UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Paulo Victor da Silva Oliveira

Criação de um Ambiente de Implantação no GCP Utilizando IaC no Free Tier

Uberlândia, Brasil

2025

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Paulo Victor da Silva Oliveira

Criação de um Ambiente de Implantação no GCP Utilizando IaC no Free Tier

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Ronaldo Castro de Oliveira

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2025

Paulo Victor da Silva Oliveira

Criação de um Ambiente de Implantação no GCP Utilizando IaC no Free Tier

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Ronaldo Castro de Oliveira
Orientador

Bruno Augusto Nassif Travencolo

Mauricio Cunha Escarpinati

Uberlândia, Brasil
2025

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

O48 Oliveira, Paulo Victor da Silva, 1997-
2025 Criação de um Ambiente de Implantação no GCP Utilizando IaC no Free Tier [recurso eletrônico] : Criação de um Ambiente de Implantação no GCP Utilizando IaC no Free Tier / Paulo Victor da Silva Oliveira. - 2025.

Orientador: Ronaldo Castro de Oliveira.
Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Uberlândia, Graduação em Sistemas de Informação.
Modo de acesso: Internet.
Inclui bibliografia.
Inclui ilustrações.

1. Tecnologia da informação - Administração. I. Oliveira, Ronaldo Castro de,1968-, (Orient.). II. Universidade Federal de Uberlândia. Graduação em Sistemas de Informação. III. Título.

CDU: 658:681.3

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

Agradecimentos

Inicio este agradecimento dedicando minhas palavras a uma pessoa muito especial, que infelizmente não está mais entre nós, mas foi fundamental para que eu chegasse até aqui: minha querida avó. Sou imensamente grato por toda a ajuda financeira no início da minha caminhada, especialmente quando pagava minhas passagens de ônibus para que eu pudesse estudar. Se não fosse por sua generosidade e apoio naquele momento, certamente eu não estaria aqui hoje.

Agradeço também aos meus pais, que sempre fizeram o possível e o impossível para que eu pudesse estudar, enfrentando todas as dificuldades e se desdobrando para me proporcionar a oportunidade de realizar este sonho. Foram eles que, com palavras e atitudes, me motivaram a seguir em frente. Aos meus irmãos, que sempre estiveram presentes para me animar, mesmo nos momentos de maior exaustão. Um agradecimento especial ao meu irmão do meio, que viveu a computação de perto comigo e compartilhou dessa jornada de forma mais intensa.

À minha namorada, que foi essencial nos momentos mais difíceis, especialmente na reta final deste trabalho. Mesmo nos dias em que o cansaço parecia vencer, ela esteve ao meu lado, me motivando a continuar e a não desistir.

Aos meus amigos, que tornaram essa faculdade uma experiência inesquecível. Desde os amigos que fiz no início da graduação, no grupo CKXP, até os irmãos que conquistei na atlética, todos foram fundamentais para que essa jornada fosse mais leve, divertida e repleta de aprendizados. As vivências e experiências compartilhadas com vocês contribuíram diretamente para a pessoa que me tornei.

Não poderia deixar de agradecer à Atlética da Computação da UFU. Participar dessa instituição me moldou como pessoa, desenvolvendo em mim valores de responsabilidade, compromisso, garra e, sobretudo, a importância de pensar no bem comum. Foi um ambiente que me desafiou tanto fisicamente quanto mentalmente, e pelo qual serei eternamente grato.

Por fim, agradeço à Universidade Federal de Uberlândia, uma instituição que foi essencial para minha formação pessoal e profissional. Tenho certeza de que, sem a UFU, eu não seria o profissional que sou hoje, tampouco teria conquistado um terço do que já conquistei — e do que ainda irei conquistar. A UFU faz parte da minha história e da pessoa que me tornei, abrindo portas e me proporcionando a chance de crescer e ser melhor.

Agradeço também a todos os professores que, de forma direta ou indireta, contri-

buíram para minha formação. Levo ensinamentos de cada um, mas em especial, expresso minha gratidão ao professor Bruno Nassif, ao professor Miani e, principalmente, ao meu orientador, professor Ronaldo. Mesmo tendo assumido a orientação deste trabalho de última hora, aceitou o desafio e contribuiu para que este projeto alcançasse um resultado acima do esperado. Sou imensamente grato por sua disponibilidade e apoio.

A todos vocês, meu mais sincero e profundo agradecimento.

Resumo

Este trabalho propõe a criação de um ambiente em nuvem no Google Cloud Platform (GCP) voltado para a realização de implantações automatizadas. Por meio de práticas de Infraestrutura como Código (IaC), foi realizada a automação do provisionamento de servidores virtuais (Compute Engine) e clusters Kubernetes (GKE), com o objetivo de simplificar e acelerar o processo de configuração desses recursos. A proposta visa minimizar as dificuldades técnicas comumente enfrentadas por iniciantes no provisionamento de infraestrutura, utilizando os recursos gratuitos do Free Tier do GCP. Adicionalmente, apresenta-se um estudo de caso comparativo entre os dois ambientes, evidenciando na prática as vantagens e limitações dos deploys realizados no Compute Engine e no GKE.

Palavras-chave: DevOps, SRE, Google Cloud Platform, GCP, Free Tier, Terraform, Kubernetes, GKE, Compute Engine, Automação de Infraestrutura, Provisionamento, Infraestrutura como Código.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
CI/CD	Continuous Integration / Continuous Deployment
GCP	Google Cloud Platform
GKE	Google Kubernetes Engine
IaC	Infrastructure as Code
IP	Internet Protocol
OS	Operating System
PaaS	Platform as a Service
SRE	Site Reliability Engineering
TF	Terraform
VM	Virtual Machine

Sumário

1 Introdução

Com o crescimento do uso de ambientes em nuvem, a praticidade e a flexibilidade oferecidas por essa abordagem tornaram-se indispensáveis no cenário tecnológico atual [DigitalOcean 2023]. Grandes provedores como AWS, Google Cloud Platform (GCP) e Microsoft Azure são parte essencial das operações modernas de TI.

A utilização de soluções em nuvem, no entanto, ainda é limitada durante o processo de graduação, apesar de ser uma competência fundamental exigida no mercado de trabalho. É necessário introduzir abordagens que permitam aos estudantes explorar essas tecnologias de forma acessível, utilizando especialmente os recursos gratuitos disponíveis no Free Tier oferecido pelos principais provedores [ReviewNPrep 2022].

A aplicação de práticas de Infraestrutura como Código (IaC) possibilita criar ambientes escaláveis e automatizados para executar deploys de aplicações complexas. Isso proporciona aos estudantes uma experiência prática com tecnologias amplamente utilizadas no mercado, preparando-os para desafios profissionais pós-graduação [HashiCorp 2023].

Ferramentas como Terraform permitem automatizar o provisionamento de infraestrutura, garantindo padronização e replicabilidade. O Docker simplifica a distribuição de aplicações em ambientes isolados e consistentes, enquanto o Kubernetes fornece uma plataforma robusta para gerenciamento eficiente de containers em escala.

Além disso, a migração para a nuvem trouxe mudanças significativas na maneira como os deploys são realizados. Anteriormente, era necessário solicitar a outros times o provisionamento do ambiente e a preparação das máquinas para colocar o sistema em produção. Atualmente, tecnologias como Terraform, Docker e Kubernetes facilitam significativamente esse processo, tornando-o ágil e independente [Networks 2023].

1.1 Problema

Estudantes frequentemente enfrentam limitações de recursos computacionais em seus dispositivos pessoais para executar deploys complexos, o que afeta negativamente a produtividade e o processo de aprendizado. Além disso, a grade curricular atual do curso não contempla disciplinas que introduzam os alunos ao uso de ambientes em nuvem ou à automação de infraestrutura, resultando em um primeiro contato tardio com essas soluções e limitando o preparo técnico dos graduandos para o mercado.

1.2 Justificativa

Atualmente, é praticamente impossível discutir infraestrutura de TI sem mencionar a computação em nuvem. No entanto, a velocidade com que a tecnologia avança nem sempre é acompanhada pelas instituições de ensino superior. Isso gera uma lacuna significativa na formação de profissionais capacitados em práticas modernas de automação, como aquelas aplicadas na cultura DevOps.

A ausência de disciplinas específicas voltadas à automação de infraestrutura e ao uso de serviços em nuvem durante a graduação gera um interesse tardio pela área. Esse cenário agrava ainda mais a demanda crescente por profissionais com esse perfil. Além disso, muitos estudantes não têm conhecimento sobre os níveis gratuitos (Free Tier) oferecidos por provedores como o Google Cloud Platform, o que dificulta o contato inicial prático com essas tecnologias [Google Cloud 2024].

Segundo relatório do DevOps Institute, a escassez de profissionais qualificados em DevOps é um dos maiores desafios enfrentados pelas organizações atualmente. Cerca de 58% dos entrevistados apontaram a lacuna de habilidades como uma barreira crítica para a adoção de práticas DevOps [Institute 2023].

1.3 Objetivos

Os objetivos deste trabalho foram divididos em objetivos gerais e específicos:

1.3.1 Objetivos Gerais

O objetivo principal é demonstrar, de forma prática, a criação de um ambiente utilizando recursos da nuvem GCP (Google Cloud Platform). Para isso, adotamos como premissa a utilização de recursos gratuitos oferecidos pelo provedor. Desenvolveu-se um código automatizado para criação do ambiente utilizando práticas de Infraestrutura como Código (IaC), proporcionando uma experiência acessível e realista de computação em nuvem.

1.3.2 Objetivos Específicos

- Desenvolver um script em Python capaz de preparar automaticamente a máquina local para criação de ambiente no GCP.
- Provisionar, via automação, uma instância Compute Engine e um cluster Kubernetes Engine (GKE), respeitando as limitações do Free Tier do GCP.

- Realizar o deploy da aplicação disponível no repositório público https://github.com/rinormaloku mastery nos ambientes provisionados.
- Coletar métricas relevantes, como tempo de deploy, consumo de recursos computacionais e dificuldades encontradas em ambos ambientes.
- Realizar análise comparativa entre o deploy na Compute Engine e no GKE, destacando vantagens, limitações e recomendações de uso.
- Validar o uso de práticas de Infraestrutura como Código (IaC) para ambientes acadêmicos e experimentais, promovendo acessibilidade à computação em nuvem.

1.4 Organização do Trabalho

O trabalho está estruturado nos seguintes capítulos:

- **Capítulo 1 – Introdução:** Contextualiza o problema, a justificativa, objetivos e organização geral.
- **Capítulo 2 – Referencial Teórico:** Discute conceitos essenciais para o projeto, como computação em nuvem, IaC, Docker, Kubernetes, Terraform e Google Cloud Platform.
- **Capítulo 3 – Metodologia:** Descreve ferramentas, arquitetura proposta, processos de automação e estratégia para execução dos testes.
- **Capítulo 4 – Estudo de Caso:** Relata execução prática do projeto, apresentando deploy da aplicação nos dois ambientes e análise das métricas coletadas.
- **Capítulo 5 – Resultados:** Apresenta resultados detalhados dos testes, gráficos comparativos e análise crítica.
- **Capítulo 6 – Conclusão:** Considerações finais, limitações do desenvolvimento e sugestões para trabalhos futuros.

2 Referencial Teórico

Este capítulo apresenta os fundamentos teóricos que embasam o projeto, destacando especialmente os conceitos e tecnologias de Cloud Computing e Infraestrutura como Código (IaC). A compreensão desses temas possibilita a criação e o gerenciamento de ambientes computacionais eficientes e escaláveis, fundamentais para o escopo do trabalho.

2.1 Cloud

As primeiras ideias relacionadas ao conceito hoje denominado Cloud Computing surgiram com Joseph Carl Robnett Licklider na década de 1960, sob a designação Intergalactic Computer Network [Internet Hall of Fame 2025]. As ideias propostas por Licklider, relacionadas a redes interconectadas e compartilhamento de recursos computacionais, foram essenciais para o desenvolvimento da infraestrutura inicial que deu origem à ARPANET, considerada precursora da atual computação em nuvem.

Outro importante precursor do conceito foi John McCarthy, que apresentou ideias sobre o compartilhamento de recursos computacionais e a computação como utilidade pública, prevendo aspectos que atualmente caracterizam os principais provedores de serviços em nuvem [Talbot 2011].

Somente na década de 1990, segundo Forbes [Technologies 2011], o termo Cloud Computing foi efetivamente cunhado durante uma palestra ministrada por Ramnath Chellappa, professor da área de sistemas. Esse termo promoveu uma significativa mudança no paradigma tecnológico, da computação tradicional para a oferta de serviços através da internet.

No início dos anos 2000, surgiram os primeiros grandes provedores comerciais de serviços em nuvem, com destaque para a AWS, que lançou os serviços Simple Storage Service (S3) em 2006 [Amazon Web Services 2006], seguido pelo Elastic Compute Cloud (EC2) [Amazon Web Services 2021]. Posteriormente, o Google lançou serviços com foco em Plataforma como Serviço (PaaS) [Google Cloud 2015], e a Microsoft ingressou nesse mercado com a plataforma Azure em 2010 [Microsoft Azure 2021].

De acordo com a definição fornecida pela AWS [Amazon Web Services 2024], uma nuvem é um sistema integrado que unifica fontes, armazenamentos e infraestruturas de dados. A Microsoft, por sua vez, define a nuvem como uma rede global de servidores que realizam diferentes funções, oferecendo acesso remoto e integrado a serviços e dados [Microsoft Azure 2025]. Para o Google, a computação em nuvem é caracterizada pela disponibilidade sob demanda de recursos computacionais através da internet, eliminando

a necessidade de gerenciamento local dos recursos físicos [Google Cloud 2025].

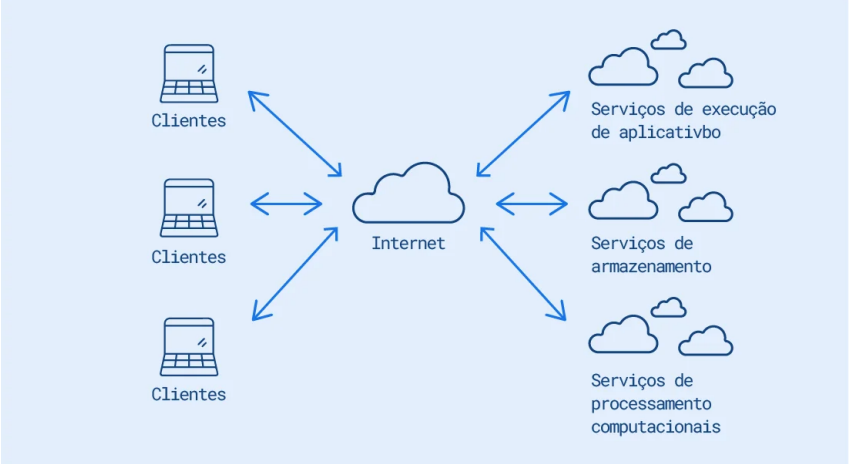

Figura 1 – Princípio básico da computação em nuvem. Fonte: Alura (2021).

Essas definições convergem na ideia central de que a computação em nuvem visa simplificar a tecnologia, oferecendo escalabilidade, flexibilidade, redução de custos e serviços avançados para melhorar a interação entre usuários e sistemas computacionais.

2.1.1 Tipos de Nuvem

De acordo com o Google [Google Cloud 2025], existem três tipos principais de nuvem:

- **Nuvem pública**: gerenciada por provedores terceiros, disponibiliza recursos compartilhados através da internet, de forma dinâmica e sob demanda.
- **Nuvem privada**: criada e gerenciada por uma única organização, oferecendo controle avançado, segurança aprimorada e gerenciamento dedicado dos recursos.
- **Nuvem híbrida**: combina elementos das nuvens pública e privada, permitindo maior flexibilidade e segurança.

2.1.2 Tipos de Serviço em Nuvem

Ainda segundo o Google [Google Cloud 2025], existem três principais modelos de serviços oferecidos em ambientes de nuvem:

- **Infrastructure as a Service (IaaS)**: fornece acesso sob demanda a infraestrutura como computação, armazenamento e redes, oferecendo alto controle dos recursos.
- **Platform as a Service (PaaS)**: fornece infraestrutura e ambiente completo para desenvolvimento de aplicações sem a necessidade de gerenciar a infraestrutura física.

- **Software as a Service (SaaS)**: oferece acesso a aplicações completas, gerenciadas e mantidas diretamente pelo provedor de serviços.

2.2 Google Cloud Platform (GCP)

O Google Cloud Platform é um dos três principais provedores mundiais de nuvem ficando atrás apenas da AWS e Azure segundo Gartner [Gartner 2025]. Segundo publicação oficial da Google ([Google Official Blog 2006]), em 2006 a empresa lançou o serviço "Google Apps for Your Domain", um conjunto de ferramentas online incluindo Gmail, Google Calendar e Google Talk. Começando num formato SAAS ou seja oferecia um software como um serviço para seus clientes. Em 7 de Abril de 2008 foi quando a google começou a comercializar seus serviços num modelo PAAS [Google App Engine Team 2008] que permitia aos desenvolvedores criar e hospedar aplicações web na infraestrutura do Google. Inicialmente, o App Engine oferecia suporte à linguagem Python e fornecia recursos como armazenamento persistente, balanceamento de carga automático e APIs para autenticação de usuários e envio de e-mails. Somente em 28 de Junho de 2012 que ai sim a Google passa a comercializar serviços numm formato IAAS [Google App Engine Team 2008]que disponibilizava máquinas virtuais Linux escaláveis, permitindo aos desenvolvedores executar cargas de trabalho intensivas e personalizadas na infraestrutura do Google.

2.2.1 Serviços Google

O Google disponibiliza uma ampla variedade de produtos dentro do seu portifilio destacando-se especialmente o Compute Engine, Google Kubernetes Engine (GKE) e a CLI Gcloud [Google Cloud 2025].

- **Compute Engine** , esse serviço permite criar e executar máquinas virtuais nos Data Centers do Google.O Compute Engine permite a escala sobre demanda do seu ambiente. Em um cenário em que sua aplicação receberá um fluxo muito alto de requisições a escala de recursos computacionais sobre demanda permitem você atender aquele fluxo de requisições e depois desmobilizar esses recursos pagando proporcionalmente por aquilo que você utiliza. [Google Cloud 2025]
- **GKE** O Google Kubernetes Engine (GKE) estrutura seus clusters em um plano de controle e por nós, que são as instâncias responsáveis por executar os serviços baseados em contêineres. O plano de controle é o componente responsável por definir o que será executado em cada nó, gerenciando tarefas como agendamento e escalabilidade. Para facilitar a operação, o GKE oferece o modo Autopilot, no qual

a complexidade da infraestrutura é abstraída, permitindo ao usuário focar apenas na implantação e execução das aplicações. [Google Cloud 2025].

- **Gcloud** Um outro recurso muito importante da Google e o CLI Gcloud, e uma biblioteca de de comandos da Google que facilita a criação e administração de recursos do Google. Permitindo gerenciar maquinas virtuais , cloud storage banco de dados, implementar script e cloud run funcitions de maneira muito mais prática. [Google Cloud 2025]

2.2.2 Free Tier

O *Free Tier* é um recurso do GCP muito importante principalmente na fase da graduação e aprendizado ele visa fornecer acesso gratuito a diversos serviços na nuvem Ele é dividido em duas categorias principais: um crédito promocional de $300 válido por 90 dias após a criação da conta e um conjunto de recursos gratuitos sempre disponíveis, que não expiram enquanto os limites de uso forem respeitados [Google Cloud 2024].

O Compute Engine oferece o uso gratuito de uma instância `e2-micro` por mês, com 30 GB de disco padrão e 1 GB de saída de rede na América do Norte. [Google Cloud 2024].

Já o Google Kubernetes Engine (GKE) disponibiliza, no nível gratuito, acesso ao controle do cluster (ou seja, ao plano de controle gerenciado) sem custo, para um cluster por conta. No entanto, os nós utilizados no cluster ainda são cobrados como instâncias do Compute Engine, o que significa que para se manter dentro do nível gratuito, o usuário pode utilizar nós do tipo `e2-micro` em conjunto com o Free Tier do Compute Engine [Google Cloud 2024].

O Free Tier é uma excelente porta de entrada para explorar e experimentar as capacidades da nuvem pública da Google, especialmente para estudantes, projetos acadêmicos e desenvolvedores iniciantes.

2.3 Python3

Criada no final dos anos 1980 por Guido van Rossum , com o desenvolvimento iniciado em dezembro de 1989 e o primeiro lançamento oficial em 1991 [Python Software Foundation 2025]. O python hoje e uma das linguagens de programação mais utilizadas na atualidade segundo o site [IEEE Spectrum 2024]

Linguagem essa que teve em sua premissa de ter uma sintase mais simples, e que ficou conhecida justamente por isso além de ter uma grande quantidade de bibliotecas para uso em diversos cenários.

Algumas das inúmeras bibliotecas do python são : subprocess , os , json , plataform , shutil , sys , abaixo iremos detalhar um pouco sobre a funcionalidade de cada uma

- **subprocess** O módulo subprocess permite a criação e gerenciamento de novos processos, além de possibilitar a captura de suas saídas e o controle de seus códigos de retorno. Trata-se de uma alternativa moderna e robusta a funções mais antigas como os.system, oferecendo maior controle e segurança na execução de comandos externos [Python Software Foundation 2025].

- **os** O módulo os oferece uma interface que permite interagir diretamente com funcionalidades do sistema operacional, como manipulação de arquivos e diretórios, gerenciamento de permissões e acesso a variáveis de ambiente, sendo essencial para scripts que precisam operar em diferentes plataformas [Python Software Foundation 2025].

- **json** Já o módulo json é utilizado para codificar e decodificar dados no formato JSON (JavaScript Object Notation), sendo amplamente empregado para realizar a serialização de dados estruturados e sua posterior comunicação entre diferentes aplicações [Python Software Foundation 2025].

- **plataform** O módulo platform fornece informações detalhadas sobre o sistema operacional em que o código está sendo executado, como nome, versão, arquitetura e distribuição. Esses dados permitem ajustar dinamicamente o comportamento da aplicação com base nas características do ambiente [Python Software Foundation 2025].

- **shutil** Complementando as operações com arquivos, o módulo shutil disponibiliza funções de alto nível para cópia, movimentação e remoção de arquivos e diretórios, ampliando as possibilidades oferecidas pelo módulo os [Python Software Foundation 2025].

- **sys** Por fim, o módulo sys oferece acesso direto a funcionalidades e variáveis do interpretador Python, como os argumentos da linha de comando, a manipulação da saída padrão e a configuração da busca por pacotes. Sua utilização é essencial para capturar parâmetros passados ao script e controlar o fluxo de execução [Python Software Foundation 2025].

2.4 Automação de Infraestrutura de TI

A prática de Infrastructure as Code (IaC) tem suas raízes nos primórdios da computação, com o desenvolvimento de ferramentas como o CFEngine por Mark Burgess em 1993 [Tyson 2023]. O conceito foi posteriormente consolidado por Kief Morris em seu

livro de 2016, que fornece práticas e padrões para gerenciar infraestruturas na era da nuvem [Morris 2016].

De acordo com o livro [Morris 2021] a prática de Infrastructure as Code substitui interfaces manuais por arquivos de código, permitindo que alterações sejam aplicadas de forma consistente e previsível. Isso reduz falhas humanas e garante maior confiabilidade na automação de infraestrutura. Além disso as ferramentas sem código não nos permitem explorar a infinidade de recursos existentes nos diversos ecossistema de tecnologia. A Infraestutura como código vai muito além de uma tecnologia, mas sim de uma forma de praticas e princípios que garantem agilidade e diversidade no dia a dia. Essa capacidade de mudanças rápidas se devem as tecnologias da nuvem que entregam a capacidade de provisionar e alterar hardware sobre demanda. Segundo [Morris 2021], um dos equívocos mais comuns é imaginar que a infraestrutura de TI seja estática. No contexto da automação, esse pensamento atrasa a evolução dos sistemas e contribui para a acumulação de débitos técnicos e falhas de segurança.

Além disso, o autor destaca que a automação deve ser incorporada desde as fases iniciais do projeto, pois postergar esse processo gera retrabalho e aumenta a fragilidade do sistema.

2.4.1 Docker

Docker, criado em 2013 por Solomon Hykes, transformou completamente a forma como softwares são construídos, empacotados e entregues. Inicialmente confundido com uma tecnologia de virtualização, o Docker foi, na verdade, o primeiro sistema acessível a utilizar a tecnologia de containerização de forma prática e escalável [Kane e Matthias 2023].

Seu principal propósito é encapsular aplicações e todas as suas dependências em containers, que podem ser executados de forma consistente em diferentes ambientes. Essa abordagem reduz a complexidade do trabalho entre equipes de desenvolvimento e operações, favorecendo a automação, o teste e o deployment contínuo de aplicações, além de diminuir falhas relacionadas ao ambiente de execução [Kane e Matthias 2023].

Diferente de máquinas virtuais, os contêineres compartilham o mesmo kernel do sistema operacional e utilizam menos recursos, tornando a execução mais leve e eficiente. Isso permite que empresas entreguem software com mais velocidade e segurança, enquanto mantêm uma arquitetura robusta e escalável [Kane e Matthias 2023].

Entre os principais benefícios do Docker estão: o empacotamento padronizado das aplicações com todos os arquivos necessários para sua execução, a consistência entre ambientes de desenvolvimento e produção, e a simplificação do ciclo DevOps. Além disso, o Docker ajuda a isolar alterações, evitando que configurações antigas ou erros persistam

entre versões [Kane e Matthias 2023].

O impacto do Docker foi tão significativo que ele transformou a expectativa sobre como pipelines de integração e entrega contínua (CI/CD) devem funcionar. Ao forçar boas práticas como a imutabilidade dos containers e o descarte de ambientes antigos durante novas implantações, o Docker contribui diretamente para a criação de sistemas mais confiáveis [Kane e Matthias 2023].

Desde sua introdução, o Docker se consolidou como uma tecnologia essencial para aplicações distribuídas modernas e continua sendo amplamente adotado por organizações ao redor do mundo [Kane e Matthias 2023].

O processo de deploy tradicional exigia que o time de desenvolvimento solicitasse uma máquina para a implantação do sistema. Esse recurso era provisionado pelo time de infraestrutura e, após o recebimento, realizava-se a instalação da aplicação e de cada dependência adicional, demandando interação constante com a equipe responsável pela configuração do ambiente. Esse processo é ilustrado na Figura 2.

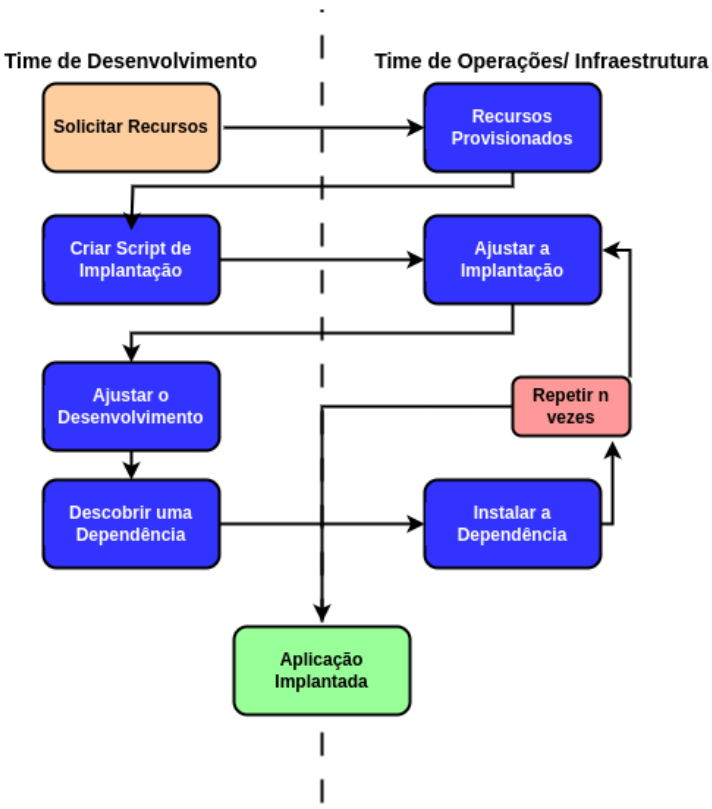

Figura 2 – Processo de deploy antes do Docker. Fonte: Adaptado de [Kane e Matthias 2023].

Com a introdução do Docker, o time de desenvolvimento passou a criar imagens contendo todos os componentes necessários, abstraindo as etapas de configuração da infraestrutura. O time de operações define apenas o ambiente de implantação da imagem

Docker, e a aplicação pode ser disponibilizada de forma ágil e padronizada, conforme ilustrado na Figura 3.

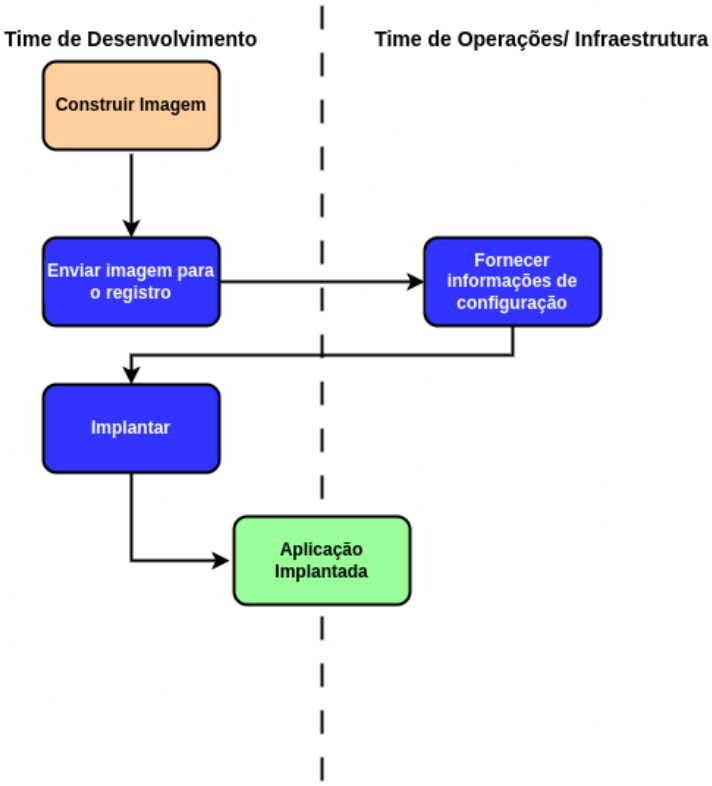

Figura 3 – Processo de deploy com Docker. Fonte: Adaptado de [Kane e Matthias 2023].

2.4.2 Kubernets

Kubernetes é atualmente a principal plataforma para orquestração de contêineres, permitindo a automação de processos essenciais como o provisionamento, escalonamento, monitoramento e recuperação de aplicações em ambientes distribuídos. Conforme apontado por [Hightower, Burns e Beda 2022], o Kubernetes abstrai a infraestrutura subjacente, permitindo que desenvolvedores e operadores tratem o cluster como uma única unidade lógica, em vez de um conjunto de máquinas individuais. Essa abordagem simplifica o gerenciamento de cargas de trabalho e promove maior consistência e confiabilidade.

2.4.2.1 Por que utilizar Kubernets

O Kubernetes oferece benefícios que impactam diretamente no ciclo de vida do desenvolvimento e da operação de software moderno:

- **Velocidade de desenvolvimento:** permite implementar e atualizar serviços com agilidade sem comprometer a disponibilidade [Burns, Beda e Hightower 2022].

- **Escalabilidade:** possibilita escalar tanto aplicações quanto equipes de desenvolvimento de forma organizada e eficiente [Burns, Beda e Hightower 2022].
- **Abstração de infraestrutura:** desacopla a aplicação do ambiente físico ou da cloud, permitindo portabilidade e consistência entre ambientes [Burns, Beda e Hightower 2022].
- **Eficiência operacional:** promove o uso de práticas como imutabilidade e configuração declarativa, melhorando a confiabilidade e facilitando rollback e automação [Burns, Beda e Hightower 2022].
- **Ecossistema cloud native:** é o núcleo de um vasto ecossistema de ferramentas voltadas para infraestrutura como código, observabilidade, CI/CD, e muito mais [Burns, Beda e Hightower 2022].

2.4.2.2 Conceitos Gerais

Quando falamos do kubernets os conceitos gerais que garantem o funcionamento e beneficios dessa solução são:

- **Imutabilidade**: contêineres são construídos como artefatos imutáveis, evitando modificações manuais e facilitando rollback [Burns, Beda e Hightower 2022].
- **Configuração declarativa**: define-se o estado desejado do sistema e o Kubernetes garante que o estado atual o reflita, promovendo maior previsibilidade e controle por meio de "infrastructure as code" [Burns, Beda e Hightower 2022].
- **Auto-recuperação** o Kubernetes monitora continuamente o estado dos componentes e realiza correções automáticas quando detecta falhas [Burns, Beda e Hightower 2022].
- **Escalonamento**: facilita tanto o aumento de capacidade de aplicações quanto a organização de times através de uma arquitetura desacoplada, baseada em microserviços [Burns, Beda e Hightower 2022].

2.4.3 Arquitetura do Kubernetes

Segundo a documentação oficial do Kubernetes [Kubernets.io 2025], a arquitetura da plataforma é composta por dois grandes blocos principais: o **Control Plane** e os **Node Components**. Essa divisão permite o gerenciamento eficiente e a orquestração de aplicações em ambientes distribuídos e escaláveis.

O **Control Plane** é responsável por controlar e gerenciar o estado desejado do cluster. Entre seus principais componentes, destacam-se:

- **API Server**: expõe a API RESTful do Kubernetes, sendo o ponto central de comunicação com o cluster. Todas as interações, como criação, leitura, atualização e exclusão de recursos, passam por esse componente.
- **Scheduler**: responsável por decidir em quais nós os pods recém-criados devem ser alocados, levando em consideração aspectos como disponibilidade de recursos e políticas de afinidade.
- **Controller Manager**: executa os controladores que monitoram o estado do cluster e garantem que ele atenda ao estado desejado, como o controle de replicação de pods e gerenciamento de falhas.
- **etcd**: é o armazenamento de chave-valor distribuído usado para manter todos os dados de configuração e estado do cluster de forma consistente e confiável.

Os **Node Components** são responsáveis por executar as aplicações em containers. Cada nó no cluster contém:

- **Kubelet**: agente que garante que os containers estejam rodando conforme especificado nas definições dos pods.
- **Kube Proxy**: gerencia as regras de rede e balanceamento de carga, permitindo a comunicação de rede dentro e fora do cluster.
- **Container Runtime**: é o software responsável por executar os containers propriamente ditos. Os runtimes mais utilizados são o `containerd` e o `CRI-O`.

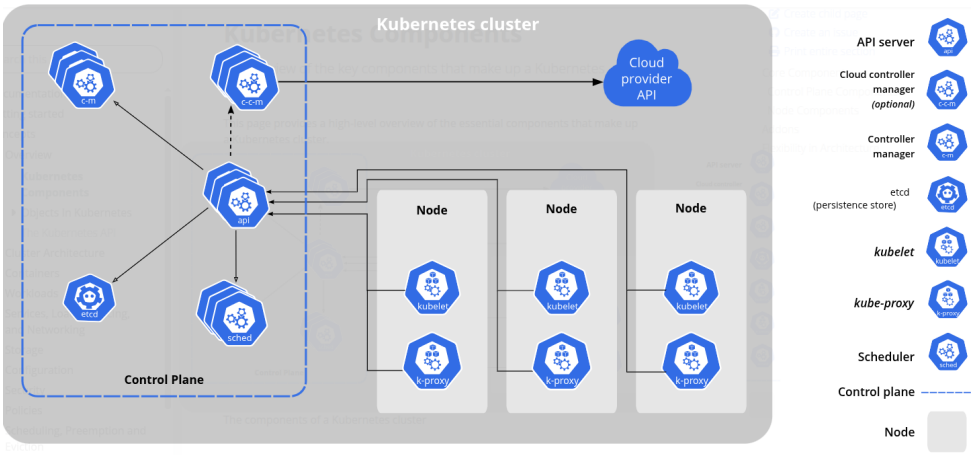

Figura 4 – Arquitetura dos componentes do Kubernetes. Fonte: [Kubernets.io 2025].

Essa arquitetura modular e escalável permite que o Kubernetes ofereça alta disponibilidade, balanceamento de carga, escalabilidade automática e recuperação de falhas de forma eficiente.

2.4.4 Terraform

Dentre as ferramentas de IaC, o Terraform se destaca por sua linguagem declarativa e sua capacidade de operar com múltiplos provedores de nuvem. De acordo com Brikman, o Terraform permite que a infraestrutura seja tratada como software versionado, promovendo reprodutibilidade, auditabilidade e automação nos ambientes computacionais [Brikman 2022].

2.4.4.1 Funcionamento do Terraform

O Terraform é baseado no conceito de infraestrutura como código (IaC), em que a infraestrutura de TI é definida através de arquivos de texto. Esses arquivos utilizam a extensão `.tf` e são escritos na linguagem HCL (HashiCorp Configuration Language).

O HCL foi projetado para ser uma linguagem de configuração human-readable, combinando a facilidade de leitura de formatos como JSON e YAML com a expressividade necessária para descrever estruturas de infraestrutura complexas. Sua sintaxe declarativa permite que o usuário defina o estado desejado dos recursos, sem precisar descrever imperativamente como atingi-lo, o que simplifica a automação e o gerenciamento de ambientes de nuvem [Brikman 2022].

O ciclo de vida operacional do Terraform é dividido em quatro etapas principais:

- **Write:** Definição da infraestrutura desejada nos arquivos `.tf`.
- **Plan:** Geração de um plano de execução que mostra as ações que o Terraform realizará para atingir o estado descrito.
- **Apply:** Aplicação efetiva das mudanças na infraestrutura, seguindo o plano previamente gerado.
- **Destroy:** Remoção controlada de recursos criados anteriormente, retornando a infraestrutura ao seu estado inicial.

Esse ciclo torna o Terraform uma ferramenta poderosa para implementar mudanças seguras, auditáveis e reproduzíveis em ambientes de nuvem.

A Figura 5 ilustra de forma simplificada o fluxo de funcionamento do Terraform, conforme apresentado por Brikman [Brikman 2022].

O processo se inicia com a definição da infraestrutura como código, por meio de arquivos de configuração conhecidos como *Terraform templates*. Esses arquivos são escritos na linguagem HCL (HashiCorp Configuration Language), permitindo descrever os recursos desejados, como instâncias de máquinas virtuais e bancos de dados. No exemplo da figura,

são especificados dois recursos: uma instância EC2 da AWS e uma instância de banco de dados MySQL.

Essas definições são processadas pelo Terraform, que se comunica com o provedor de nuvem (neste caso, AWS) através de APIs. O Terraform, então, provisiona os recursos definidos, criando as instâncias computacionais e serviços de banco de dados diretamente no ambiente do provedor de nuvem.

Esse fluxo reforça a principal proposta da ferramenta: simplificar a criação, alteração e destruição de ambientes de forma automatizada, segura e reprodutível.

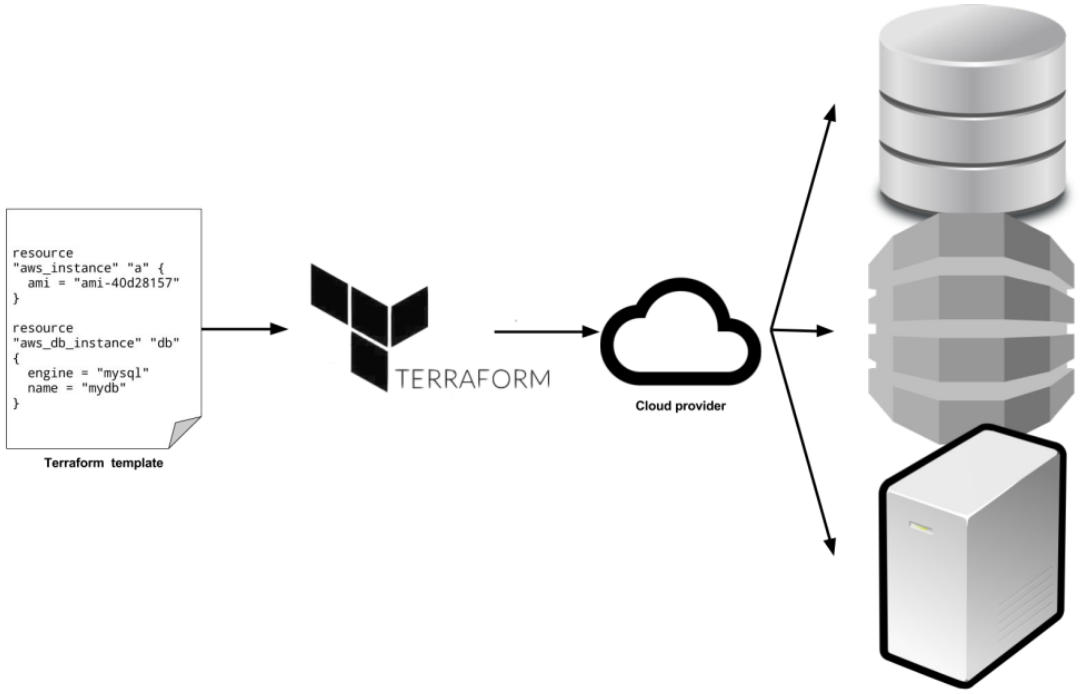

Figura 5 – Fluxo de funcionamento do Terraform. Fonte: [Brikman 2022].

2.5 Trabalhos Relacionados

Nesta seção, são apresentados estudos que possuem relação direta com a proposta deste trabalho, abordando práticas de automação, provisionamento de infraestrutura em nuvem e a modernização de ambientes computacionais.

- **Infrastructure as Code for Dynamic Deployments** [Sokolowski 2022]: O estudo explora a importância da prática de Infrastructure as Code (IaC) para possibilitar o provisionamento dinâmico de ambientes computacionais. Reforça como práticas de automação são essenciais para escalar aplicações de forma ágil e confiá-

vel, alinhando-se ao objetivo deste projeto de automatizar a criação de ambientes em nuvem.

- **Exploring Infrastructure as Code Using Terraform in Multi-Cloud Deployments** [Jasani, Patel e Doshi 2024]: Este trabalho investiga a utilização do Terraform para gerenciar infraestrutura em ambientes multicloud. Apesar do foco multicloud, o estudo reforça a flexibilidade e a padronização que o uso do Terraform proporciona, pontos que também são aplicados no provisionamento da infraestrutura no GCP realizado neste projeto.

- **Automação do Provisionamento de Infraestrutura em Nuvem para Implantação de Sistemas** [Treviso 2023]: Esta dissertação propõe um modelo de automação do provisionamento de recursos na nuvem, com foco em simplificar e agilizar implantações. A proposta está diretamente alinhada à ideia central do presente trabalho, aonde a proposta do trabalho dele busca auto-escaling e e redundancia e a nossa busca oferecer uma solução prática para automação utilizando recursos gratuitos do GCP.

- **Containerization of Legacy Applications** [Developer 2021]: O artigo discute o processo de modernização de aplicações legadas por meio da conteinerização, evidenciando os benefícios da utilização de tecnologias como Docker e Kubernetes. Essa abordagem é fundamental no projeto desenvolvido, que utiliza containers para garantir portabilidade e escalabilidade na implantação das aplicações.

- **Cloud Migration: A Case Study of Migrating an Enterprise IT System to IaaS** [Khajeh-Hosseini, Greenwood e Sommerville 2010]: Este estudo de caso apresenta a migração de sistemas tradicionais para uma arquitetura baseada em nuvem, evidenciando os ganhos em flexibilidade, escalabilidade e redução de custos. Esses benefícios também são esperados ao se utilizar a infraestrutura provisionada automaticamente no GCP, conforme proposto neste trabalho.

3 Metodologia

3.1 Método de Pesquisa

O método de pesquisa adotado neste trabalho é o estudo de caso. Essa abordagem permite analisar, de forma prática e detalhada, a implementação de um ambiente de computação em nuvem no Google Cloud Platform (GCP) utilizando recursos gratuitos (Free Tier).

A proposta consiste no desenvolvimento de um código que, a partir de uma chave de autenticação da GCP, automatiza o provisionamento de infraestrutura para o deploy de uma aplicação real. A aplicação selecionada para o estudo é proveniente do repositório *https://github.com/rinormaloku/k8s-mastery*, que será implantada primeiramente em uma máquina virtual (Compute Engine) e, posteriormente, em um cluster Kubernetes (GKE).

Por meio deste estudo de caso, serão coletadas métricas e observações relevantes, permitindo a análise comparativa entre os dois ambientes de deploy, considerando aspectos como tempo de implementação, consumo de recursos e facilidade de gerenciamento.

3.2 Visão Geral

Este projeto propõe a criação de um código automatizado capaz de preparar um ambiente de computação em nuvem no Google Cloud Platform (GCP), utilizando exclusivamente os recursos disponíveis no Free Tier. Com apenas uma chave de autenticação gerada pelo usuário após a criação da conta no GCP, o sistema é capaz de configurar automaticamente o ambiente necessário para realizar o deploy de uma aplicação, seja em uma máquina virtual (Compute Engine) ou em um cluster Kubernetes (GKE).

3.3 Ferramentas e Tecnologias Utilizadas

Neste projeto, foram utilizadas ferramentas amplamente reconhecidas no mercado de tecnologia da informação, com foco em automação, provisionamento de infraestrutura e orquestração de aplicações em nuvem. A escolha de cada tecnologia foi orientada pela sua robustez, facilidade de uso, compatibilidade com o Google Cloud Platform (GCP) e aderência às práticas modernas de DevOps.

A seguir, detalha-se a função de cada ferramenta no contexto deste projeto:

Google Cloud Platform (GCP): A plataforma foi selecionada como plataforma de computação em nuvem por oferecer uma ampla gama de serviços, fácil integração com ferramentas de automação e uma camada gratuita (Free Tier), que permite a realização de experimentos sem custos adicionais. O GCP também se destaca por sua escalabilidade, segurança e integração nativa com o Kubernetes (GKE), facilitando a implementação de ambientes modernos baseados em contêineres.

Terraform: A ferramenta de Infrastructure as Code (IaC) escolhida foi o Terraform devido à sua capacidade de criar, modificar e versionar recursos de infraestrutura de forma declarativa. Com ele, foi possível automatizar a criação de ambientes tanto na Compute Engine quanto no GKE, garantindo reprodutibilidade e agilidade no provisionamento. A escolha do Terraform também se justifica por sua ampla comunidade e integração nativa com o GCP.

Kubernetes: O Kubernetes Engine foi utilizado para a orquestração de contêineres, permitindo gerenciar a aplicação de forma escalável e resiliente. A escolha do GKE se deu pela sua facilidade de integração com os serviços do GCP e pela simplificação no gerenciamento de clusters Kubernetes, reduzindo a complexidade operacional e facilitando o deploy de aplicações em ambientes distribuídos.

Docker: Docker foi utilizado para o empacotamento e execução das aplicações em contêineres, permitindo isolar as dependências e garantir a portabilidade entre ambientes. Com o uso de Docker, foi possível criar imagens leves e facilmente transportáveis, acelerando o processo de deploy e garantindo que a aplicação funcione de forma consistente em qualquer ambiente.

Shell Script: Os scripts Shell foram empregados para automatizar tarefas de instalação de pacotes, configuração de ambientes e execução de comandos nas máquinas provisionadas. Essa automação foi essencial para reduzir o tempo de configuração manual e garantir que as etapas de instalação fossem executadas de forma padronizada e eficiente.

Python: Python foi utilizado para a criação de scripts responsáveis pela preparação da máquina local, gerenciamento de variáveis de ambiente e integração com a CLI do GCP. A escolha do Python se deve à sua simplicidade, grande quantidade de bibliotecas disponíveis para automação e facilidade de integração com APIs de terceiros, o que contribuiu diretamente para a automação dos processos de deploy.

3.4 Etapas do Projeto

Nesta seção, são apresentadas as principais etapas realizadas para a implementação do projeto, desde a configuração inicial dos ambientes até a execução final da aplicação. Cada etapa foi cuidadosamente planejada para garantir a automação dos processos, a

utilização eficiente dos recursos em nuvem e a coleta de métricas que permitam a análise comparativa entre diferentes cenários de deploy.

1. Instalação automática das dependências locais (Terraform, gcloud, pip, etc).
2. Recebimento da chave JSON do GCP pelo usuário.
3. Autenticação no GCP e configuração do projeto.
4. Provisionamento de ambiente (VM ou Cluster GKE).
5. Realização da conexão com os ambientes e deploy da aplicação clonada do GitHub.
6. Coleta de métricas para análise comparativa.

3.5 Arquitetura da Solução

A solução foi desenvolvida visando simplicidade e automação do processo de deploy. O processo inicia-se com a geração das credenciais de acesso (Cloud Key), permitindo a criação do ambiente diretamente na nuvem. A arquitetura geral do sistema é apresentada na Figura 6.

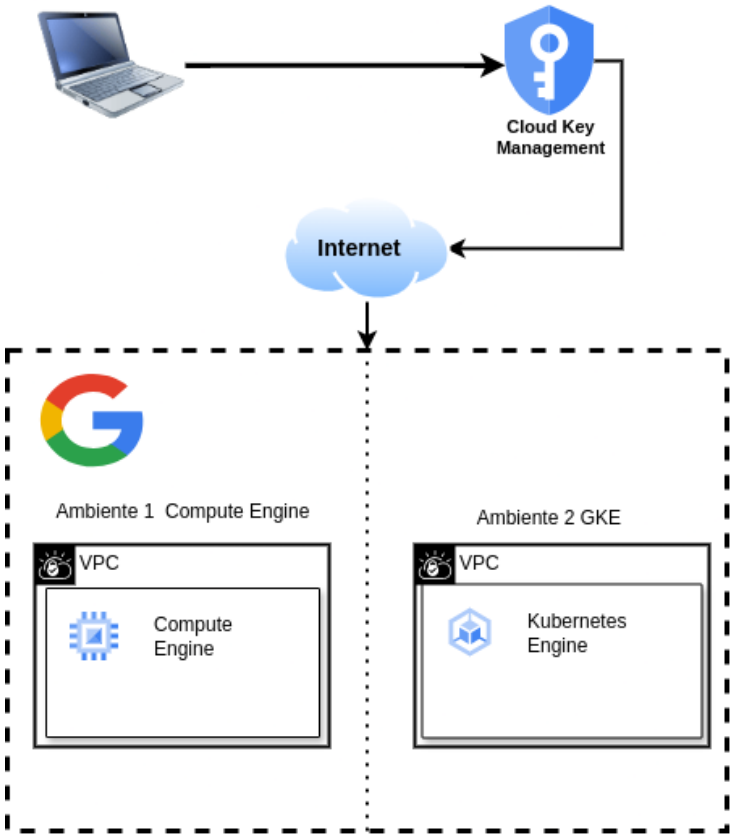

Figura 6 – Arquitetura geral do sistema. Fonte: Elaborado pelo autor.

A Figura 6 apresenta a arquitetura geral do sistema, demonstrando a interação entre a máquina local, a chave de autenticação gerada no GCP e o provedor Google Cloud. Essa comunicação ocorre de forma segura por meio da internet, possibilitando a automação dos ambientes de computação em nuvem, tanto no Compute Engine quanto no GKE.

3.5.1 Arquitetura Compute Engine

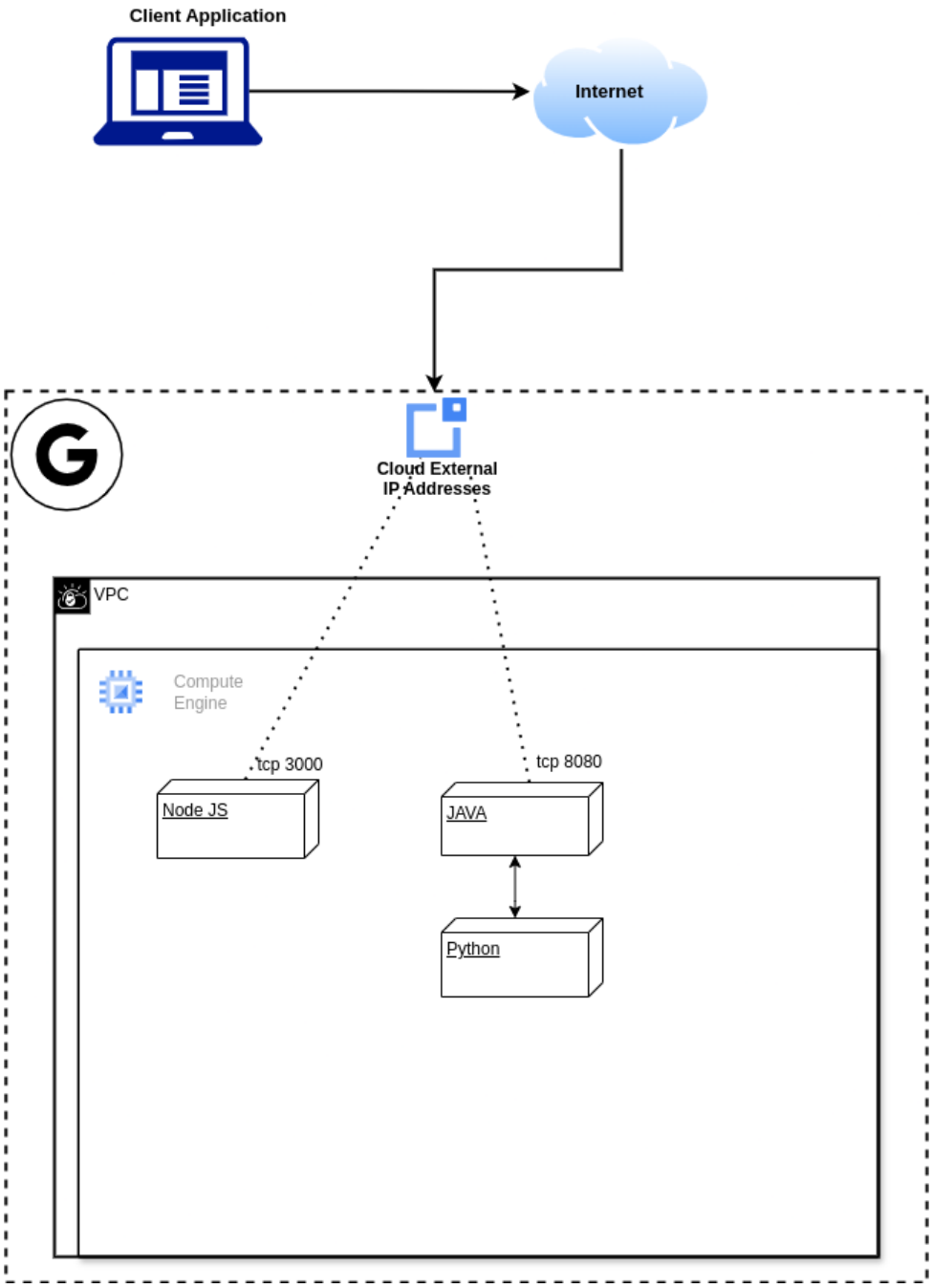

Figura 7 – Arquitetura do ambiente com Compute Engine. Fonte: Elaborado pelo autor.

A Figura 7 apresenta a arquitetura do ambiente implementado no Compute Engine. Nesse modelo, a aplicação é acessível por meio de um IP público.

3.5.2 Arquitetura GKE

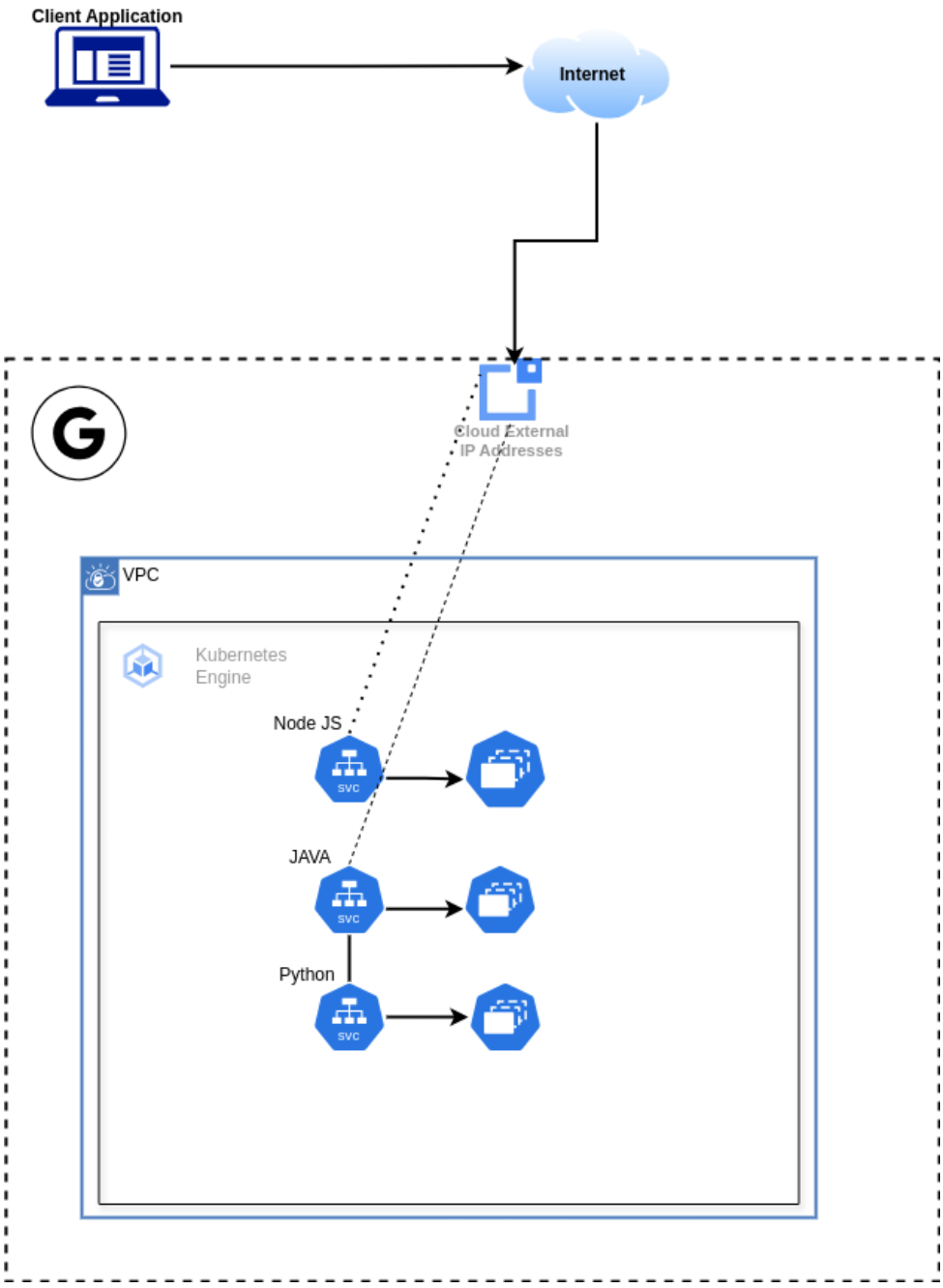

Figura 8 – Arquitetura do Ambiente Kubernetes. Fonte: Elaborado pelo autor.

A Figura 8 apresenta o ambiente GKE, no qual a aplicação pode ser acessada por meio de um IP alocado pelo serviço de Load Balancer, responsável por redirecionar as requisições para os pods

3.6 Aplicação Escolhida para Deploy

A aplicação escolhida para este projeto foi o repositório `https://github.com/rinormaloku/k8s-mastery`. A aplicação realiza a análise de sentimentos.. e retorna a polaridade dela, além disso ela serve como um guia para orquestração de contêineres com Kubernetes. O objetivo desta escolha é demonstrar o funcionamento do ambiente na Compute Engine e no GKE, utilizando uma abordagem em duas fases: primeiro, o deploy tradicional (instalação manual das dependências) e, posteriormente, o deploy com containers em um cluster Kubernetes.

A aplicação é composta por três componentes principais:

- Node.js: Um servidor web nginx que serve os arquivos estáticos da aplicação ReactJS.
- Java: Uma aplicação web em Spring, que lida com as requisições provenientes do frontend.
- Python: Uma aplicação que realiza análise sentimental sobre as requisições enviadas pela aplicação Java.

O fluxo de interação entre os componentes segue o seguinte processo:

1. O cliente requisita o arquivo 'index.html', que inclui os scripts da aplicação ReactJS.
2. O usuário interage com a interface, acionando uma requisição para a aplicação web em Java (Spring).
3. A aplicação Java direciona a requisição para a aplicação Python, responsável pela análise sentimental.
4. A aplicação Python calcula o sentimento e retorna a resposta à aplicação Java.
5. Por fim, a aplicação Java retorna a resposta ao frontend React, que exibe a informação ao usuário.

3.7 Critérios de Comparação

Para avaliar a eficiência dos ambientes provisionados no Compute Engine e no GKE, foram definidos critérios de comparação que permitem mensurar o desempenho, a escalabilidade e a complexidade de cada solução. A seguir, são apresentados os critérios utilizados e a forma como cada um será analisado:

- **Tempo de Provisionamento:** O tempo de provisionamento será medido utilizando um cronômetro manual, iniciado no momento da execução do script de provisionamento e finalizado quando o ambiente estiver completamente disponível para uso. Este critério permite avaliar a agilidade de cada abordagem na criação dos recursos necessários.

- **Tempo de Deploy da Aplicação:** O tempo de deploy será mensurado desde a execução do comando de deploy até o momento em que a aplicação estiver totalmente acessível no navegador. Esta métrica foi cronometrada manualmente.

- **Tempo de Resposta das Requisições HTTP:** Este critério avalia o tempo necessário para que uma requisição HTTP seja processada pela aplicação, desde o envio até o recebimento da resposta. A medição será realizada utilizando a ferramenta de inspeção do navegador Google Chrome, acessando a aba "Network"para verificar o tempo total das requisições realizadas ao backend. Esse indicador é fundamental para comparar a latência e o desempenho da aplicação em cada ambiente provisionado.

- **Recursos Utilizados (CPU/RAM):** A análise de recursos será realizada por meio do console do Google Cloud Platform, observando o consumo de CPU e memória durante a execução da aplicação em ambos os ambientes. Esse monitoramento permite avaliar o impacto da aplicação na infraestrutura provisionada e identificar possíveis gargalos de desempenho.

- **Redundância:** Será verificado se o ambiente provisionado é capaz de garantir a disponibilidade da aplicação em caso de falhas, considerando os recursos de redundância nativos de cada ambiente. No GKE, essa análise será realizada observando a capacidade de redistribuição de cargas entre pods, enquanto no Compute Engine será verificada a necessidade de intervenção manual para restabelecer o serviço.

- **Escalabilidade e Manutenção:** Este critério avalia a facilidade de escalar a aplicação e realizar manutenções nos ambientes provisionados. Será considerada a complexidade das etapas necessárias para adicionar novos recursos ou atualizar componentes, bem como a capacidade de o ambiente lidar com aumento repentino de demanda.

- **Tempo de exclusão:** Este critério avalia e compara o tempo necessário para exclusão desse recurso no GCP.

Com essa abordagem, será possível realizar uma análise comparativa prática e objetiva, identificando os pontos fortes e limitações de cada ambiente para o cenário proposto.

4 Estudo de Caso

4.1 Objetivo do Estudo

Este estudo de caso propõe o provisionamento automatizado de recursos em nuvem utilizando o Free-Tier do Google Cloud Platform (GCP). Após o provisionamento dos recursos, será realizado o deploy da aplicação *k8s-mastery*. Para isso, foi desenvolvido um menu automatizado que realiza o provisionamento dos ambientes no Compute Engine e no Google Kubernetes Engine (GKE). A comparação entre os dois ambientes considera métricas como tempo de provisionamento, tempo de deploy, desempenho, escalabilidade e redundância.

4.2 Preparação da Máquina

A etapa inicial deste trabalho tem como objetivo garantir a implantação facilitada de um ambiente utilizando a prática de Infraestrutura como Código (IaC). Para isso, é necessário que a máquina local esteja devidamente configurada com as dependências necessárias e apta a executar comandos de comunicação com o Google Cloud Platform (GCP).

A funcionalidade implementada na opção 1 do código automatiza a instalação das dependências na máquina local, considerando inicialmente ambientes com sistemas operacionais baseados em Debian ou Ubuntu. As premissas para o correto funcionamento incluem: a presença do Python instalado, utilização de um sistema operacional Debian/Ubuntu e a disponibilidade de uma chave de acesso à conta do GCP, devidamente criada e com as permissões apropriadas.

O primeiro procedimento executado pelo código consiste na instalação dos componentes essenciais para o processo de deploy. A ferramenta `PIP` é instalada inicialmente, sendo fundamental para a adição de pacotes que não são nativos da linguagem Python.

Após a instalação do `PIP`, realiza-se a instalação da CLI `gcloud`, responsável pela comunicação direta com o ambiente do Google Cloud. Esse componente oferece comandos nativos que permitem tanto a criação quanto a conexão com os recursos disponibilizados na nuvem.

Em sequência, é realizada a instalação do `Terraform`, ferramenta utilizada para o provisionamento automatizado da infraestrutura no GCP. Somente após essa instalação é possível criar e gerenciar os recursos de forma declarativa, conforme a sintaxe definida pelo provider do Google.

Por fim, utilizam-se novamente os recursos do `PIP` para a instalação de bibliotecas essenciais fornecidas pela Google, que serão utilizadas em etapas posteriores do projeto:

- `google-cloud-resource-manager`: Permite a manipulação de projetos, pastas e organizações no Google Cloud via scripts Python.
- `google-api-python-client`: Biblioteca genérica que possibilita a realização de chamadas HTTP para qualquer API do Google Cloud, como a ativação de APIs e a criação de recursos.
- `google-auth`: Responsável pelo gerenciamento de autenticação, permitindo a utilização de chaves e tokens para autenticação segura nas APIs do Google.
- `google-cloud-storage`: Facilita a manipulação de buckets e objetos no Google Cloud Storage utilizando scripts Python.

Essas bibliotecas serão utilizadas nas etapas seguintes do projeto, auxiliando nos processos de autenticação e ativação dos recursos necessários no ambiente do Google Cloud.

```
Menu:
1 - Instalar pré-requisitos
2 - Configurar GCP
3 - Criar uma VM
4 - Criar um cluster GKE
5 - Destruir a VM criada
6 - Destruir o cluster criado
7 - Conectar-se a VM criada
8 - Conectar-se ao cluster criado
0 - Sair
Escolha uma opção: 1
Verificando se o PIP está instalado...
PIP já está instalado.
Verificando se o Cloud CLI (gcloud) está instalado...
Cloud CLI (gcloud) já está instalado.
Verificando se o Terraform está instalado...
Terraform já está instalado.
Pacote google-cloud-resource-manager já está instalado.
Pacote google-api-python-client já está instalado.
Pacote google-auth já está instalado.
Pacote google-cloud-storage já está instalado.
Todos os pré-requisitos foram instalados com sucesso.
```

Figura 9 – Tela do terminal da máquina local após a execução da opção 1 do menu. Fonte: Elaborado pelo autor.

4.3 Autenticação no GCP

Conforme abordado na seção anterior, uma premissa fundamental para a execução deste projeto é a disponibilidade da chave de autenticação com o Google Cloud Platform (GCP). O processo de criação dessa chave está documentado no arquivo `README` do repositório, oferecendo um passo a passo para orientar sua geração.

A criação da chave de autenticação é realizada por meio do Google Cloud Console, envolvendo as seguintes etapas: criação de um novo projeto, geração de uma conta de serviço e criação da chave no formato JSON, com a atribuição dos papéis de Administrador do Projeto e Administrador da API Management. Estes papéis são necessários para permitir a criação de recursos computacionais por meio de APIs. Após a geração, recomenda-se armazenar a chave de forma segura. Mais detalhes sobre esse procedimento podem ser consultados na documentação oficial do Google [Cloud 2025].

A funcionalidade implementada na opção 2 do código é responsável pela configuração do ambiente no GCP, a qual está dividida nas seguintes etapas.

4.3.1 Autenticação com a Chave de Serviço

A primeira etapa consiste na autenticação utilizando a chave de serviço. Na função authenticate_with_service_account, é verificado se a credencial da chave de serviço está disponível. Caso esteja, o caminho da chave é armazenado na variável de ambiente GOOGLE_APPLICATION_CREDENTIALS, permitindo que as bibliotecas do Google Cloud utilizem automaticamente essa chave para autenticação nas interações subsequentes com a plataforma.

4.3.2 Recuperação do project_id

Na função get_project_id_from_credentials, o valor do project_id é recuperado a partir da chave de autenticação. Este identificador é exclusivo dentro do GCP e vincula os recursos provisionados a um projeto específico. O project_id será utilizado em todas as interações subsequentes, garantindo que os recursos sejam criados e gerenciados no projeto correto. A obtenção desse valor é realizada diretamente a partir do arquivo de credenciais, conforme descrito na documentação oficial do GCP [Cloud 2025].

4.3.3 Habilitação das APIs Necessárias

Antes de prosseguir com a terceira função, é importante compreender como o GCP gerencia suas APIs. No nosso código, são utilizadas as bibliotecas do GCP para habilitar as APIs necessárias à execução do projeto. As APIs ativadas incluem:

- `cloudresourcemanager.googleapis.com`: Permite criar, listar, excluir e gerenciar projetos, pastas e organizações no GCP.
- `compute.googleapis.com`: Gerencia recursos do Compute Engine, incluindo instâncias de máquinas virtuais (VMs), regras de firewall e redes virtuais (VPCs).
- `container.googleapis.com`: Responsável pela gestão do Google Kubernetes Engine (GKE), permitindo a criação de clusters Kubernetes gerenciados, configuração de autenticação e autorização, além de adicionar ou remover nós do cluster.

A ativação dessas APIs é um requisito fundamental para que o código possa interagir com os recursos de computação e contêineres do GCP. Segundo a documentação oficial do Google [Cloud 2025], as APIs são desabilitadas por padrão como medida de segurança, visando evitar o uso não autorizado de recursos e garantir maior controle sobre os custos da plataforma.

4.3.4 Geração do Arquivo terraform.tfvars

A última etapa da configuração consiste na criação dos arquivos terraform.tfvars, responsáveis por armazenar as variáveis utilizadas pelo Terraform para autenticação no GCP e configuração dos recursos. A função write_tfvars_from_env realiza a geração automática desses arquivos.

Para o correto funcionamento do Terraform, é necessário fornecer informações cruciais, como:

- O identificador do projeto GCP no qual os recursos serão provisionados;
- A região de criação dos recursos;
- A zona específica dentro da região selecionada.

Os arquivos terraform.tfvars são gerados nos diretórios infraestrutura/computeengine e infraestrutura/gke, de acordo com a estrutura do repositório. A Figura 10 apresenta um exemplo dos arquivos criados para o provisionamento da infraestrutura.

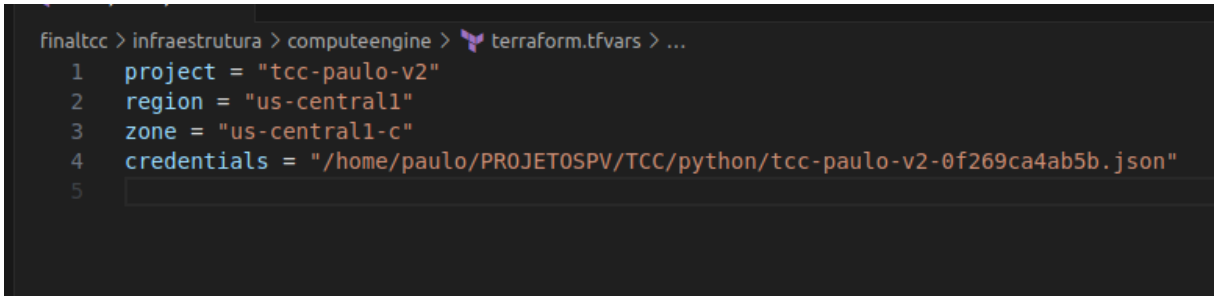

Figura 10 – Exemplo de arquivos tfvars criados para o provisionamento do ambiente. Fonte: Elaborado pelo autor.

Esses arquivos contêm as variáveis necessárias para que o Terraform realize a criação dos recursos conforme as configurações definidas. Conforme discutido no livro *Terraform: Up and Running: Writing Infrastructure as Code* [Brikman 2022], o uso de arquivos `tfvars` permite a parametrização dos ambientes, promovendo flexibilidade e consistência no processo de provisionamento. Durante a execução, essas variáveis são lidas pelo Terraform, possibilitando a criação automatizada dos recursos de acordo com os parâmetros fornecidos.

Figura 11 – Tela do terminal da máquina local após a execução da opção 2 do menu. Fonte: Elaborado pelo autor.

4.4 Criação de VM no Compute Engine

Após a configuração do ambiente e a autenticação com o GCP, a próxima etapa consiste na criação da máquina virtual (VM) utilizando o Compute Engine.

As definições da máquina virtual são especificadas no arquivo main.tf, escrito na linguagem HCL (HashiCorp Configuration Language). Cada provedor de nuvem possui parâmetros obrigatórios para a criação de recursos. No GCP, são utilizados os recursos provider, compute network, compute subnetwork e google compute instance. A seguir, são apresentados os blocos de código com a respectiva explicação das configurações realizadas.

4.4.1 Configuração do Provider do Google Cloud

Listing 4.1 – Arquivo main.tf - Provider Google

```
terraform {
required_providers {
google = {
source = "hashicorp/google"
version = "6.15.0"
}
}
}
```

Este bloco define a utilização do provider oficial do Google Cloud, desenvolvido pela HashiCorp, além de estabelecer a versão 6.15.0. Essa definição garante compatibilidade e estabilidade na execução dos recursos.

Listing 4.2 – Arquivo main.tf - Configuração do Provider

```
provider "google" {
project = var.project
region = var.region
zone = var.zone
credentials = file(var.credentials)
}

variable "project" {}
variable "region" {}
variable "zone" {}
variable "credentials" {}
```

Neste trecho, são configurados os parâmetros de conexão com o Google Cloud Platform, utilizando variáveis definidas previamente (`var.project`, `var.region`, `var.zone` e `var.credentials`). O arquivo gerado na etapa de autenticação armazena essas informações, permitindo o correto direcionamento do provisionamento da infraestrutura.

4.4.2 Criação de Rede, Sub-rede e Regras de Firewall

Listing 4.3 – Arquivo main.tf - Criação de Rede e Regras de Firewall

```
resource "google_compute_network" "vpc_network" {
```

```
name = "my-custom-mode-network"
auto_create_subnetworks = false
mtu = 1460
}

resource "google_compute_subnetwork" "default" {
name = "my-custom-subnet"
ip_cidr_range = "10.0.1.0/24"
region = var.region
network = google_compute_network.vpc_network.id
}

resource "google_compute_firewall" "ssh_rule" {
name = "allow-ssh"
network = google_compute_network.vpc_network.id

allow {
protocol = "tcp"
ports = ["22", "3000", "8080"]
}

source_ranges = ["0.0.0.0/0"]
target_tags = ["ssh"]
}
```

Neste trecho do código, são definidos:

- google_compute_network: Criação de uma rede VPC denominada mycustommode-network. A criação automática de sub-redes foi desativada, permitindo maior controle sobre a segmentação de endereços IP.
- google_compute_subnetwork: Criação de uma sub-rede regional com faixa de endereços IP 10.0.1.0/24. As instâncias criadas terão endereços internos dentro dessa faixa.
- google_compute_firewall: Definição de regras de firewall para permitir comunicação nas portas 22 (SSH), 3000 (Node.js) e 8080 (Java). A regra permite o acesso de qualquer IP público 0.0.0.0/0 e é aplicada a instâncias com a tag ssh.

4.4.3 Criação da Instância de Computação

Listing 4.4 – Arquivo main.tf - Criação da Instância

```
resource "google_compute_instance" "default" {
name = "flask-vm"
machine_type = "e2-micro"
```

```
zone = var.zone
tags = ["ssh"]

scheduling {
preemptible = true
automatic_restart = false
}

boot_disk {
initialize_params {
image = "debian-cloud/debian-11"
}
}
}
```

Este bloco de código cria a instância de VM denominada flask-vm, utilizando o tipo de máquina e2-micro, que faz parte da camada gratuita (Free Tier) do GCP. Para garantir a elegibilidade ao Free Tier, a opção preemptible é ativada, permitindo que a máquina seja interrompida a qualquer momento pelo GCP. A tag ssh vincula a instância à regra de firewall configurada anteriormente. A imagem de inicialização utilizada é baseada no Debian 11, disponibilizada oficialmente pelo Google Cloud.

4.4.4 Execução de Script na Inicialização da VM

Listing 4.5 – Arquivo main.tf - Script de Inicialização

```
metadata_startup_script = "sudo␣apt-get␣update;␣sudo␣apt-get␣install␣-yq
    ␣build-essential␣npm␣openjdk-11-jdk␣maven␣python3␣python3-pip␣rsync;␣
    pip␣install␣--upgrade␣flask"
```

O script definido acima é executado automaticamente no momento da inicialização da VM, realizando as seguintes ações:

- Atualização do sistema operacional.
- Instalação de dependências essenciais: buildessential, npm, Node.js, openjdk-11-jdk e maven (Java), python3 e pip (Python).
- Instalação do framework Flask utilizando o comando pip install –upgrade flask, preparando o ambiente para a execução de aplicações em Flask, Java e Node.js.

4.4.5 Execução dos Comandos Terraform

Com o manifesto main.tf finalizado, a criação da infraestrutura é realizada por meio dos comandos terraform init e terraform apply, os quais são executados automaticamente ao selecionar a opção 3 no menu do código.

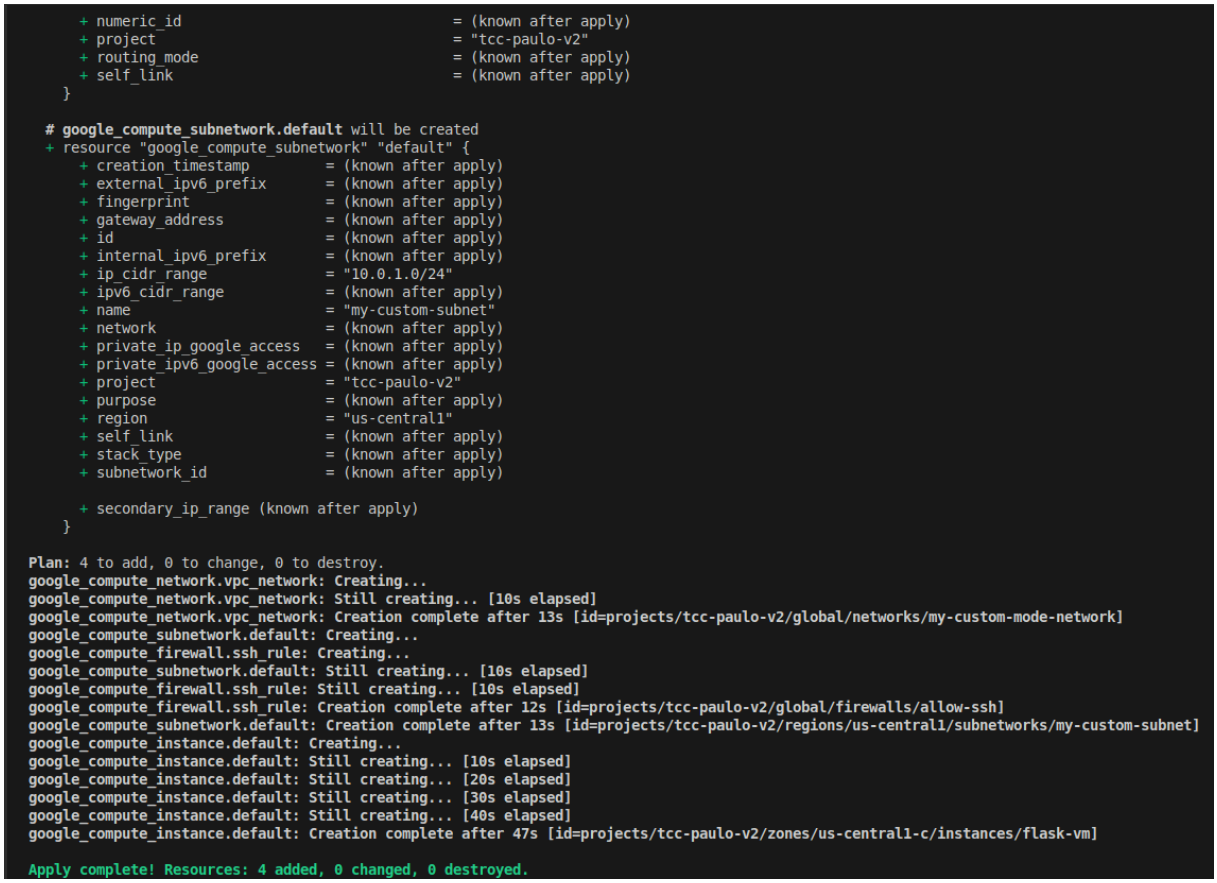

Figura 12 – Tela do terminal após a execução da opção 3 do menu. Fonte: Elaborado pelo autor.

- terraform init: Inicializa o diretório de trabalho, realizando o download dos providers necessários e configurando o ambiente para o gerenciamento da infraestrutura.
- terraform apply: Aplica as configurações definidas no manifesto, realizando a criação dos recursos especificados. No contexto deste projeto, este comando cria a instância da máquina virtual e os recursos de rede associados.

O tempo estimado para a conclusão do deploy da máquina é de aproximadamente 3 minutos. Após a execução dos comandos, a VM estará pronta para acesso e para o deploy da aplicação.

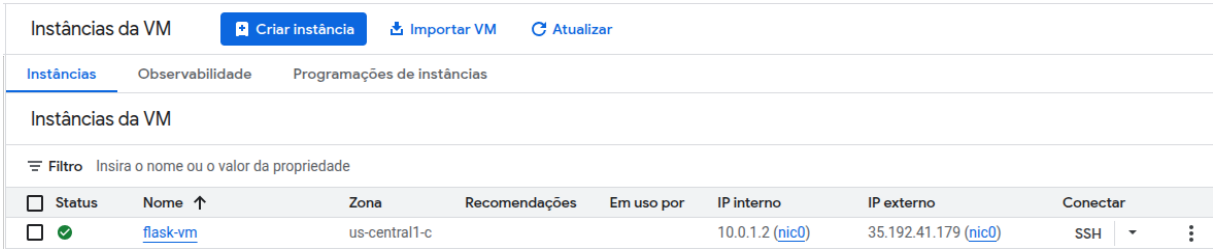

Figura 13 – Console do GCP apresentando a VM provisionada. Fonte: Elaborado pelo autor.

4.5 Criação de Cluster GKE

A criação do cluster Kubernetes (GKE) no Google Cloud segue um processo semelhante ao da criação da instância de máquina virtual, porém com configurações específicas para o gerenciamento de contêineres. O GKE é uma solução do Google Cloud voltada para a orquestração de contêineres, permitindo a automação de implantação, dimensionamento e gerenciamento de aplicativos em ambientes baseados em contêineres.

Os parâmetros necessários para a criação do cluster incluem: provider, compute network, subnetwork, cluster node pool. A seguir, são apresentados os blocos de código responsáveis pela criação do cluster, com a devida explicação de seus componentes.

4.5.1 Criação de Rede e Sub-rede para o GKE

Listing 4.6 – Arquivo main.tf - Criação de Rede e Sub-rede para GKE

```
resource "google_compute_network" "vpc_network" {
name = "gke-vpc"
auto_create_subnetworks = false
}

resource "google_compute_subnetwork" "subnet_gke" {
name = "gke-subnet"
ip_cidr_range = "10.0.0.0/16"
region = var.region
network = google_compute_network.vpc_network.id
}
```

A principal diferença neste cenário é a faixa de endereços IP atribuída à sub-rede. Foi utilizado um intervalo 16, considerando a necessidade do Kubernetes de dispor de uma quantidade maior de endereços IP. Como o GKE realiza o escalonamento automático de pods, é essencial garantir a disponibilidade de IPs para comunicação interna entre as instâncias e os pods.

- google_compute_network: Criação de uma rede VPC denominada gkevpc. A criação automática de sub-redes foi desativada, permitindo o controle manual da segmentação IP.
- google_compute_subnetwork: Definição de uma sub-rede com faixa IP 10.0.0.0/16. Esta sub-rede será utilizada pelo GKE tanto para o tráfego interno entre os nós quanto para a comunicação do cluster com outros recursos da VPC.

4.5.2 Criação da Regra de Firewall para o GKE

Listing 4.7 – Arquivo main.tf - Regra de Firewall para GKE

```
resource "google_compute_firewall" "allow_gke" {
name = "allow-gke"
network = google_compute_network.vpc_network.id

allow {
protocol = "tcp"
ports = ["443", "10250"]
}

source_ranges = ["0.0.0.0/0"]
target_tags = ["gke-node"]
}
```

A regra de firewall criada permite o tráfego nas portas necessárias para o funcionamento e administração do cluster:

- `Porta 443`: Utilizada para comunicação segura via HTTPS, necessária para o acesso à API do Kubernetes.
- `Porta 10250`: Utilizada para a comunicação entre os nós do GKE e o plano de controle do cluster.
- `Origem`: O tráfego é permitido de qualquer endereço IP público (`0.0.0.0/0`), garantindo a acessibilidade externa para a administração do cluster.

A regra de firewall aplica-se às instâncias com a tag gkenode, assegurando que apenas os nós do GKE possam ser acessados pelas portas configuradas.

4.5.3 Criação do Cluster GKE

Listing 4.8 – Arquivo main.tf - Criação do Cluster GKE

```
resource "google_container_cluster" "gke_cluster" {
name = "gke-cluster"
```

```
3 location = var.zone
4 deletion_protection = false
5 network = google_compute_network.vpc_network.id
6 subnetwork = google_compute_subnetwork.subnet_gke.id
7
8 remove_default_node_pool = true
9 initial_node_count = 1
10 }
```

Neste trecho de código, realiza-se a criação do cluster:

- google_container_cluster: Criação do cluster denominado gkecluster, configurado para utilizar a VPC e a sub-rede previamente definidas.
- remove_default_node_pool: Desabilita o pool de nós padrão do GKE, permitindo o controle manual sobre a criação de pools personalizados.

4.5.4 Criação do Node Pool do Cluster GKE

Listing 4.9 – Arquivo main.tf - Node Pool do GKE

```
1 resource "google_container_node_pool" "primary_nodes" {
2 name = "primary-node-pool"
3 cluster = google_container_cluster.gke_cluster.name
4 location = var.zone
5 node_count = 2
6
7 node_config {
8 preemptible = true
9 machine_type = "e2-micro"
10 disk_size_gb = 30
11 oauth_scopes = [
12 "https://www.googleapis.com/auth/cloud-platform"
13 ]
14 tags = ["gke-node"]
15 }
16 }
```

O Node Pool configura o número de nós e as características das instâncias utilizadas no cluster:

- preemptible: Define que os nós são preemptíveis, o que reduz os custos de operação, com a contrapartida de possível interrupção dos recursos.
- node_count: O pool de nós será composto por 2 instâncias.

- machine_type: O tipo de máquina utilizado será e2-micro, contemplado no Free Tier do GCP, ideal para ambientes de teste e cargas de trabalho leves.

Após a finalização do arquivo main.tf, a criação do cluster é realizada por meio da opção 4 do menu do código, que executa automaticamente os comandos terraform init e terraform apply.

```
google_container_cluster.gke_cluster: Still creating... [4m20s elapsed]
google_container_cluster.gke_cluster: Still creating... [4m30s elapsed]
google_container_cluster.gke_cluster: Still creating... [4m40s elapsed]
google_container_cluster.gke_cluster: Still creating... [4m50s elapsed]
google_container_cluster.gke_cluster: Still creating... [5m0s elapsed]
google_container_cluster.gke_cluster: Still creating... [5m10s elapsed]
google_container_cluster.gke_cluster: Still creating... [5m20s elapsed]
google_container_cluster.gke_cluster: Still creating... [5m30s elapsed]
google_container_cluster.gke_cluster: Still creating... [5m40s elapsed]
google_container_cluster.gke_cluster: Still creating... [5m50s elapsed]
google_container_cluster.gke_cluster: Still creating... [6m0s elapsed]
google_container_cluster.gke_cluster: Still creating... [6m10s elapsed]
google_container_cluster.gke_cluster: Still creating... [6m20s elapsed]
google_container_cluster.gke_cluster: Still creating... [6m30s elapsed]
google_container_cluster.gke_cluster: Still creating... [6m40s elapsed]
google_container_cluster.gke_cluster: Still creating... [6m50s elapsed]
google_container_cluster.gke_cluster: Still creating... [7m0s elapsed]
google_container_cluster.gke_cluster: Still creating... [7m10s elapsed]
google_container_cluster.gke_cluster: Still creating... [7m20s elapsed]
google_container_cluster.gke_cluster: Still creating... [7m30s elapsed]
google_container_cluster.gke_cluster: Still creating... [7m40s elapsed]
google_container_cluster.gke_cluster: Still creating... [7m50s elapsed]
google_container_cluster.gke_cluster: Still creating... [8m0s elapsed]
google_container_cluster.gke_cluster: Still creating... [8m10s elapsed]
google_container_cluster.gke_cluster: Still creating... [8m20s elapsed]
google_container_cluster.gke_cluster: Still creating... [8m30s elapsed]
google_container_cluster.gke_cluster: Still creating... [8m40s elapsed]
google_container_cluster.gke_cluster: Still creating... [8m50s elapsed]
google_container_cluster.gke_cluster: Still creating... [9m0s elapsed]
google_container_cluster.gke_cluster: Still creating... [9m10s elapsed]
google_container_cluster.gke_cluster: Still creating... [9m20s elapsed]
google_container_cluster.gke_cluster: Still creating... [9m30s elapsed]
google_container_cluster.gke_cluster: Still creating... [9m40s elapsed]
google_container_cluster.gke_cluster: Still creating... [9m50s elapsed]
google_container_cluster.gke_cluster: Still creating... [10m0s elapsed]
google_container_cluster.gke_cluster: Creation complete after 10m1s [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster]
google_container_node_pool.primary_nodes: Creating...
google_container_node_pool.primary_nodes: Still creating... [10s elapsed]
google_container_node_pool.primary_nodes: Still creating... [20s elapsed]
google_container_node_pool.primary_nodes: Still creating... [30s elapsed]
google_container_node_pool.primary_nodes: Still creating... [40s elapsed]
google_container_node_pool.primary_nodes: Still creating... [50s elapsed]
google_container_node_pool.primary_nodes: Still creating... [1m0s elapsed]
google_container_node_pool.primary_nodes: Still creating... [1m10s elapsed]
google_container_node_pool.primary_nodes: Creation complete after 1m16s [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster/nodePools/primary-node
-pool]
```

Figura 14 – Tela do terminal após a execução da opção 4 do menu. Fonte: Elaborado pelo autor.

O tempo estimado para a conclusão do deploy do cluster é de aproximadamente 12 minutos. Após a execução dos comandos, o cluster estará disponível para acesso e pronto para o deploy da aplicação.

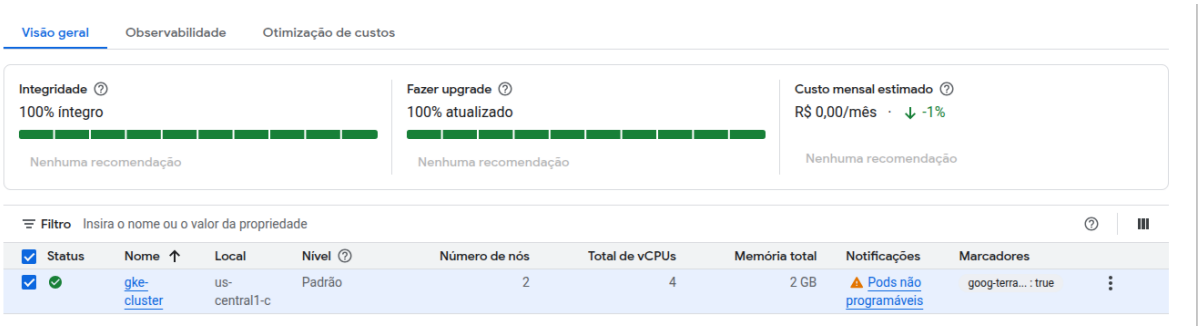

Figura 15 – Console do GCP apresentando o cluster GKE provisionado. Fonte: Elaborado pelo autor.

4.6 Deploy Compute Engine

Após o provisionamento do ambiente, inicia-se o processo de deploy da aplicação. A proposta central do código é simplificar a utilização dos recursos em nuvem. A conexão via SSH com a máquina virtual é realizada por meio da opção 7 do menu implementado no código.

Essa conexão é efetuada utilizando o comando `gcloud compute ssh`, integrante do Google Cloud SDK. Os parâmetros de zona e projeto são fornecidos por meio de variáveis de ambiente, o que automatiza o processo de conexão e elimina a necessidade de inserção manual desses parâmetros. Segundo a documentação oficial da Google Cloud [Google 2023], o comando `gcloud compute ssh` possibilita o acesso rápido e seguro às instâncias de máquinas virtuais, desde que as credenciais e variáveis de ambiente estejam corretamente configuradas.

```
Menu:
1 - Instalar pré-requisitos
2 - Configurar GCP
3 - Criar uma VM
4 - Criar um cluster GKE
5 - Destruir a VM criada
6 - Destruir o cluster criado
7 - Conectar-se a VM criada
8 - Conectar-se ao cluster criado
0 - Sair
Escolha uma opção: 7
Linux flask-vm 5.10.0-34-cloud-amd64 #1 SMP Debian 5.10.234-1 (2025-02-24) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu May  1 15:59:24 2025 from 177.191.117.170
paulo@flask-vm:~$
```

Figura 16 – Tela do terminal após a execução da opção 7 do menu. Fonte: Elaborado pelo autor.

4.6.1 Deploy da Aplicação e Ajustes Necessários

Com a conexão estabelecida, o próximo passo consiste na realização do deploy da aplicação. Este processo foi automatizado por meio de um script `.sh`, cujo conteúdo pode ser consultado no Apêndice A.1. A utilização do script permite reduzir o tempo de implantação e garantir a execução padronizada das etapas, favorecendo a coleta de métricas e a reprodutibilidade dos testes.

O script é responsável pela instalação das dependências e pela configuração dos componentes da aplicação: frontend, backend e módulo de análise sentimental. A seguir, apresentam-se as principais instruções contidas no script:

- `if [[ $EUID -ne 0 ]]; then`: Verifica se a execução está sendo realizada com privilégios de superusuário. Caso contrário, reinicia o script com as permissões necessárias.
- `set -e`: Interrompe a execução do script caso algum comando retorne erro, evitando a continuidade de processos incorretos.
- `set -x`: Exibe no terminal os comandos à medida que são executados, facilitando o processo de depuração.
- `EXTERNAL_IP=$(curl -s ifconfig.me)`: Obtém o IP externo da instância, armaze nando-o na variável `EXTERNAL_IP`. Esse endereço será utilizado na configuração do frontend.
- `apt update && apt install -y git npm maven python3-pip curl openjdk-11-jdk`: Realiza a atualização do sistema e instala as principais dependências para execução da aplicação.
- `git clone https://github.com/rinormaloku/k8s-mastery.git`: Clona o repositório da aplicação.
- `npm install`: Instala as dependências do frontend baseadas em Node.js.
- `sed -i "s|http://localhost:8080/sentiment|http://$EXTERNAL_IP:8080/sentiment|g"src/App.js`: Atualiza a configuração do frontend para que as requisições sejam direcionadas ao IP público da instância.
- `npm start &`: Inicia o frontend de forma assíncrona.
- `mvn install`: Instala as dependências do backend Java utilizando o Maven.
- `java -jar target/sentiment-analysis-web-0.0.1-SNAPSHOT.jar -sa.logic.api.url=http://localhost:5000`: Inicia o backend apontando para o serviço de análise de sentimentos.
- `python3 -m pip install -upgrade Werkzeug Flask textblob`: Instala e atualiza os pacotes Python necessários.
- `python3 -m textblob.download_corpora`: Realiza o download dos dados necessários para a biblioteca TextBlob.

- `python3 sentiment_analysis.py &`: Inicia o serviço de análise de sentimentos de forma assíncrona.

Após aproximadamente 23 minutos, a aplicação torna-se acessível externamente por meio do endereço IP `http://35.192.41.179:3000/`. Esse endereço foi provisionado automaticamente pelo Terraform por meio da configuração access_config no bloco de subnetwork, permitindo o acesso externo à aplicação.

Observa-se que o tempo de deploy foi reduzido em virtude da automação proporcionada pelo script `.sh`. Sem a utilização dessa automação, o processo demandaria um tempo superior, estimado em aproximadamente 10 minutos e 30 segundos adicionais.

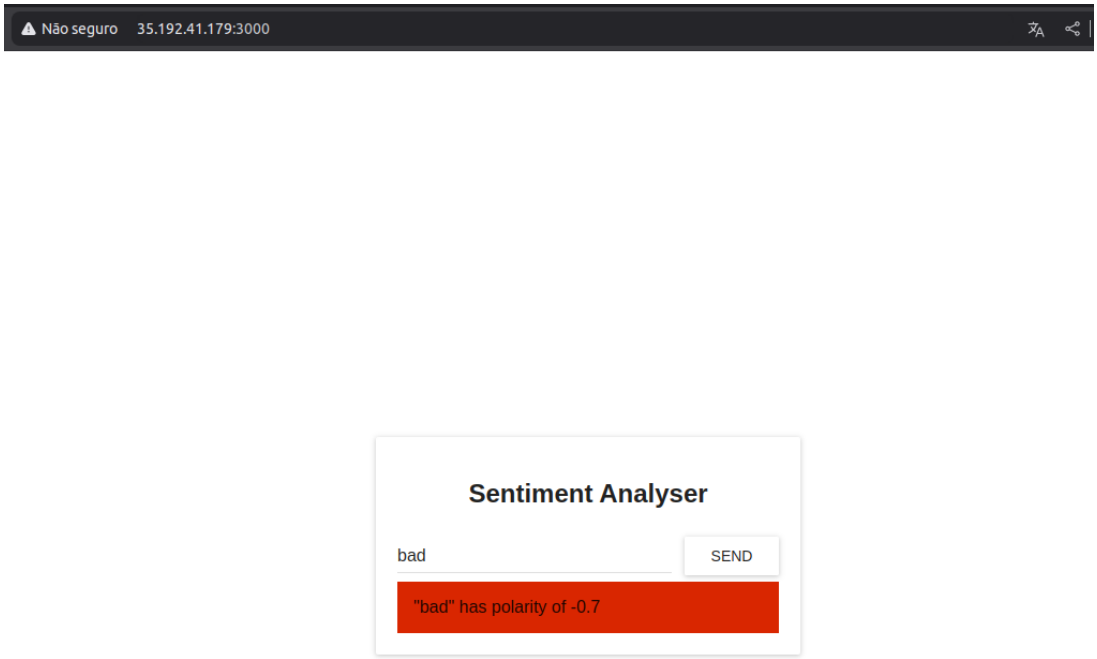

Figura 17 – Aplicação acessível via internet após o deploy na VM. Fonte: Elaborado pelo autor.

A interação entre os componentes da aplicação pode ser observada na Figura 18.

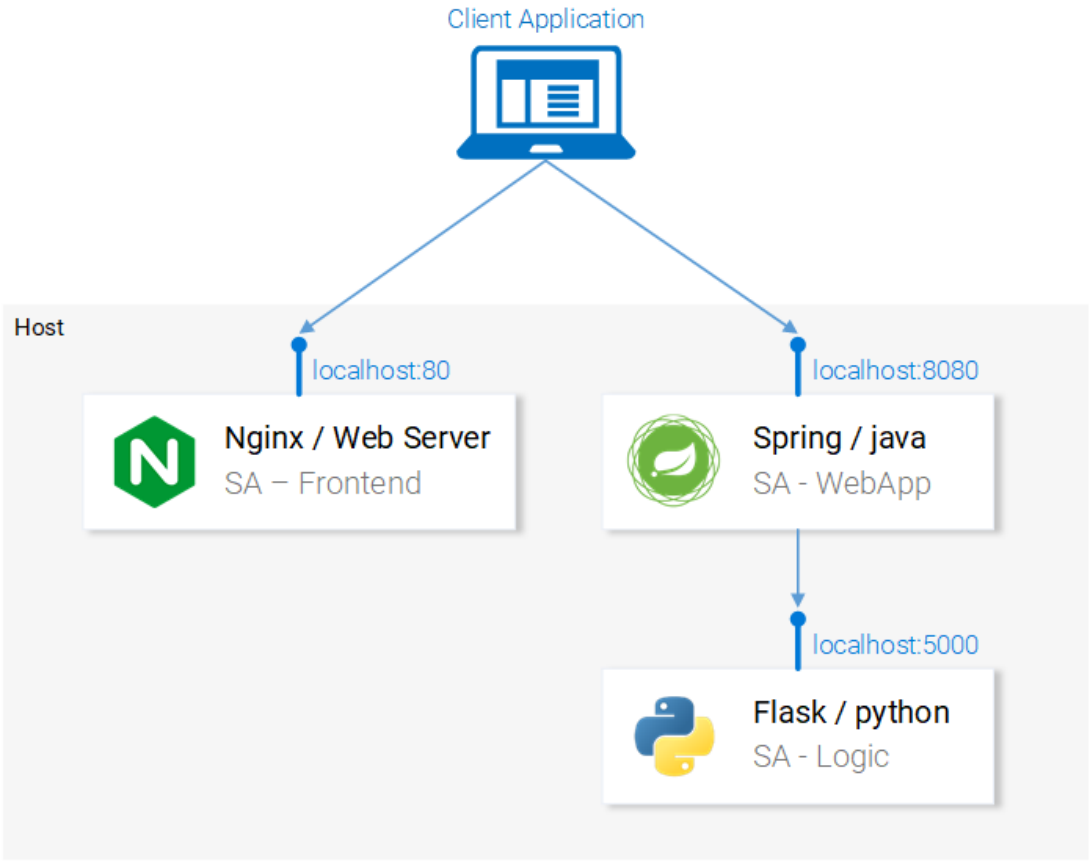

Figura 18 – Fluxo da aplicação durante as requisições. Fonte: [freeCodeCamp 2023].

4.7 Deploy GKE

O Kubernetes é uma ferramenta de orquestração de contêineres que permite automatizar a implantação, o dimensionamento e a gestão de aplicações baseadas em contêineres. Os contêineres são pacotes de software leves e executáveis que incluem tudo o que é necessário para a execução da aplicação, conforme detalhado no referencial teórico. No Kubernetes, os contêineres são executados dentro de pods, que representam a menor unidade de execução da plataforma. Os pods podem conter um ou mais contêineres, compartilhando o mesmo espaço de rede e armazenamento, o que permite uma interação eficiente entre eles.

Para garantir o acesso aos contêineres, tanto por usuários quanto por outros pods, são utilizados recursos denominados serviços. Nesta etapa, os serviços serão configurados utilizando o tipo `LoadBalancer`, que permite a exposição da aplicação para o ambiente externo, além de realizar o balanceamento de carga entre os pods disponíveis.

4.7.1 Manifestos de Serviços Kubernetes

O manifesto responsável pela criação dos serviços no Kubernetes está disponível no Apêndice A.2.

A seguir, apresentam-se as principais seções de um manifesto de serviço no Kubernetes:

- `apiVersion`: Define a versão da API utilizada.
- `kind`: Especifica o tipo de recurso, neste caso, um serviço.
- `metadata`: Contém informações como o nome e as labels associadas ao recurso.
- `spec`: Define as configurações específicas do serviço, incluindo o tipo de serviço, as portas utilizadas e o seletor de pods.
- `ports`: Especifica as portas expostas pelo serviço, indicando a porta de entrada e a porta de destino no pod.
- `selector`: Relaciona o serviço aos pods, utilizando as labels configuradas.

4.7.2 Conexão com o Cluster

Após a criação dos manifestos de serviço, é necessário estabelecer a conexão com o cluster GKE. Esta etapa é realizada por meio do comando `gcloud container clusters get-credentials`, disponível na opção 8 do menu do código. Esse comando recupera as credenciais necessárias para autenticação e acesso ao cluster, configurando o contexto da ferramenta `kubectl` para a execução dos comandos de gerenciamento de recursos.

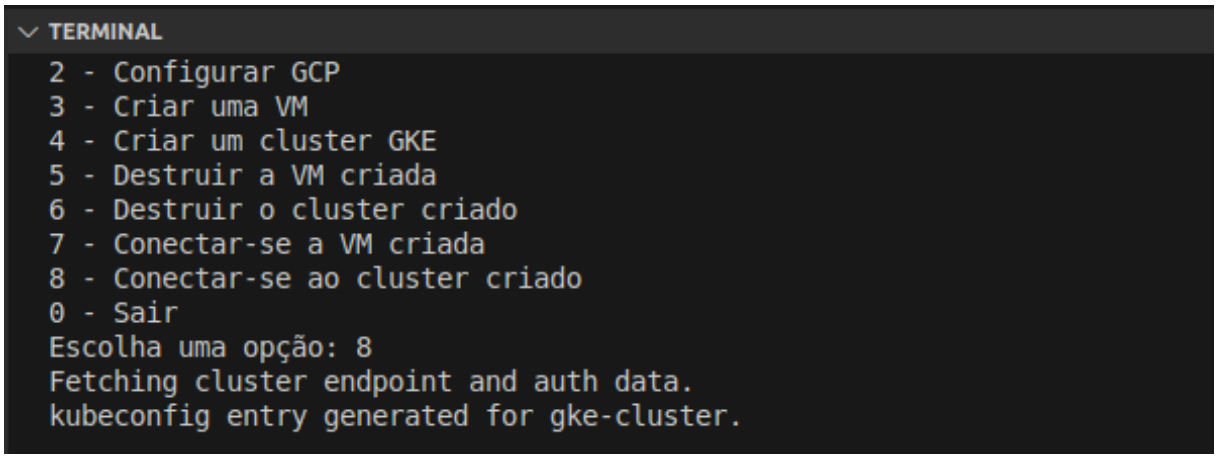

Figura 19 – Tela do terminal após conexão com o cluster GKE. Fonte: Elaborado pelo autor.

Com a conexão estabelecida, os manifestos de serviço são aplicados utilizando o comando:

```
kubectl apply -f <nome-do-arquivo>.yaml
```

Os arquivos de manifesto utilizados são: service-sa-frontend-lb.yaml, service-sa-logic.yaml e service-sa-web-app-lb.yaml.

Após a aplicação dos manifestos, a execução do comando:

```
kubectl get services
```

permite verificar a criação dos serviços e a atribuição dos IPs externos, conforme apresentado na Figura 20.

```
→  resource-manifests git:(master) ✗ kubectl get svc -n default
NAME            TYPE           CLUSTER-IP       EXTERNAL-IP      PORT(S)        AGE
kubernetes      ClusterIP      34.118.224.1     <none>           443/TCP        107m
sa-frontend-lb  LoadBalancer   34.118.233.78    35.239.109.186   80:32499/TCP   37m
sa-logic        ClusterIP      34.118.236.209   <none>           80/TCP         37m
sa-web-app-lb   LoadBalancer   34.118.236.30    35.193.28.105    80:30553/TCP   37m
```

Figura 20 – Serviços provisionados no GKE. Fonte: Elaborado pelo autor.

4.7.3 Geração dos Contêineres da Aplicação

As etapas a seguir descrevem a criação das imagens dos contêineres da aplicação e o respectivo envio para o Docker Hub.

4.7.3.1 Contêiner Frontend

Primeiramente, é realizado o ajuste do IP de comunicação entre o frontend e o backend, conforme executado na implantação via Compute Engine. Na sequência, o comando `npm run build` gera a pasta `build` com os apontamentos corretos.

A imagem do contêiner é criada utilizando a imagem base oficial do NGINX, conforme definido no Apêndice A.3. A pasta build é copiada para o diretório /usr/share/nginx/html, garantindo que o frontend seja disponibilizado corretamente.

Os comandos utilizados são:

```
docker build -f Dockerfile -t paulov1997/sentiment-analysis-frontend:v4
docker push paulov1997/sentiment-analysis-frontend:v4
```

4.7.3.2 Contêiner Web-App

A imagem do backend é gerada utilizando a base `OpenJDK 8 Alpine`, que oferece um ambiente otimizado e enxuto. O Dockerfile completo encontra-se no Apêndice A.4.

A criação e publicação da imagem são realizadas por meio dos comandos:

```
docker build -f Dockerfile -t paulov1997/sentiment-analysis-web:v1
docker push paulov1997/sentiment-analysis-web:v1
```

4.7.3.3 Contêiner de Análise de Sentimentos

A imagem responsável pela análise de sentimentos é criada com base na imagem `Python 3.6-slim`, garantindo um ambiente leve para execução da aplicação. O Dockerfile correspondente está disponível no Apêndice A.5.

Os comandos de criação e envio da imagem são:

```
docker build -f Dockerfile -t paulov1997/sentiment-analysis-logic:v1
docker push paulov1997/sentiment-analysis-logic:v1
```

4.7.4 Deploy dos Contêineres no Kubernetes

Com as imagens armazenadas no Docker Hub, procede-se com o deploy no Kubernetes por meio dos manifestos de deployment, disponíveis no Apêndice A.6.

Os principais parâmetros dos manifestos são:

- `apiVersion` e `kind`: Definem o tipo de recurso, neste caso, `Deployment`.
- `replicas`: Define a quantidade de pods a serem executados.
- `strategy` e `RollingUpdate`: Configuram a estratégia de atualização das instâncias.
- `template`: Define o modelo de pod a ser criado, incluindo as imagens dos contêineres.
- `imagePullPolicy`: Especifica a política de atualização das imagens.

A aplicação dos manifestos é realizada com o comando:

```
kubectl apply -f <nome-do-arquivo>.yaml
```

Após a execução dos comandos, a verificação do estado dos deployments é realizada por meio do comando:

```
kubectl get deployment -n default
```

A Figura 21 apresenta os deployments com status `READY`.

```
→  resource-manifests git:(master) ✗ kubectl get deployment -n default
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
sa-frontend   1/1     1            1           21h
sa-logic      1/1     1            1           21h
sa-web-app    1/1     1            1           21h
→  resource-manifests git:(master) ✗
```

Figura 21 – Deployments prontos no GKE. Fonte: Elaborado pelo autor.

A aplicação torna-se disponível externamente, conforme ilustrado na Figura 22.

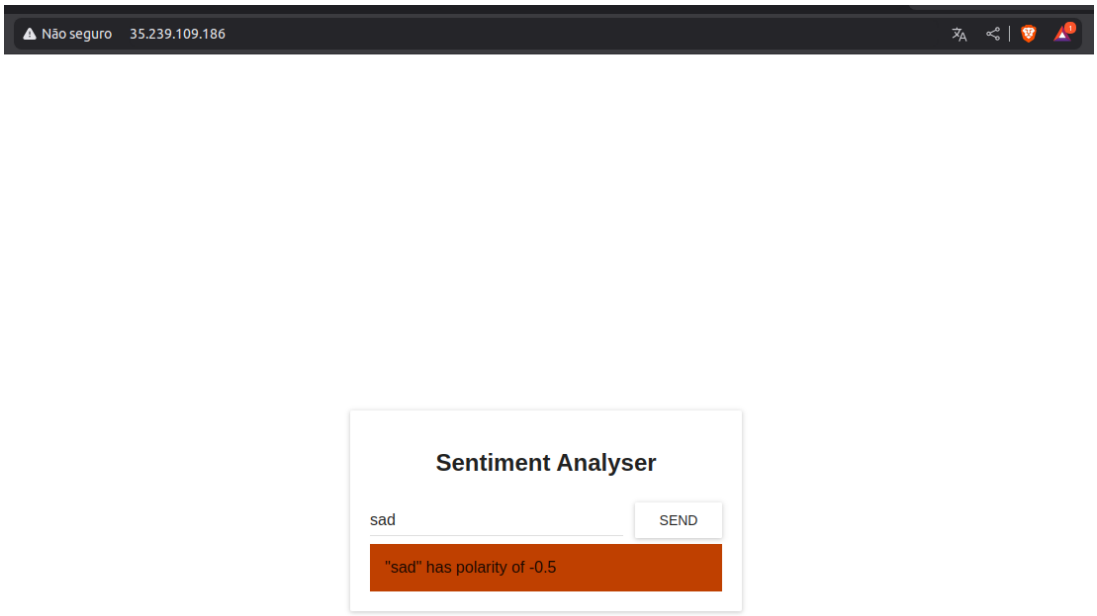

Figura 22 – Aplicação acessível pela internet após deploy no GKE. Fonte: Elaborado pelo autor.

4.7.5 Fluxo das Requisições no GKE

O fluxo das requisições segue as etapas descritas a seguir:

- **Recebimento da Requisição**: As requisições são recebidas pelos Load Balancers `sa-frontend-lb` e `sa-web-app-lb`.
- **Encaminhamento para o Frontend**: O serviço `sa-frontend` recebe a requisição e a redireciona para o pod correspondente.
- **Interação com Backend e Serviço de Lógica**: Os serviços `sa-web-app` e `sa-logic` processam as requisições, garantindo a correta execução da aplicação.

A Figura 23 apresenta a representação gráfica do fluxo de requisições.

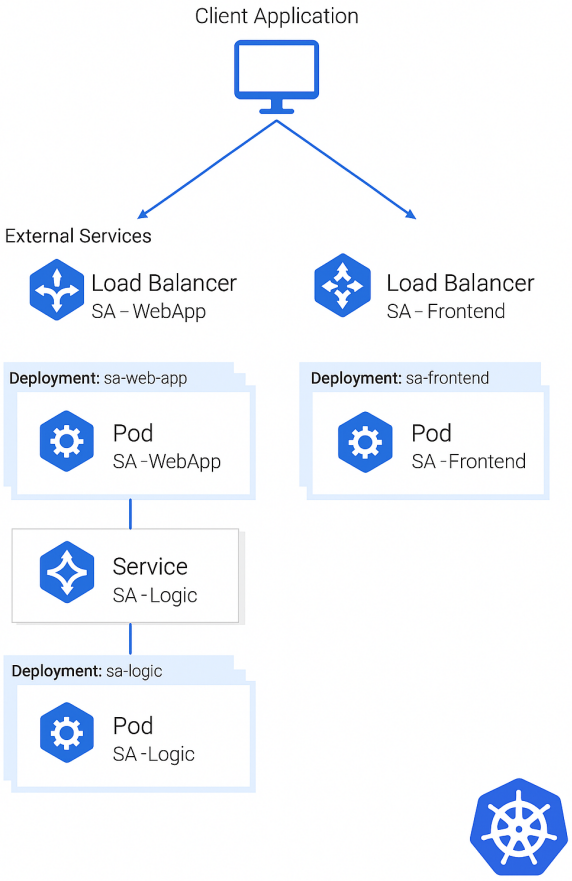

Figura 23 – Fluxo das requisições no GKE. Fonte: [freeCodeCamp 2023].

4.8 Exclusão de VM e Container

Com o objetivo de facilitar a remoção dos recursos criados e garantir a limpeza do ambiente de forma eficiente e automatizada, foram implementadas as opções 5 e 6 no menu do código. Essas opções eliminam a necessidade de intervenções manuais, reduzindo a probabilidade de erros e agilizando o processo de liberação dos recursos.

A opção 5 é responsável pela exclusão da instância de máquina virtual (VM) no Compute Engine, enquanto a opção 6 realiza a exclusão do cluster Kubernetes (GKE). Ambas as opções utilizam comandos automatizados para garantir que a remoção dos recursos ocorra de forma segura e completa.

A Figura 24 apresenta a execução da exclusão da VM por meio da opção 5.

```
Plan: 0 to add, 0 to change, 4 to destroy.
google_compute_firewall.ssh_rule: Destroying... [id=projects/tcc-paulo-v2/global/firewalls/allow-ssh]
google_compute_instance.default: Destroying... [id=projects/tcc-paulo-v2/zones/us-central1-c/instances/flask-vm]
google_compute_firewall.ssh_rule: Still destroying... [id=projects/tcc-paulo-v2/global/firewalls/allow-ssh, 10s elapsed]
google_compute_instance.default: Still destroying... [id=projects/tcc-paulo-v2/zones/us-central1-c/instances/flask-vm, 10s elapsed]
google_compute_firewall.ssh_rule: Destruction complete after 12s
google_compute_instance.default: Still destroying... [id=projects/tcc-paulo-v2/zones/us-central1-c/instances/flask-vm, 20s elapsed]
google_compute_instance.default: Still destroying... [id=projects/tcc-paulo-v2/zones/us-central1-c/instances/flask-vm, 30s elapsed]
google_compute_instance.default: Still destroying... [id=projects/tcc-paulo-v2/zones/us-central1-c/instances/flask-vm, 40s elapsed]
google_compute_instance.default: Still destroying... [id=projects/tcc-paulo-v2/zones/us-central1-c/instances/flask-vm, 50s elapsed]
google_compute_instance.default: Still destroying... [id=projects/tcc-paulo-v2/zones/us-central1-c/instances/flask-vm, 1m0s elapsed]
google_compute_instance.default: Destruction complete after 1m8s
google_compute_subnetwork.default: Destroying... [id=projects/tcc-paulo-v2/regions/us-central1/subnetworks/my-custom-subnet]
google_compute_subnetwork.default: Still destroying... [id=projects/tcc-paulo-v2/regions/us-central1/subnetworks/my-custom-subnet, 10s elapsed]
google_compute_subnetwork.default: Still destroying... [id=projects/tcc-paulo-v2/regions/us-central1/subnetworks/my-custom-subnet, 20s elapsed]
google_compute_subnetwork.default: Destruction complete after 23s
google_compute_network.vpc_network: Destroying... [id=projects/tcc-paulo-v2/global/networks/my-custom-mode-network]
google_compute_network.vpc_network: Still destroying... [id=projects/tcc-paulo-v2/global/networks/my-custom-mode-network, 10s elapsed]
google_compute_network.vpc_network: Still destroying... [id=projects/tcc-paulo-v2/global/networks/my-custom-mode-network, 20s elapsed]
google_compute_network.vpc_network: Destruction complete after 22s

Destroy complete! Resources: 4 destroyed.
```

Figura 24 – Tela do terminal após a execução da opção 5 do menu. Fonte: Elaborado pelo autor.

De forma similar, a Figura 25 ilustra o processo de exclusão do cluster Kubernetes, realizado por meio da opção 6.

```
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 2m10s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 2m20s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 2m30s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 2m40s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 2m50s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 3m0s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 3m10s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 3m20s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 3m30s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 3m40s elapsed]
google_container_cluster.gke_cluster: Still destroying... [id=projects/tcc-paulo-v2/locations/us-central1-c/clusters/gke-cluster, 3m50s elapsed]
google_container_cluster.gke_cluster: Destruction complete after 3m56s
google_compute_subnetwork.subnet_gke: Destroying... [id=projects/tcc-paulo-v2/regions/us-central1/subnetworks/gke-subnet]
google_compute_subnetwork.subnet_gke: Still destroying... [id=projects/tcc-paulo-v2/regions/us-central1/subnetworks/gke-subnet, 10s elapsed]
google_compute_subnetwork.subnet_gke: Destruction complete after 14s
google_compute_network.vpc_network: Destroying... [id=projects/tcc-paulo-v2/global/networks/gke-vpc]
google_compute_network.vpc_network: Still destroying... [id=projects/tcc-paulo-v2/global/networks/gke-vpc, 10s elapsed]
google_compute_network.vpc_network: Still destroying... [id=projects/tcc-paulo-v2/global/networks/gke-vpc, 20s elapsed]
google_compute_network.vpc_network: Destruction complete after 22s

Destroy complete! Resources: 5 destroyed.
```

Figura 25 – Tela do terminal após a execução da opção 6 do menu. Fonte: Elaborado pelo autor.

Essas funcionalidades garantem que o ambiente seja restaurado ao seu estado inicial, sem a necessidade de remoção manual dos recursos via console do Google Cloud Platform, promovendo maior praticidade e controle durante o ciclo de vida dos recursos provisionados.

4.9 Tabela de Comparação

Com o objetivo de apresentar de forma clara os resultados obtidos nos testes realizados, a Tabela 1 exibe a comparação das principais métricas observadas durante o processo de deploy nos ambientes Compute Engine (VM) e Kubernetes (GKE).

Para garantir a consistência dos resultados, os ambientes foram provisionados de forma automatizada. No ambiente de máquina virtual, foi utilizado um script `.sh` para automatização do deploy, enquanto no ambiente Kubernetes as imagens e os manifestos de serviços e deployments foram previamente configurados.

Métricas	**VM**	**GKE**
Tempo de Provisionamento	180 segundos	680 segundos
Tempo de Deploy	1380 segundos	120 segundos
Tempo de Requisição HTTP	307 ms	242 ms
Redundância	Não	Sim
Pico de CPU	86%	74%
Escalabilidade e Manutenção	Não	Sim
Tempo de Exclusão	115 segundos	420 segundos

Tabela 1 – Tabela de Comparação entre os ambientes VM e GKE

As métricas apresentadas foram avaliadas conforme os seguintes critérios:

- Tempo de Provisionamento: Refere-se ao tempo necessário para provisionar os recursos em cada ambiente.
- Tempo de Deploy: Representa o tempo necessário para realizar o deploy da aplicação após o provisionamento.
- Tempo de Requisição HTTP: Mede a latência de uma requisição HTTP média nos dois ambientes, refletindo a eficiência de cada um.
- Redundância: Indica a capacidade do ambiente de fornecer alta disponibilidade através de múltiplas instâncias.
- Pico de CPU: O maior uso de CPU observado durante os testes.
- Escalabilidade e Manutenção: Avalia a capacidade de aumentar ou diminuir a carga automaticamente e a facilidade de manutenção de cada ambiente.
- Tempo de Exclusão: O tempo necessário para excluir recursos e retornar ao estado original.

5 Resultados

Nesta seção, serão apresentados os resultados obtidos da comparação entre as abordagens de deploy utilizadas para o ambiente do projeto: deploy em VM no Google Cloud Platform (Compute Engine) e deploy utilizando Kubernetes no Google Kubernetes Engine (GKE). A análise será focada nos seguintes aspectos principais: tempo de provisionamento, tempo de deploy,tempo da requisição http , redes e redundância, escalabilidade e manutenção e uso de recursos.

5.1 Tempo de Provisionamento

O tempo de provisionamento refere-se ao intervalo necessário para criar e configurar os ambientes de computação para o deploy das aplicações. No Compute Engine (VM), o tempo de provisionamento foi de 180 segundos, enquanto no GKE, o tempo registrado foi de 680 segundos. Observa-se que, para ambos os casos, foram utilizadas máquinas preemptivas do free-tier, cujas instâncias não têm garantia de disponibilidade contínua [Google 2023]. Essa característica influencia diretamente no tempo de provisionamento devido à eventual indisponibilidade dessas máquinas no momento da solicitação

Diferença no tempo de provisionamento = 680 - 180 = 500 segundos

Essa diferença de 500 segundos pode ser atribuída a alguns fatores distintos entre as duas abordagens. Primeiramente, o Compute Engine requer apenas a criação de uma instância virtual simples, o que facilita o processo de provisionamento. Em contraste, o GKE envolve uma configuração mais complexa, que inclui a criação e o gerenciamento de um cluster Kubernetes. Este cluster requer a alocação de múltiplos nós, a definição de redes e a configuração de políticas de escalabilidade automática, o que naturalmente resulta em um tempo maior de provisionamento.

Além disso, a arquitetura de containerização do GKE permite um gerenciamento mais avançado de aplicações, facilitando o escalonamento e a alta disponibilidade. Contudo, esses recursos exigem a preparação e a configuração do cluster, o que aumenta o tempo necessário para disponibilizar o ambiente, especialmente se comparado ao provisionamento mais simples oferecido pelo Compute Engine.

5.2 Tempo de Deploy

O tempo de deploy, que corresponde ao tempo necessário para implementar a aplicação em cada ambiente, foi significativamente mais rápido no GKE 120 segundos do

que no Compute Engine (1380 segundos). A principal razão para essa diferença é que o GKE oferece uma plataforma altamente otimizada para o gerenciamento de containers, com ferramentas como o kubectl, que simplificam e automatizam o processo de deploy. No GKE, após a criação do container, as dependências já estão encapsuladas, permitindo que o deploy seja realizado diretamente com o comando `kubectl apply -f`, disponibilizando rapidamente a aplicação.

Por outro lado, no Compute Engine, o processo de deploy é mais complexo e manual. Embora seja possível automatizar a criação da máquina virtual utilizando ferramentas como o Terraform, a instalação manual de pacotes essenciais (como Node.js, Java e Python) e suas respectivas dependências gera um tempo de deploy consideravelmente mais longo. Isso ocorre porque, além da configuração da infraestrutura da VM, é necessário configurar e instalar todos os componentes da aplicação individualmente em cada instância, o que resulta em uma maior sobrecarga.

Diferença no tempo de deploy = 1380 - 120 = 1260 segundos

Essa diferença de 1260 segundos reflete a eficiência do GKE em processos de deploy, especialmente para aplicativos que exigem múltiplos containers e escalabilidade automática com base na carga.

5.3 Tempo de Requisição HTTP

Ao compararmos o tempo de requisição HTTP entre o ambiente de Compute Engine e o ambiente de GKE, não observamos uma diferença significativa no tempo de resposta. Durante nossos testes, utilizamos requisições simples via navegador (Chrome DevTools), não realizando stress tests devido às limitações do ambiente gratuito e tempo disponível para execução dos testes. Como resultado, não conseguimos evidenciar de forma clara os benefícios de escalabilidade proporcionados pelos pods no GKE em relação ao Compute Engine.

Diferença no tempo http = 307 - 242 = 65 milissegundos de segundo

Nos testes realizados, observou-se uma diferença média de 65 ms indicando uma pequena vantagem para o ambiente GKE. Contudo, devido às limitações dos testes realizados com requisições via navegador (Chrome DevTools), essa diferença não evidencia claramente os benefícios da escalabilidade do GKE."

5.4 Redundância e Alta Disponibilidade

Um dos aspectos mais destacados do GKE é a sua capacidade de fornecer redundância e alta disponibilidade de forma integrada. Em um cluster Kubernetes, os pods são

distribuídos automaticamente entre vários nós, o que garante a disponibilidade contínua dos serviços, mesmo que um nó falhe. No entanto, o Compute Engine não oferece essa redundância por padrão, sendo necessário configurar manualmente replicação de instâncias ou balanceamento de carga para alcançar a alta disponibilidade.

No GKE, o gerenciamento de falhas e a recuperação automática ocorrem de forma integrada, o que reduz o risco de downtime e a sobrecarga operacional. Já no Compute Engine, é necessário realizar um esforço adicional para configurar essas redundâncias, o que torna o processo mais complexo e suscetível a falhas.

5.5 Escalabilidade e Manutenção

No aspecto de escalabilidade, o GKE se destaca por sua capacidade de escalabilidade automática. O Kubernetes ajusta o número de pods e nós automaticamente com base nas necessidades de carga, garantindo que os recursos sejam alocados de maneira eficiente e econômica. O Compute Engine, por outro lado, exige que a escalabilidade seja gerenciada manualmente ou por meio de scripts de automação, o que aumenta a complexidade e o tempo gasto na manutenção.

A manutenção também é um ponto forte do GKE, pois a plataforma permite atualizações automáticas e a possibilidade de realizar rolling updates, ou seja, atualizações de versões sem afetar a disponibilidade dos serviços. No Compute Engine, as atualizações e a manutenção precisam ser feitas de forma manual, e qualquer falha pode resultar em períodos de inatividade.

5.6 Uso de Recursos (CPU e Memória)

Ao comparar o uso de CPU, observamos que o Compute Engine teve um pico de utilização de 86%, enquanto o GKE atingiu 74%. Embora o Compute Engine tenha utilizado mais CPU, o GKE apresenta uma melhor alocação de recursos, garantindo maior eficiência no uso de memória e CPU ao distribuir as cargas de trabalho de forma mais equilibrada entre os pods.

O GKE se beneficia de sua capacidade de orquestrar contêineres e gerenciar automaticamente a alocação de recursos, otimizando o uso de CPU e memória. Já o Compute Engine requer uma configuração manual do uso de recursos, o que pode levar a um consumo ineficiente, especialmente em ambientes que requerem alta disponibilidade e resiliência.

5.7 Tempo de Exclusão

No que diz respeito ao tempo de exclusão, o Compute Engine teve um tempo de 115 segundos para excluir a instância, enquanto o GKE levou 420 segundos. O Compute Engine executa o desligamento de instâncias de maneira mais direta, enquanto o GKE exige a exclusão de múltiplos recursos, como pods, serviços e volumes, o que torna o processo mais demorado.

Essa diferença reflete a complexidade e a necessidade de manter a integridade do cluster Kubernetes, mesmo durante o processo de exclusão de recursos.

6 Conclusão

O objetivo deste trabalho foi desenvolver uma solução prática e acessível para o uso de recursos em nuvem por meio da Infraestrutura como Código (IaC). Durante a graduação, foi observado que, em disciplinas como Projeto de Desenvolvimento de Sistemas de Informação I (PDSI1), a falta de conhecimentos sobre computação em nuvem e automação de infraestrutura dificultava a realização de deploys e o acesso simultâneo às aplicações desenvolvidas pelos grupos. Na prática, cada integrante testava a aplicação apenas em ambiente local (*localhost*), uma vez que não havia facilidade para disponibilizá-la de forma pública na internet. Esse cenário limitava a colaboração entre os membros da equipe e dificultava a validação conjunta das soluções desenvolvidas. A partir desse contexto, a proposta apresentada neste trabalho buscou justamente superar essas limitações, demonstrando como o uso da nuvem pode simplificar o deploy e facilitar o acesso compartilhado às aplicações.

A partir dessa necessidade, foi implementada uma solução que permite a qualquer usuário, mesmo com conhecimentos básicos, o acesso a recursos gratuitos em nuvem. Utilizou-se a linguagem *Python* para a preparação do ambiente local, *Terraform* para a automação do provisionamento de infraestrutura, e *Docker* e *Kubernetes* para o gerenciamento e deploy das aplicações.

A solução proposta foi aplicada a um sistema de análise de sentimentos, demonstrando a viabilidade do uso de ambientes em nuvem para aplicações reais. Durante a implantação nos ambientes **Compute Engine** e **GKE**, verificou-se que cada plataforma apresenta vantagens específicas de acordo com o cenário de uso. O **Compute Engine** mostrou-se mais adequado para cargas de trabalho simples, que não demandam alta disponibilidade ou escalabilidade dinâmica. Por outro lado, o **GKE** destacou-se pela facilidade de manutenção, escalabilidade automática e recursos de alta disponibilidade, embora seu tempo de provisionamento inicial seja superior.

A escolha entre as duas soluções deve considerar o perfil da aplicação e os requisitos de disponibilidade e escalabilidade. Com o uso do Free Tier do **GCP**, foi possível experimentar ambas as abordagens e realizar uma análise comparativa baseada em métricas reais de desempenho e consumo de recursos.

Como proposta de continuidade, recomenda-se a ampliação da solução para suportar outros sistemas operacionais, como o Windows, além da utilização dos créditos promocionais de $300 oferecidos pelo GCP, possibilitando a avaliação de ambientes mais robustos. Também se sugere a implementação de suporte a múltiplos provedores de nuvem, tornando a solução compatível com ambientes *multi-cloud*.

Adicionalmente, este trabalho apresenta forte correlação com as disciplinas do curso de Sistemas de Informação, como Engenharia de Software, Redes de Computadores, Banco de Dados e Arquitetura de Sistemas Distribuídos. Os conhecimentos adquiridos nessas áreas foram fundamentais para a construção da solução proposta, abrangendo desde o planejamento e desenvolvimento de sistemas até a implementação de ambientes computacionais escaláveis e seguros.

Dessa forma, este projeto contribui para a democratização do acesso a tecnologias de computação em nuvem, promovendo a adoção de práticas modernas de automação de infraestrutura e aproximando o ambiente acadêmico das demandas do mercado de trabalho.

Referências

Amazon Web Services. **Announcing Amazon S3 - Simple Storage Service**. 2006. Acessado em: 5 abr. 2025. Disponível em: <https://aws.amazon.com/about-aws/whats-new/2006/03/13/announcing-amazon-s3---simple-storage-service/>. Citado na página 15.

_____. **Happy 15th Birthday Amazon EC2**. 2021. Acessado em: 5 abr. 2025. Disponível em: <https://aws.amazon.com/blogs/aws/happy-15th-birthday-amazon-ec2/>. Citado na página 15.

_____. **O que é computação em nuvem?** 2024. Acessado em: 5 abr. 2025. Disponível em: <https://aws.amazon.com/pt/what-is-cloud-computing/>. Citado na página 15.

BRIKMAN, Y. **Terraform: Up and Running: Writing Infrastructure as Code**. 3. ed. O'Reilly Media, 2022. ISBN 978-1-098-15034-1. Disponível em: <https://learning.oreilly.com/library/view/terraform-up-and/9781098150341/>. Citado 4 vezes nas páginas 5, 25, 26 e 39.

BURNS, B.; BEDA, J.; HIGHTOWER, K. **Kubernetes: Up and Running: Dive into the Future of Infrastructure**. 3rd. ed. O'Reilly Media, 2022. ISBN 9781098110192. Disponível em: <https://learning.oreilly.com/library/view/kubernetes-up-and/9781098110192/>. Citado 2 vezes nas páginas 22 e 23.

CLOUD, G. **Create and manage service accounts**. 2025. <https://cloud.google.com/docs/authentication/getting-started>. Accessed: 2025-04-27. Citado na página 37.

_____. **Enable APIs and Services**. 2025. <https://cloud.google.com/apis/docs/dashboard>. Accessed: 2025-04-27. Citado na página 38.

_____. **Project ID and project number**. 2025. <https://cloud.google.com/docs/overview#project-id>. Accessed: 2025-04-27. Citado na página 37.

DEVELOPER, I. **Containerization of Legacy Applications**. 2021. Acessado em: 27 abr. 2025. Disponível em: <https://developer.ibm.com/articles/containerization-of-legacy-applications/>. Citado na página 27.

DIGITALOCEAN. **Comparing AWS, Azure, GCP**. 2023. Disponível em: <https://www.digitalocean.com/resources/articles/comparing-aws-azure-gcp>. Citado na página 12.

freeCodeCamp. **DevOps Deployment Pipeline Diagram**. 2023. <https://cdn-media-1.freecodecamp.org/images/JwIBwPsTfBmelKgSrCCkEZuTzC5Ty1pZi3K7>. Acessado em: 1 maio 2025. Citado 4 vezes nas páginas 5, 6, 51 e 56.

Gartner. **Top Amazon Web Services Competitors & Alternatives**. 2025. Acessado em: 6 abr. 2025. Disponível em: <https://www.gartner.com/reviews/market/strategic-cloud-platform-services/vendor/amazon-web-services/product/amazon-web-services/alternatives>. Citado na página 17.

GOOGLE. **gcloud compute ssh**. 2023. Acessado em: 27 abr. 2025. Disponível em: <https://cloud.google.com/sdk/gcloud/reference/compute/ssh>. Citado na página 48.

_____. **Preemptible VMs | Google Cloud**. 2023. Accessed: 2025-04-30. Disponível em: <https://cloud.google.com/preemptible-vms>. Citado na página 59.

Google App Engine Team. **Introducing Google App Engine: our new cloud computing platform**. 2008. Acessado em: 6 abr. 2025. Disponível em: <https://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html>. Citado na página 17.

Google Cloud. **Strengthening the G Suite ecosystem**. 2015. Acessado em: 5 abr. 2025. Disponível em: <https://cloud.google.com/blog/topics/inside-google-cloud/strengthening-the-google-apps-ecosystem_3>. Citado na página 15.

_____. **Always Free Usage Limits**. 2024. Acesso em: 16 abr. 2025. Disponível em: <https://cloud.google.com/free>. Citado 2 vezes nas páginas 13 e 18.

_____. **Compute Engine Free Tier Overview**. 2024. Acesso em: 16 abr. 2025. Disponível em: <https://cloud.google.com/free/docs/compute-getting-started>. Citado na página 18.

_____. **Google Kubernetes Engine Pricing**. 2024. Acesso em: 16 abr. 2025. Disponível em: <https://cloud.google.com/kubernetes-engine/pricing>. Citado na página 18.

_____. **Compute Engine - Google Cloud**. 2025. Acessado em: 6 abr. 2025. Disponível em: <https://cloud.google.com/products/compute>. Citado na página 17.

_____. **Google Kubernetes Engine (GKE)**. 2025. Acessado em: 6 abr. 2025. Disponível em: <https://cloud.google.com/kubernetes-engine>. Citado na página 18.

_____. **O que é computação na nuvem?** 2025. Acessado em: 5 abr. 2025. Disponível em: <https://cloud.google.com/learn/what-is-cloud-computing?hl=pt-BR>. Citado na página 16.

_____. **Produtos do Google Cloud**. 2025. Acessado em: 6 abr. 2025. Disponível em: <https://cloud.google.com/products>. Citado na página 17.

_____. **Quickstart: Install the Google Cloud CLI**. 2025. Acessado em: 6 abr. 2025. Disponível em: <https://cloud.google.com/cli?hl=pt-br>. Citado na página 18.

Google Official Blog. **Get your people talking**. 2006. Acessado em: 6 abr. 2025. Disponível em: <https://googleblog.blogspot.com/2006/08/get-your-people-talking.html>. Citado na página 17.

HASHICORP. **What is Infrastructure as Code with Terraform?** 2023. Disponível em: <https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code>. Citado na página 12.

HIGHTOWER, K.; BURNS, B.; BEDA, J. **Kubernetes: Up & Running: Dive into the Future of Infrastructure**. 3. ed. Sebastopol: O'Reilly Media, 2022. ISBN 9781098110208. Citado na página 22.

IEEE Spectrum. **The Top Programming Languages 2024**. 2024. <https://spectrum.ieee.org/top-programming-languages-2024>. Acesso em: 09 abr. 2025. Citado na página 18.

INSTITUTE, D. **2023 Upskilling IT Report**. 2023. <https://www.devopsinstitute.com/upskilling-it-2023/>. Acesso em: 21 abr. 2025. Citado na página 13.

Internet Hall of Fame. **JCR Licklider**. 2025. Acessado em: 30 mar. 2025. Disponível em: <https://www.internethalloffame.org/inductee/jcr-licklider/>. Citado na página 15.

JASANI, H.; PATEL, J.; DOSHI, H. Exploring infrastructure as code using terraform in multi-cloud deployments. **SSRN Electronic Journal**, 2024. Acessado em: 27 abr. 2025. Disponível em: <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5075647>. Citado na página 27.

KANE, S. P.; MATTHIAS, K. **Docker: Up & Running: Shipping Reliable Containers in Production**. 3. ed. O'Reilly Media, 2023. Acesso em: 20 abr. 2025. ISBN 9781098131821. Disponível em: <https://learning.oreilly.com/library/view/docker-up/9781098131814/preface01.html\#id301>. Citado 4 vezes nas páginas 5, 20, 21 e 22.

KHAJEH-HOSSEINI, A.; GREENWOOD, D.; SOMMERVILLE, I. Cloud migration: A case study of migrating an enterprise it system to iaas. **arXiv preprint arXiv:1002.3492**, 2010. Acessado em: 27 abr. 2025. Disponível em: <https://arxiv.org/abs/1002.3492>. Citado na página 27.

KUBERNETS.IO. **Components of Kubernetes**. 2025. <https://kubernetes.io/images/docs/components-of-kubernetes.svg>. Acessado em: 08/05/2025. Citado 3 vezes nas páginas 5, 23 e 24.

Microsoft Azure. **What is Azure?** 2021. Acessado em: 5 abr. 2025. Disponível em: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-azure>. Citado na página 15.

_____. **O que é nuvem – definição**. 2025. Acessado em: 5 abr. 2025. Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-the-cloud>. Citado na página 15.

MORRIS, K. **Infrastructure as Code: Managing Servers in the Cloud**. O'Reilly Media, 2016. Acesso em: 10 abr. 2025. Disponível em: <https://www.amazon.com/Infrastructure-Code-Managing-Servers-Cloud/dp/1491924357>. Citado na página 20.

_____. **Infrastructure as Code: Dynamic Systems for the Cloud Age**. 2. ed. O'Reilly Media, 2021. Acessado em: 09 abr. 2025. Disponível em: <https://www.oreilly.com/library/view/infrastructure-as-code/9781098150341/>. Citado na página 20.

NETWORKS, P. A. **Kubernetes and Infrastructure as Code**. 2023. Disponível em: <https://www.paloaltonetworks.com/cyberpedia/kubernetes-infrastructure-as-code>. Citado na página 12.

Python Software Foundation. **FAQ Geral do Python**. 2025. Acessado em: 8 abr. 2025. Disponível em: <https://docs.python.org/pt-br/3.8/faq/general.html>. Citado na página 18.

_____. **json — JSON encoder and decoder**. 2025. Acessado em: 8 abr. 2025. Disponível em: <https://docs.python.org/3/library/json.html>. Citado na página 19.

_____. **os — Miscellaneous operating system interfaces**. 2025. Acessado em: 8 abr. 2025. Disponível em: <https://docs.python.org/3/library/os.html>. Citado na página 19.

_____. **platform — Access to underlying platform's identifying data**. 2025. Acessado em: 8 abr. 2025. Disponível em: <https://docs.python.org/3/library/platform.html>. Citado na página 19.

_____. **shutil — High-level file operations**. 2025. Acessado em: 8 abr. 2025. Disponível em: <https://docs.python.org/3/library/shutil.html>. Citado na página 19.

_____. **subprocess — Subprocess management**. 2025. Acessado em: 8 abr. 2025. Disponível em: <https://docs.python.org/3/library/subprocess.html>. Citado na página 19.

_____. **sys — System-specific parameters and functions**. 2025. Acessado em: 8 abr. 2025. Disponível em: <https://docs.python.org/3/library/sys.html>. Citado na página 19.

REVIEWNPREP. **AWS, Azure, GCP - How To Start Cloud Learning With Free Credits**. 2022. Disponível em: <https://reviewnprep.com/blog/aws-azure-gcp-how-to-start-cloud-learning-with-free-credits/>. Citado na página 12.

SOKOLOWSKI, D. Infrastructure as code for dynamic deployments. In: **Proceedings of the 24th Annual International Conference on Information Technology**. [s.n.], 2022. Acessado em: 27 abr. 2025. Disponível em: <https://dl.acm.org/doi/abs/10.1145/3540250.3558912>. Citado na página 26.

TALBOT, D. **The Cloud Imperative**. 2011. MIT Technology Review, acesso em 30 mar. 2025. Disponível em: <https://www.technologyreview.com/2011/10/03/190237/the-cloud-imperative/>. Citado na página 15.

TECHNOLOGIES, D. **The History and Future of Cloud Computing**. 2011. Acessado em: 30 mar. 2025. Disponível em: <https://www.forbes.com/sites/dell/2011/12/20/the-history-and-future-of-cloud-computing/>. Citado na página 15.

TREVISO, A. **Automação do Provisionamento de Infraestrutura em Nuvem para Implantação de Sistemas**. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul, 2023. Acessado em: 27 abr. 2025. Disponível em: <https://lume.ufrgs.br/bitstream/handle/10183/251768/001153699.pdf>. Citado na página 27.

TYSON, M. **The Origins of Infrastructure as Code: A Brief History of DevOps**. 2023. Acesso em: 10 abr. 2025. Disponível em: <https://medium.com/@mike_tyson_cloud/the-origins-of-infrastructure-as-code-a-brief-history-of-devops-a883d8877f19>. Citado na página 19.

A Apêndice A - Códigos Utilizados no Trabalho

Nesta seção estão apresentados os códigos principais utilizados ao longo do trabalho. Referências a esses códigos estão presentes ao longo do texto.

A.1 Script de Deploy da Aplicação

Listing A.1 – Script de Deploy Automatizado

```
#!/bin/bash

# Verifica se o usuario e root senao reinicia o script como root
if [[ $EUID -ne 0 ]]; then
    echo "Este␣script␣precisa␣ser␣executado␣como␣root!"
    exec sudo su -c "$0"
fi

set -e  # Para o script se algum comando falhar
set -x  # Exibe os comandos enquanto sao executados

\# Obtem o IP externo da maquina
EXTERNAL_IP=$(curl -s ifconfig.me)
echo "IP␣Externo␣da␣Maquina:␣\$EXTERNAL_IP"

\# Instala dependencias basicas
apt update \&\& apt install -y git npm maven python3-pip curl openjdk
   -11-jdk

\# Clona o repositorio
cd /opt/
git clone https://github.com/rinormaloku/k8s-mastery.git
cd /opt/k8s-mastery/

\# Configurar e iniciar o frontend
cd sa-frontend
npm install
sed -i "s|http://localhost:8080/sentiment|http://\$EXTERNAL_IP:8080/
   sentiment|g" src/App.js
npm start \&

\# Configurar e iniciar o backend (WebApp)
cd /opt/k8s-mastery/sa-webapp/
```

```
mvn install
java -jar target/sentiment-analysis-web-0.0.1-SNAPSHOT.jar --sa.logic.
    api.url=http://localhost:5000 \&

\# Configurar e iniciar a logica de analise (Python)
cd /opt/k8s-mastery/sa-logic/sa/
python3 -m pip install --upgrade Werkzeug
python3 -m pip install --upgrade Flask
python3 -m pip install textblob
python3 -m textblob.download_corpora
python3 sentiment_analysis.py \&
```

A.2 Serviços Kubernetes

Listing A.2 – service-sa-frontend-lb.yaml - Serviço Frontend

```
apiVersion: v1
kind: Service
metadata:
  name: sa-frontend-lb
spec:
  type: LoadBalancer
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: sa-frontend
```

Listing A.3 – service-sa-web-app-lb.yaml, Serviço Web-App

```
apiVersion: v1
kind: Service
metadata:
  name: sa-web-app-lb
spec:
  type: LoadBalancer
  ports:
  - port: 80
    protocol: TCP
    targetPort: 8080
  selector:
    app: sa-web-app
```

Listing A.4 – service-sa-logic.yaml, Serviço Logic

```
apiVersion: v1
```

```
kind: Service
metadata:
  name: sa-logic
spec:
  ports:
    - port: 80
      protocol: TCP
      targetPort: 5000
  selector:
    app: sa-logic
```

A.3 Manifesto Docker Front

Listing A.5 – Arquivo Dockerfile - Frontend

```
FROM nginx
COPY build /usr/share/nginx/html
```

A.4 Manifesto Docker JAVA

Listing A.6 – Arquivo Dockerfile - Web-App

```
FROM openjdk:8-jdk-alpine
# Environment Variable that defines the endpoint of sentiment-analysis
    python api.
ENV SA_LOGIC_API_URL http://localhost:5000
ADD target/sentiment-analysis-web-0.0.1-SNAPSHOT.jar /
EXPOSE 8080
CMD ["java", "-jar", "sentiment-analysis-web-0.0.1-SNAPSHOT.jar", "--sa.
    logic.api.url=${SA_LOGIC_API_URL}"]
```

A.5 Manifesto Docker Python

Listing A.7 – Arquivo Dockerfile - Analisador de Sentimento

```
FROM python:3.6-slim
COPY sa /app
WORKDIR /app
RUN pip3 install -r requirements.txt && \
    python3 -m textblob.download_corpora
EXPOSE 5000
ENTRYPOINT ["python3"]
CMD ["sentiment_analysis.py"]
```

A.6 Manifesto Deployment Kubernets

Listing A.8 – sa-frontend-deployment.yaml - Deployment Front-End

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sa-frontend
  labels:
    app: sa-frontend
spec:
  selector:
    matchLabels:
      app: sa-frontend
  replicas: 1
  minReadySeconds: 15
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: sa-frontend
    spec:
      containers:
        - image: paulov1997/sentiment-analysis-frontend:v3
          imagePullPolicy: Always
          name: sa-frontend
          ports:
            - containerPort: 80
```

Listing A.9 – sa-web-app-deployment.yaml- Deployment Web-App

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sa-web-app
  labels:
    app: sa-web-app
spec:
  selector:
    matchLabels:
      app: sa-web-app
  replicas: 1
  minReadySeconds: 15
  strategy:
    type: RollingUpdate
```

```
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: sa-web-app
    spec:
      containers:
      - image: paulov1997/sentiment-analysis-web:v1
        imagePullPolicy: Always
        name: sa-web-app
        env:
          - name: SA_LOGIC_API_URL
            value: "http://sa-logic"
        ports:
          - containerPort: 8080
```

Listing A.10 – sa-logic-deployment.yaml - Deployment Logic

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: sa-logic
  labels:
    app: sa-logic
spec:
  selector:
     matchLabels:
       app: sa-logic
  replicas: 2
  minReadySeconds: 15
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
  template:
    metadata:
      labels:
        app: sa-logic
    spec:
      containers:
        - image:  paulov1997/sentiment-analysis-logic:v1
          imagePullPolicy: Always
          name: sa-logic
          ports:
            - containerPort: 5000
```

B Apêndice B - Repositório GitHub

O código-fonte desenvolvido para este trabalho está disponível publicamente no repositório GitHub, acessível no seguinte endereço:

<https://github.com/paulov1997/finaltcc>

O repositório contém os scripts responsáveis por preparar a máquina local, realizar a autenticação com o *Google Cloud Platform* (GCP), provisionar, conectar e excluir as máquinas virtuais no ambiente da nuvem.

Para acessar o código, recomenda-se utilizar o seguinte comando:

```
git clone https://github.com/paulov1997/finaltcc.git
```