi escolhido o GitHub, pois, apesar de não ser *Open Source*, possui maior adoção da comunidade e maior acessibilidade, já que não é necessário possuir uma conta para acessar a plataforma.

4.1.5 Provedor de Nuvem (Cloud)

Assim como para plataformas de versionamento de código, para esse tipo de ferramenta foi verificada a popularidade com base na procura de ferramentas de *cloud* no (STACKOVERFLOW, 2021) para pré-seleção. Assim, é possível verificar a maior adesão à AWS e a Azure do que ao Google Cloud Platform, com base na porcentagem de perguntas realizadas na plataforma, como mostrado na figura a seguir:

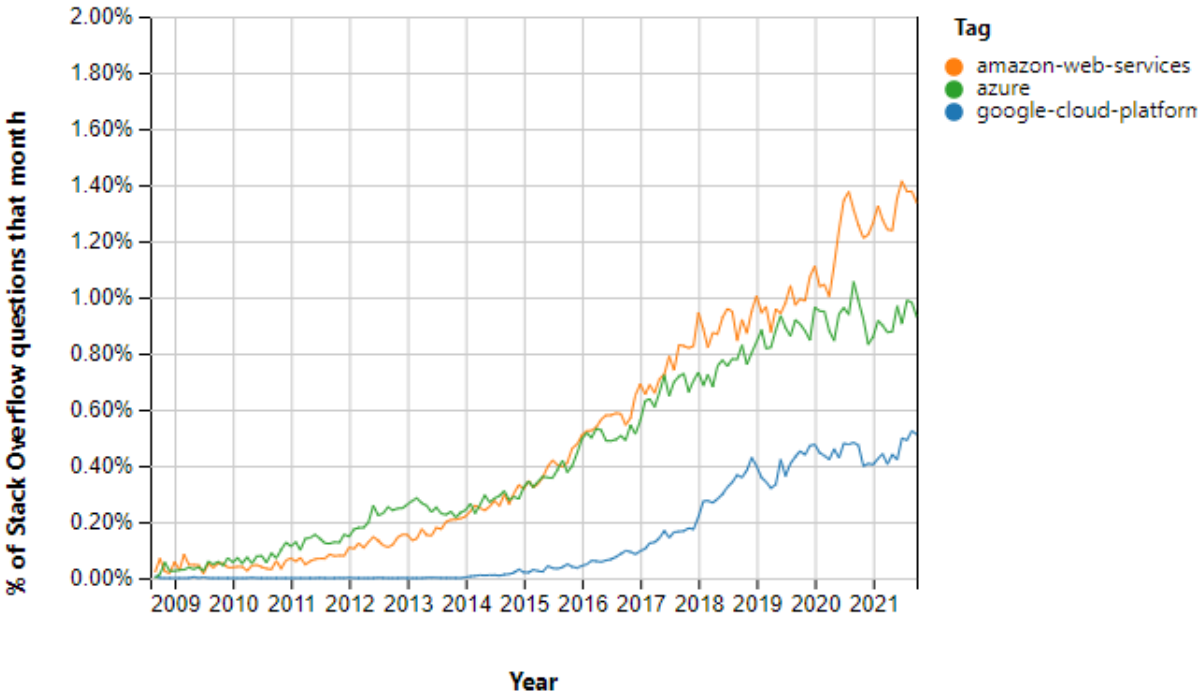


Figura 8 – Comparação de procura sobre as ferramentas de *cloud* no (STACKOVERFLOW, 2021)

A comparação entre as ferramentas pré-selecionadas pode ser vista na tabela a seguir:

Critérios	**AWS**	**Azure**	**Google-Cloud**
Maior adoção da comunidade	x	-	-
Incentivo de créditos para aprendizado	-	-	x
Documentação explicativa	x	x	x

Tabela 3 – Comparação das ferramentas de *cloud* em relação aos critérios adotados

Um dos critérios mais pertinentes para a escolha de qual ferramenta utilizar é o custo. Assim, apesar da maior adesão da comunidade às outras ferramentas, um aspecto mais favorável ao Google Cloud é a facilidade de utilização por meio de créditos oferecidos, já que é esperado que os desenvolvedores que queremos atingir não tenham muitos recursos financeiros para investir em aprendizado de forma expressiva. O Google Cloud oferece um crédito de 300 dólares, o que facilita na criação de projetos para estudo. Considerando os critérios apresentados, o Google Cloud acabou sendo a opção considerada mais viável para o aprendizado no tutorial, já que também possui uma documentação tão boa quanto

as outras plataformas.

4.1.6 Ferramenta utilizada para *IaC*

Dentre as ferramentas verificadas em 4.1.2, aquelas específicas para *IaC* são: Ansible, Chef, Terraform, Puppet e CloudFormation.

Em relação à procura no (STACKOVERFLOW, 2021) considerando essas ferramentas é possível verificar a maior adesão ao Terraform, com base na porcentagem de perguntas realizadas na plataforma, como mostrado na figura a seguir:

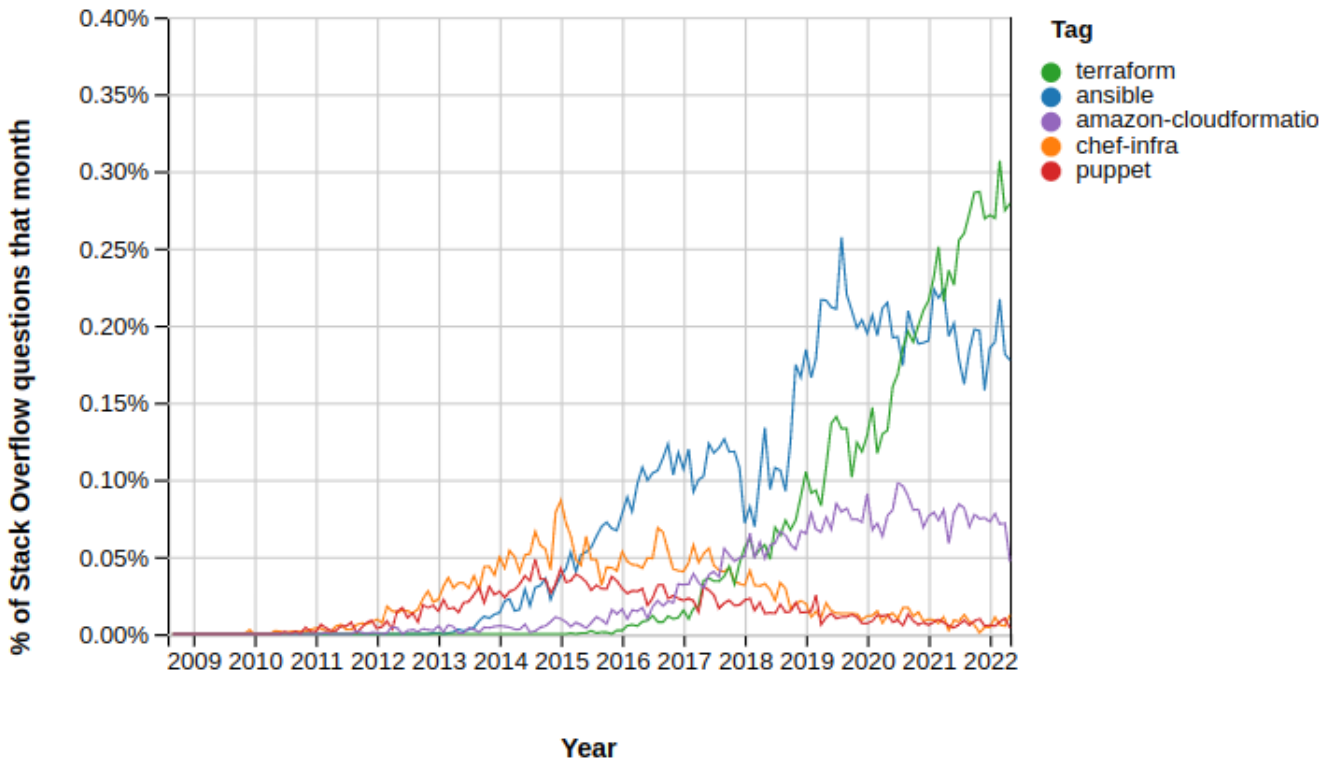


Figura 9 – Comparação de procura sobre as ferramentas de *IaC* no (STACKOVERFLOW, 2021)

A comparação entre as 3 ferramentas com maior adesão da comunidade dentre as pŕe-selecionadas pode ser vista na tabela a seguir:

Critérios	**Terraform**	**Ansible**	**Amazon Cloudformation**
Adoção da Comunidade	x	-	-
Suporte para google Cloud	x	x	-
Open Source	x	x	-

Tabela 4 – Comparação das ferramentas de *IaC* em relação aos critérios adotados

O Amazon Cloudformation somente pode ser utilizado dentro do contexto da AWS, o que não é compatível com a escolha de provedor de nuvem feita, que é o Google Cloud. Considerando os critérios avaliados, como a adoção pela comunidade e a compatibilidade com diversos serviços de *cloud*, o Terraform foi selecionado para o tutorial.

Como foi escolhido o Google Cloud por questões de custo e documentação para os desenvolvedores que vão começar, preferimos utilizar o Terraform, pelo grande grau de adoção em empresas, pela comunidade e por ser agnóstico funcionando em qualquer ambiente de *cloud*. O Terraform dá a possibilidade de criarmos *resources* de acordo com cada *cloud*, sendo, assim, possível criar um Kubernetes Engine e um *container registry* dentro do Google Cloud. O fato de o Terraform ser uma ferramenta agnóstica facilita a migração de uma infraestrutura para outra, pois é possível utilizá-lo em várias plataformas de *cloud* diferentes com pequenas mudanças do código. Um fator muito importante é o grande engajamento da comunidade com o Terraform e a quantidade de material que é possível encontrar no próprio site da HashiCorp.

4.2 Planejamento do Tutorial

Para que o desenvolvedor tenha o primeiro contato com as tecnologias e práticas relacionadas a área de *IaC*, é necessário demonstrar alguns conceitos e como utilizar as ferramentas selecionadas. Assim, foi planejado que o projeto seja um módulo considerando o tópico de interesse. Esse módulo correspondeu a um repositório dentro da organização do projeto no GitHub. O projeto é visto como um módulo inicial abrindo margem para criação de outros módulos no futuro.

O módulo criado é responsável por mostrar para o desenvolvedor como se provisiona uma infraestrutura usando uma ferramenta de *IaC*. Para realizá-lo é esperado que os desenvolvedores sejam capazes de criar uma conta de serviço no Google Cloud. Assim, será usado o Terraform para que se tenha tudo o que for necessário na criação de um

cluster inicial. A conta de serviço do Google Cloud será utilizada para a autenticação na plataforma e criar os *resources* necesssários. De modo geral, os conceitos que esperamos que os desenvolvedores obtenham são: o que são *resources*, *providers* e *instances*, além de boas práticas de utilização do Terraform.

Na tabela a seguir é possível visualizar o conteúdo abordado no tutorial:

Etapa	**Habilidades**
Entendendo o Terraform com *Local Files*	- Como criar *resources* no *provider* local e utilizar *local files*
Google Cloud Storage	- Como utilizar o *provider* do Google para acessar e modificar *buckets* do Google Cloud Storage
Google Compute Engine	- Como utilizar o *resource* de *compute engine* do Google Cloud
Utilização de *Clusters*	- Como criar um *cluster* de Kubernets no Google Cloud
Utilização de Módulos	- Como separar um projeto em módulos do Terraform

Tabela 5 – Etapas e conteúdos abordados no tutorial

É possível obter uma visualização de um exemplo de infraestrutura gerada utilizando Terraform e Google Cloud, conforme a seguinte imagem:

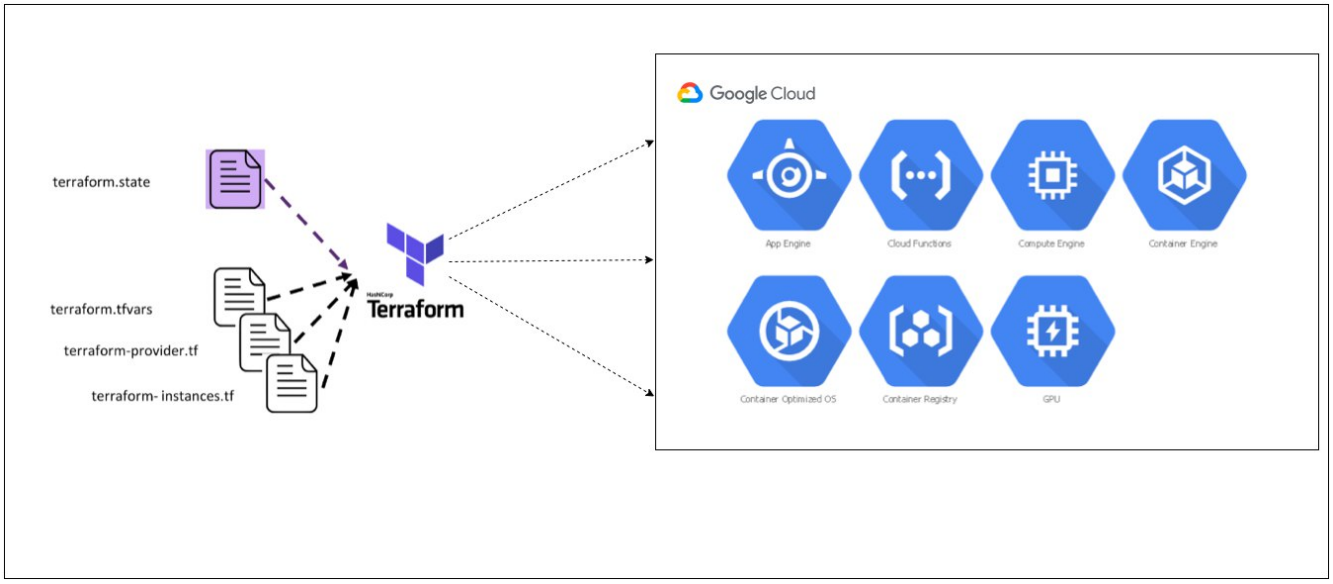


Figura 10 – Possível infraestrutura gerada utilizando Terraform e Google Cloud.

4.2.1 Criando o Tutorial

Para a criação do tutorial, conforme as etapas apresentadas na tabela 4, foram respeitadas as práticas comuns de criação de um *software* livre (GNU, 2022). Dessa forma, o resultado desejado ao utilizar essas práticas foi a criação de um tutorial apresentando código liberado para cópia, execução, estudo, distribuição e realização de melhorias conforme interesse e necessidade, de modo a contribuir com a comunidade de forma geral. Outro fator a ser considerado foi a elaboração do tutorial utilizando o modelo de aprendizagem *Hands On*, onde o intuito é que o usuário possa não só ter a teoria sobre *IaC* e Terraform, mas sim, aprender os conceitos e como aplicá-los de forma prática.

Em relação ao conteúdo efetivamente gerado, o projeto culminou em um repositório contendo o código fonte (https://github.com/DevOps-para-iniciantes/IaC) utilizado e a documentação via Github Pages (https://devops-para-iniciantes.github.io/IaC) explicando o passo a passo de como esse código foi desenvolvido e como funciona cada estrutura nele implementada. Assim, o repositório foi organizado da seguinte forma:

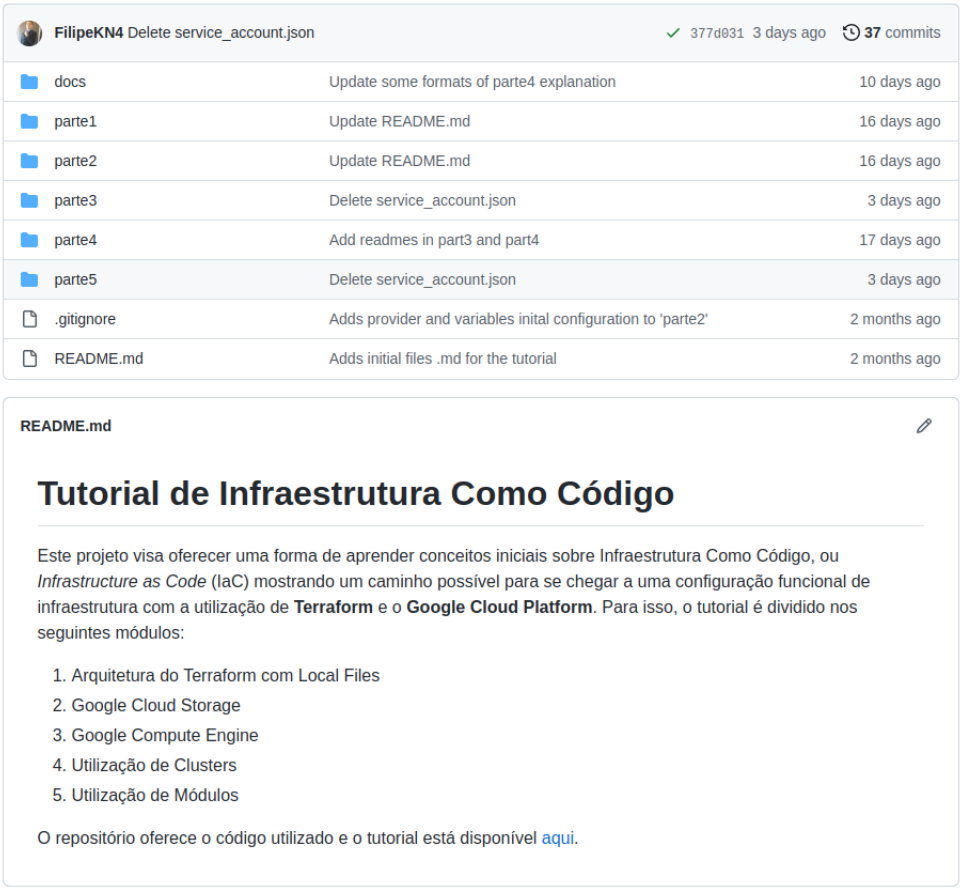
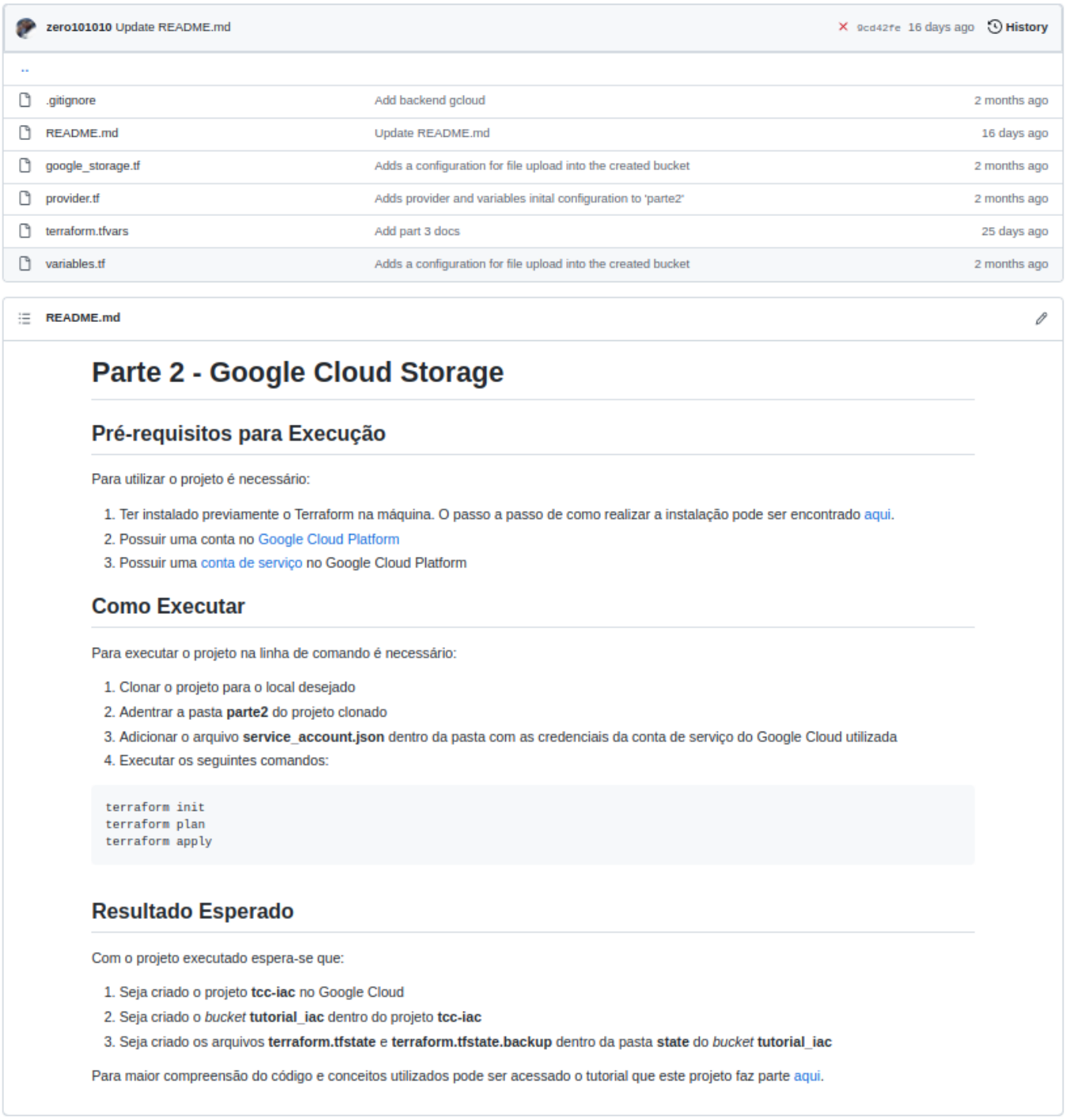


Figura 11 – Repositório *IaC* contendo o código fonte utilizado no tutorial.

O código em cada módulo segue estrutura similar à:



zero101010 Update README.md — 9cd42fe 16 days ago — History

.gitignore	Add backend gcloud	2 months ago
README.md	Update README.md	16 days ago
google_storage.tf	Adds a configuration for file upload into the created bucket	2 months ago
provider.tf	Adds provider and variables inital configuration to 'parte2'	2 months ago
terraform.tfvars	Add part 3 docs	25 days ago
variables.tf	Adds a configuration for file upload into the created bucket	2 months ago

README.md

Parte 2 - Google Cloud Storage

Pré-requisitos para Execução

Para utilizar o projeto é necessário:

1. Ter instalado previamente o Terraform na máquina. O passo a passo de como realizar a instalação pode ser encontrado aqui.
2. Possuir uma conta no Google Cloud Platform
3. Possuir uma conta de serviço no Google Cloud Platform

Como Executar

Para executar o projeto na linha de comando é necessário:

1. Clonar o projeto para o local desejado
2. Adentrar a pasta **parte2** do projeto clonado
3. Adicionar o arquivo **service_account.json** dentro da pasta com as credenciais da conta de serviço do Google Cloud utilizada
4. Executar os seguintes comandos:

```
terraform init
terraform plan
terraform apply
```

Resultado Esperado

Com o projeto executado espera-se que:

1. Seja criado o projeto **tcc-iac** no Google Cloud
2. Seja criado o *bucket* **tutorial_iac** dentro do projeto **tcc-iac**
3. Seja criado os arquivos **terraform.tfstate** e **terraform.tfstate.backup** dentro da pasta **state** do *bucket* **tutorial_iac**

Para maior compreensão do código e conceitos utilizados pode ser acessado o tutorial que este projeto faz parte aqui.

Figura 12 – Pasta da parte 2 no repositório *IaC* contendo código fonte.

Sobre a documentação, a página inicial do tutorial no Github Pages apresenta a seguinte disposição:

Figura 13 – Página inicial no Github Pages contendo a documentação do tutorial.

Em relação às páginas de partes do tutorial a estrutura é similar à:

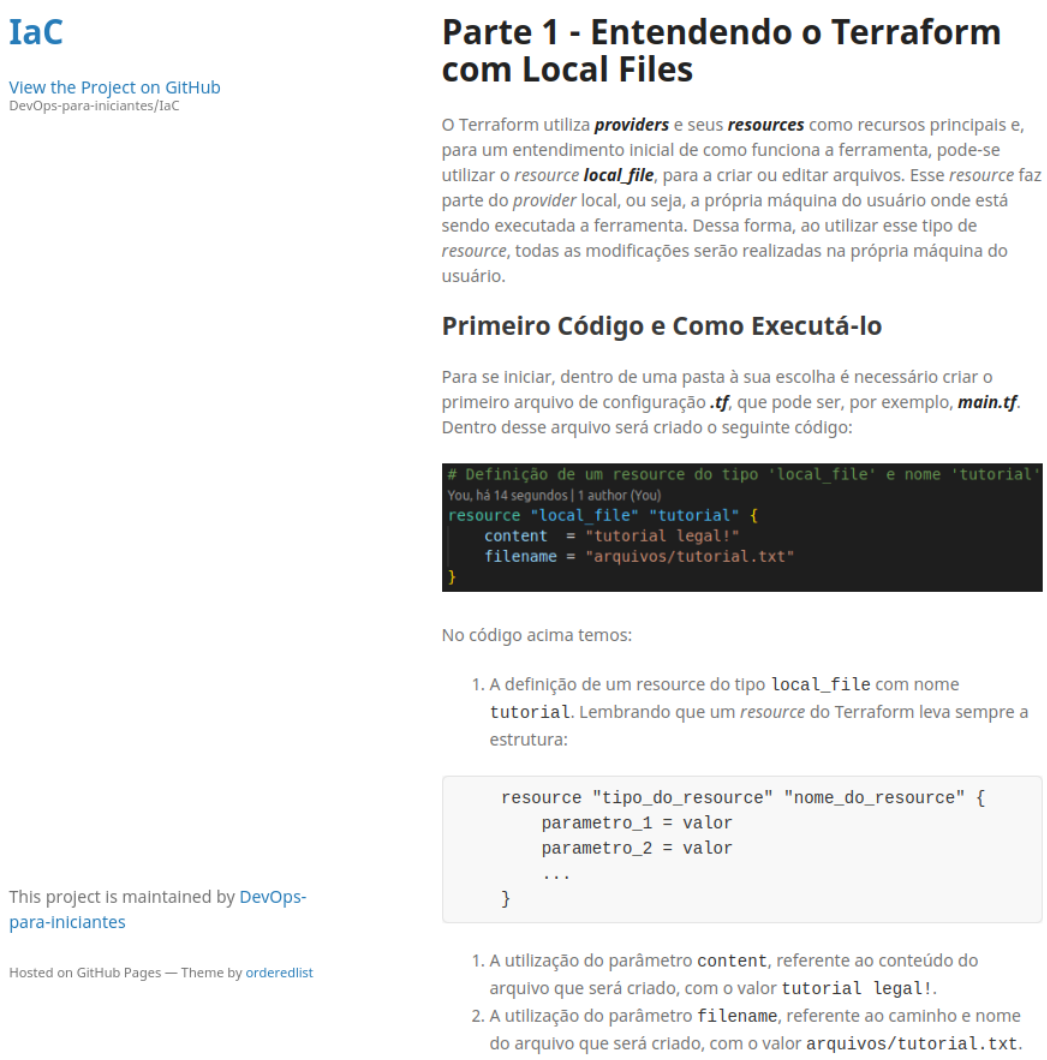


Figura 14 – Página da parte 1 no Github Pages contendo a documentação do tutorial.

4.2.2 Avaliação do Tutorial

O objetivo do tutorial é capacitar os desenvolvedores de nível júnior que acabaram de entrar no mercado e desejam adquirir conhecimento sobre uso de uma ferramenta de *IaC* e como utilizar essa ferramenta para provisionar recursos em uma plataforma provedora de nuvem. Pensando nisso, o tutorial tem como:

- **Pré requisitos**: Conhecimento em Lógica de Programação.

- **Capacitar em**: Terraform e conhecimentos básicos sobre Google Cloud Platform

Após a criação da primeira versão do tutorial completo, tanto em código armazenado no repositório e documentação de cada uma das partes planejadas, foram realizadas avaliações por três desenvolvedores de nível júnior. Cada um deles realizou a execução do

tutorial utilizando apenas o repositório e a documentação com o objetivo de verificar a qualidade do conteúdo produzido e identificar possíveis falhas. Os desenvolvedores, após o término, forneceram *feedbacks* de forma qualitativa quanto a sua percepção do trabalho em relação à documentação, parte técnica e conteúdo.

4.2.2.1 Resultados das Avaliações

Em relação as avaliações da documentação e código do tutorial os desenvolvedores forneceram *feedbacks* específicos para cada uma das 5 partes criadas no tutorial, os conceitos iniciais fornecidos e sugestões de melhoria gerais. Todos os *feedbacks* de melhorias foram aplicados de forma incremental ao tutorial, ou seja, um desenvolvedor testava, depois fazíamos uma conversa com ele para obter retorno sobre pontos positivos, negativos e propostas de soluções para os possíveis problemas. Somente após a aplicação de melhorias, de acordo com os *feedbacks* anteriores, que pedíamos para outro desenvolvedor testar. Assim, foram realizados 3 ciclos de execução do tutorial, obtenção de *feedbacks* e realização de melhorias considerando os desenvolvedores júnior que aceitaram participar. Também é importante ressaltar que, além das melhorias em relação ao conteúdo já empregado no tutorial, recebemos retornos sobre a qualidade atual para cobrir todos os aspectos possíveis do Terraform pensando em uma introdução para novos usuários. No mais, todos os desenvolvedores consenguiram concluir o tutorial sem intervenção externa.

Todos os retornos obtidos podem ser verificados nas tabelas a seguir e com as palavras utilizadas pelos usuários. Assim, em relação à parte de **Conceitos Iniciais**:

Pontos à Melhorar
- Adição de uma imagem da árvore de arquivos para cada parte no repositório
- Adição do comando *terraform destroy* ao fim de cada etapa que seja possível no tutorial
- Adicionar um *link* do repositório com o código fonte de cada parte
- Deixar claro que cada uma das etapas do tutorial é realizada dentro de uma pasta diferente
Pontos Positivos
- A leitura é simples, de fácil compreensão e com conceitos chave

Tabela 6 – *Feedbacks* sobre os **Conceitos Iniciais**

Em relação à **Parte 1 - Entendendo o Terraform com *Local Files***:

Pontos à Melhorar
- Adição do comando *terraform init* na parte 1 - Na parte de explicação da *data*, na parte 1, existe uma repetição da palavra
Pontos Positivos
- Utilizar um *localfile* facilitou o entendimento sem adicionar complexidade de um *provider* inicialmente - Os *prints* do código e o código comentado ajudam a entender qual a responsabilidade de cada *resource*

Tabela 7 – *Feedbacks* sobre a **Parte 1 - Entendendo o Terraform com *Local Files***

Em relação à **Parte 2 - Google Cloud Storage**:

Pontos à Melhorar
- Deixar claro que deve-se mudar o nome da chave baixada para *service_account.json* - Deixar claro que o projeto que passamos nas variáveis foi gerado quando criamos a conta do Google Cloud e que não vai ser gerado pelo Terraform. - Ensinar como se busca o *id* do projeto - Deixar claro para que não se destrua o *bucket* ao fim do primeiro projeto para que seja usado na parte 3 - Demonstrar erro que pode ocorrer ao usar nome de *bucket* já existente
Pontos Positivos
- Explica bem como se utiliza o *provider* do Google Cloud - Ficou claro como gerenciar o arquivo *.tfstate* dentro do *bucket*

Tabela 8 – *Feedbacks* sobre a **Parte 2 - Google Cloud Storage**

Em relação à **Parte 3 - Google Compute Engine**:

Pontos à Melhorar
- Verificar a escrita, especificamente o uso da palavra "também"em alguns casos não muito claros - Deixar claro que vai ser utilizado o mesmo *bucket* da parte 2 - Está faltando uma **}** no primeiro exemplo de código - Adicionar qual o tipo de máquina média igual está descrito no *tf.vars* - Adicionar comentários no código para explicar o que está sendo feito em cada linha
Pontos Positivos
- Apesar de ser um tutorial focado em *IaC* foi possível entender como se cria uma conta de serviço e uma máquina virtual no Google Cloud de forma fácil - As explicações são objetivas e trazem pontos que são importantes para o entendimento de como criar o recurso no Google Cloud - A forma como o tutorial foi realizado ficou muito didática, pois na parte 2, criamos um *bucket* e na parte 3 usamos esse mesmo *bucket*, o que facilitou o entendimento e o tornou gradativo

Tabela 9 – *Feedbacks* sobre a **Parte 3 - Google Compute Engine**

Em relação à **Parte 4 - Utilização de *Clusters***:

Pontos à Melhorar
- Adicionar especificamente o tipo de máquina que será utilizada nas variáveis
Pontos Positivos
- A utilização de máquinas virtuais facilitou a escolha das máquinas que seriam utilizadas - A documentação foi direta e simples e o código comentado ajudou a entender cada parte do recurso. Além disso, mesmo não tendo tanto conhecimento sobre Kubernetes ficou extremamente simples subir uma instância utilizando o Terraform.

Tabela 10 – *Feedbacks* sobre a **Parte 4 - Utilização de *Clusters***

Em relação à **Parte 5 - Utilização de Módulos**:

Pontos à Melhorar
- Deixar claro que os caminhos dos *sources* são os caminhos dos projetos contendo o código criado para as partes anteriores Explicar sobre o *warning* gerado pelo uso do *backend* na camada interna de um módulo
Pontos Positivos
- Fica muito mais simples organizar utilizando os módulos e quando se tem entendimento sobre programação é possível fazer uma associação a uma interface. Além disso, ficou muito claro como em um contexto de projeto a organização do Terraform deve ser feita. - Dentre os 3 módulos esse foi o que estava mais sintetizado e foi o mais simples. Ao entender os conceitos anteriores à parte 5, por ser uma etapa de melhoria organizacional do projeto, foi muito rápida e simples de se executar

Tabela 11 – *Feedbacks* sobre a **Parte 5 - Utilização de Módulos**

4.3 Discussão e Lições Aprendidas

Durante a elaboração do trabalho foi necessário o aprendizado de diferentes conhecimentos, sendo um dos mais importantes a capacidade de tornar o tutorial o mais compreensível e simples possível. Um grande desafio foi tentar entender os requisitos do público alvo e levantar todos os possíveis problemas, tanto na parte de configuração do ambiente quanto do conteúdo teórico. Por esse motivo acabamos optando por tentar abordar em cada exemplo de código uma explicação teórica que fizesse com que o exercício prático que fosse executado desse ênfase no detalhe do Terraform abordado.

Um outro aprendizado importante necessário foi perante a definição do escopo do trabalho, pois inicialmente o objetivo era abranger mais do que ferramentas e práticas específicas de *IaC*. Em determinado ponto percebemos que não teríamos a profundidade suficiente sobre os conteúdos com um escopo maior e, por esse motivo, criamos o projeto voltado totalmente para *IaC* visando uma maior profundidade no asssunto e um algo a mais direcionado a esse conceito.

Em relação ao conhecimento de *IaC* e, mais especificamente o Terraform, foi necessário realizar um estudo para absorção dos conceitos dessa área de conhecimento, além de tarefas práticas para entender e aplicar técnicas utilizando as estruturas dessa ferramenta, principalmente *resources* e *providers*, mais especificamente no Google Cloud Platform. Com o conhecimento obtido, pudemos aplicá-lo na criação da primeira versão do projeto e passá-lo para validação com usuários que se enquadram no público alvo

principal estabelecido. Essa etapa possibilitou a melhoria do trabalho para torná-lo mais simples e eficiente no propósito estabelecido.

Após o trabalho realizado, como resultado pudemos obter conhecimento a ser transmitido pelo tutorial criado numa área cada vez mais explorada na Engenharia de *Software*, de modo a contribuir com a construção do conhecimento de desenvolvedores e engenheiros de *software* de nível júnior que buscam aprimorar suas habilidades e se consolidarem cada vez mais no mercado de trabalho.

5 Conclusão

O objetivo estabelecido inicialmente foi apresentar conceitos de *IaC* de forma prática para engenheiros de *software* de nível júnior, de modo que eles possam alçancar um conhecimento mínimo sobre o que é essa área e como começar a trabalhar nela. Com os resultados obtidos é possível notar que a realização deste trabalho alcançou esse objetivo ao observar os *feedbacks* obtidos e o estado final do tutorial após melhorias realizadas com base neles. Os usuários que utilizaram o tutorial conseguiram executá-lo e realizar a codificação com sucesso com base nos conceitos e estruturas ensinados. Além do tutorial prático construído, foram introduzidos conceitos e explicações sobre algumas das ferramentas mais utilizadas no contexto de *IaC* atual, com foco no uso do Terraform.

Para trabalhos futuros é possível realizar uma expansão do conhecimento mostrado dentro da área de *DevOps* com a possibilidade de introduzir conceitos e ferramentas relacionados á implantação de uma aplicação na infraestrutura configurada, além da fases anteriores a esse processo, como formas de realização de CI/CD.

Referências

AMAZON. *What is cloud computing?* 2021. <https://aws.amazon.com/pt/what-is-cloud-computing/>. Accessed on Nov 2022. Citado na página 18.

AMAZON. *What is DevOps?* 2021. <https://aws.amazon.com/pt/devops/what-is-devops/#iac>. Accessed on Nov 2022. Citado na página 17.

ARTAC, M. et al. Devops: Introducing infrastructure-as-code. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2017. p. 497–498. ISBN 9781538615898. Citado na página 13.

DOCKER. *Docker overview*. 2021. <https://docs.docker.com/get-started/overview/>. Accessed on Sep 2021. Citado 3 vezes nas páginas 7, 21 e 22.

DOCKER. *Dockerfile reference*. 2021. <https://docs.docker.com/engine/reference/builder/>. Accessed on Nov 2021. Citado na página 23.

DOCKER. *Use containers to Build, Share and Run your applications*. 2021. <https://www.docker.com/resources/what-container>. Accessed on Sep 2021. Citado na página 17.

DOCKER. *Use Docker Compose*. 2021. <https://docs.docker.com/get-started/08_using_compose/>. Accessed on Oct 2021. Citado na página 23.

FORSGREN, N.; HUMBLE, J.; KIM, G. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*. 1st. ed. [S.l.]: IT Revolution Press, 2018. ISBN 1942788339. Citado na página 15.

GNU. *O que é o software livre?* 2022. <https://www.gnu.org/philosophy/free-sw.html>. Accessed on May 2022. Citado na página 39.

GOOGLECLOUD. *Visão geral do Google Cloud*. 2021. <https://cloud.google.com/docs/overview/?hl=pt-br>. Accessed on Sep 2021. Citado 2 vezes nas páginas 7 e 27.

GUERRIERO, M. et al. *Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry*. [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2019. 580–589 p. Citado 3 vezes nas páginas 9, 20 e 32.

IBM. *What is Infrastructure as Code (IaC)?* 2021. <https://www.ibm.com/cloud/learn/infrastructure-as-code>. Accessed on Oct 2021. Citado na página 20.

KUBERNETES. *What is Kubernetes?* 2021. <https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/>. Accessed on Sep 2021. Citado 3 vezes nas páginas 7, 19 e 23.

LEITE, L. et al. *A survey of DevOps concepts and challenges*. [S.l.], 2019. v. 52, n. 6. Citado na página 15.

LUZ, G. *Terraform 101: Uma breve introdução a Infra as Code*. 2021. <https://medium.com/gabriel-luz/terraform-101-uma-breve-introduç~ao-a-infra-as-code-aa32f4203f09>. Accessed on Sep 2021. Citado 2 vezes nas páginas 7 e 26.

MICROSOFT. *O que é a Infraestrutura como Código?* 2021. <https://docs.microsoft.com/pt-br/devops/deliver/what-is-infrastructure-as-code>. Accessed on Sep 2021. Citado 2 vezes nas páginas 20 e 21.

ORACLE. *Contêineres Docker e Serviços de nuvem de contêineres*. 2021. <https://www.oracle.com/br/cloud-native/container-registry/what-is-docker/>. Accessed on Sep 2021. Citado 2 vezes nas páginas 7 e 18.

RAJKUMAR, M. et al. *DevOps culture and its impact on cloud delivery and software development*. [S.l.], 2016. Citado 3 vezes nas páginas 7, 15 e 16.

REDHAT. *O que é a nuvem pública?* 2021. <https://www.redhat.com/pt-br/topics/cloud-computing/what-is-public-cloud>. Accessed on Sep 2021. Citado na página 18.

SONI, M. End to end automation on cloud with build pipeline: The case for devops in insurance industry, continuous integration, continuous testing, and continuous delivery. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2016. p. 85–89. ISBN 9781467385664. Citado na página 13.

STACKOVERFLOW. *Stackoverflow*. 2021. <https://pt.stackoverflow.com/>. Accessed on Nov 2021. Citado 6 vezes nas páginas 7, 32, 33, 34, 35 e 36.

TERRAFORM. *Introduction to Terraform*. 2021. <https://www.terraform.io/intro/index.html>. Accessed on Sep 2021. Citado na página 25.

WOHLIN, C.; RUNESON, P. *Guiding the selection of research methodology in industry–academia collaboration in software engineering*. [S.l.]: Elsevier B.V., 2021. Citado na página 29.