Politécnica UFRJ

PROVISIONAMENTO AUTOMÁTICO FIM-A-FIM DE *MULTICLUSTERS* KUBERNETES PARA APLICAÇÕES DE REDES 5G

Leonardo Gomes de Castro e Silva

Projeto de Graduação apresentado ao Curso de Engenharia Eletrônica e de Computação da Escola Politécnica, Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro.

Orientador: Luís Henrique Maciel Kosmalski Costa

Rio de Janeiro

Agosto de 2022

PROVISIONAMENTO AUTOMÁTICO FIM-A-FIM DE *MULTICLUSTERS* KUBERNETES PARA APLICAÇÕES DE REDES 5G

Leonardo Gomes de Castro e Silva

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO CURSO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRÔNICO E DE COMPUTAÇÃO

Autor:

Leonardo Gomes de Castro e Silva

Orientador:

Prof. Luís Henrique Maciel Kosmalski Costa, Dr.

Examinador:

Prof. Rodrigo de Souza Couto, D.Sc.

Examinador:

Prof. Marcelo Luiz Drumond Lanza, M.Sc.

Rio de Janeiro

Agosto de 2022

Silva, Leonardo Gomes de Castro e

Provisionamento Automático Fim-a-Fim de *Multiclusters* Kubernetes para Aplicações de Redes 5G / Leonardo Gomes de Castro e Silva. - Rio de Janeiro: UFRJ / Escola Politécnica, 2022.

XIX, 86p.: il. color; 29,7cm.

Orientador: Luís Henrique Maciel Kosmalski Costa.

Projeto de Graduação - UFRJ / Escola Politécnica / Engenharia Eletrônica e de Computação, 2022.

Referências bibliográficas: p.47-51.

1. Kubernetes. 2. 5G. 3. Infraestrutura como Código. I. Costa, Luís Henrique Maciel Kosmalski II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Curso de Engenharia Eletrônica e de Computação. IV. Provisionamento Automático Fim-a-Fim de *Multiclusters* Kubernetes para Aplicações de Redes 5G.

Declaração de Autoria e de Direitos

Eu, *Leonardo Gomes de Castro e Silva* CPF *026.356.352-93*, autor da monografia *Provisionamento Automático Fim-a-Fim de Multiclusters Kubernetes para Aplicações de Redes 5G*, subscrevo para os devidos fins, as seguintes informações:

1. O autor declara que o trabalho apresentado na disciplina de Projeto de Graduação da Escola Politécnica da UFRJ é de sua autoria, sendo original em forma e conteúdo.

2. Excetuam-se do item 1. eventuais transcrições de texto, figuras, tabelas, conceitos e ideias, que identifiquem claramente a fonte original, explicitando as autorizações obtidas dos respectivos proprietários, quando necessárias.

3. O autor permite que a UFRJ, por um prazo indeterminado, efetue em qualquer mídia de divulgação, a publicação do trabalho acadêmico em sua totalidade, ou em parte. Essa autorização não envolve ônus de qualquer natureza à UFRJ, ou aos seus representantes.

4. O autor pode, excepcionalmente, encaminhar à Comissão de Projeto de Graduação, a não divulgação do material, por um prazo máximo de 01 (um) ano, improrrogável, a contar da data de defesa, desde que o pedido seja justificado, e solicitado antecipadamente, por escrito, à Congregação da Escola Politécnica.

5. O autor declara, ainda, ter a capacidade jurídica para a prática do presente ato, assim como ter conhecimento do teor da presente Declaração, estando ciente das sanções e punições legais, no que tange a cópia parcial, ou total, de obra intelectual, o que se configura como violação do direito autoral previsto no Código Penal Brasileiro no art.184 e art.299, bem como na Lei 9.610.

6. O autor é o único responsável pelo conteúdo apresentado nos trabalhos acadêmicos publicados, não cabendo à UFRJ, aos seus representantes, ou ao(s) orientador(es), qualquer responsabilização/ indenização nesse sentido.

7. Por ser verdade, firmo a presente declaração.

Leonardo Gomes de Castro e Silva

Leonardo Gomes de Castro e Silva

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
Escola Politécnica - Departamento de Eletrônica e de Computação
Centro de Tecnologia, bloco H, sala H-217, Cidade Universitária
Rio de Janeiro - RJ CEP 21949-900

À minha família.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Flávio Castro e Valdirene, aos meus irmãos, Flávio Gomes e Eduardo, e toda minha família por estarem sempre presentes, me incentivando e me apoiando durante toda minha jornada escolar e acadêmica. Muito obrigado por tudo, até mesmo os pequenos detalhes que conseguem fazer toda a diferença. Sem vocês, essa caminhada seria praticamente impossível. Espero retribuir todo suor, amor, sacrifícios e confiança depositados em mim nesses 24 anos.

Agradeço também a todos os amigos que fiz durante a faculdade, trabalho e da época de escola pela motivação e suporte durante toda minha graduação. A presença de vocês foi essencial durante meus 6 anos de faculdade, tornando minha vida no curso mais alegre.

Agradeço ao meu ex-orientador, professor Otto, por todos os ensinamentos, puxões de orelha, paciência e confiança fornecidos a mim. Sua influência e sorriso contagiantes mudaram completamente minha vida na faculdade, contribuindo significativamente para meu crescimento pessoal e profissional. Apesar de não estar mais entre nos, você está eternamente em minha memória, e desejo de coração que esteja sempre sorridente lá em cima como sempre será lembrado aqui na Terra. Espero retribuir tudo que me proporcionou, e honrar seu legado como professor e amigo. Sou eternamente grato, obrigado!

Agradeço ao meu orientador, professor Luís, por toda paciência e compreensão durante este árduo final de curso. Mesmo com meu foco disperso por conta do trabalho, você continuou me ajudando sempre que possível, continuou depositando sua confiança no meu projeto e me incentivando a continuar. Muito Obrigado!

Aos professores e a todos os meus companheiros do Grupo de Teleinformática e Automação (GTA), por proporcionarem um ambiente amigável e de muito aprendizado durante o período em que tive a honra de trabalhar e viver minha graduação juntos. Em especial ao Gugu, Rebello, Airam e Chagas.

Agradeço aos professores Rodrigo Couto e Marcelo Lanza por aceitarem o convite para participar da banca examinadora deste trabalho.

Agradeço à UFRJ e a todos os meus professores, por sempre estarem empenhados e contribuírem para minha formação.

Por fim, agradeço ao povo brasileiro por manter e garantir o meu acesso e de várias outras pessoas à educação pública de qualidade e excelência do Ensino Superior. Este trabalho é uma maneira de retribuir todo o investimento e confiança em mim depositados.

RESUMO

As redes móveis de quinta geração (5G) trazem inúmeros avanços e melhorias na qualidade de vida da população. Serviços de *streaming*, vídeos 3D, carros autônomos, fábricas e casas inteligentes são algumas das aplicações melhoradas ou possibilitadas pelo 5G. No entanto, construir um cenário 5G e atender a seus requisitos é algo complexo envolvendo a configuração de numerosos sistemas de hardware e software, onde erros humanos podem acarretar de perdas econômicas até riscos de vida, como uma perda de conexão durante uma cirurgia remota. Assim, este projeto propõe uma ferramenta robusta de automação fim-a-fim para criação e configuração de *multicluster* Kubernetes. A ferramenta proposta é compatível com ambientes multinuvem e geodistribuídos, um cenário complexo, dinâmico e heterogêneo típico de redes 5G. Um protótipo foi desenvolvido utilizando as ferramentas Terraform e Ansible para provisionar e configurar a infraestrutura do *multicluster* Kubernetes por meio de código e de forma automática, abstraindo toda a complexidade do ambiente e dos processos, assim evitando erros humanos. Os resultados de avaliação de desempenho mostram a diferença na eficiência do provisionamento dos recursos em diferentes provedoras de nuvem, e que a ferramenta é capaz de criar um ambiente com alta disponibilidade ao possuir *n clusters* Kubernetes em diferentes regiões, monitorado e de simples gestão parcialmente apto para aplicações 5G.

Palavras-Chave: Kubernetes, aplicações 5G, infraestrutura como código, automação, nuvem computacional.

ABSTRACT

The fifth-generation mobile networks (5G) bring numerous advances and improvements in the population's quality of life. Streaming services, 3D videos, autonomous cars, factories and smart homes are some of applications improved or made possible by 5G. However, building a 5G scenario and meeting its requirements is difficult, involving the configuration of numerous hardware and software systems, where human errors can lead to economic losses to life risks, such as a loss of connection during remote surgery. Thus, this project proposes a robust end-to-end automation tool for creating and configuring a Kubernetes multicluster. The proposed tool is compatible with multi-cloud and geo-distributed environments, typical 5g scenario with complexity, dynamicity and heterogeneity. A prototype was developed using Terraform and Ansible tools to provision and configure the Kubernetes multicluster infrastructure through code and automatically, abstracting all the complexity of the environment and processes, thus avoiding human errors. The performance evaluation results show the difference in the efficiency of resource provisioning in different cloud providers, and that the tool is able to create an environment with high availability when having n Kubernetes clusters in different regions, monitored and of simple management partially suitable for 5G applications.

Key-words: Kubernetes, 5G applications, infrastructure as code, automation, computational clouds.

SIGLAS

5G - Redes Móveis de Quinta Geração

AWS - *Amazon Web Services*

CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior

CC - *Cluster* de Controle

CI/CD - *Continuos Integration/Continuos Delivery*

CLI - *Command Line Interface*

CNPq - Conselho Nacional de Desenvolvimento Científico e Tecnológico

COPPE - Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

CT - *Cluster* de Trabalho

DB - *DataBase*

eMBB - *enhanced Mobile BroadBand*

FAPESP - *Fundação de Amparo à Pesquisa do Estado de São Paulo*

FAPERJ - *Fundação Carolos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro*

GCP - *Google Cloud Platform*

GPU - *Graphics Processing Unit*

GTA - Grupo de Teleinformática e Automação

HCL - *Hashicorp Configuration Language*

IaC - *Infrastructure as Code*

IP - *Internet Protocol*

IoT - *Internet of Things*

JS - *JavaScript*

K8s - Kubernetes

KubeFed - *Kubernetes Federation*

MEC - *Mult-access Edge Computing*

mMTC - *massive Machine Type Communications*

NC - Nó de Controle

NSM - *Network Service Mesh*

NT - Nó de Trabalho

P2P - *Peer-to-Peer*

REST - *Representational State Transfer*

RTT - *Round Trip Time*

SFC - *Service Function Chaining*

SMI - *System Management Interface*

TI - Tecnologia da Informação

UFRJ - Universidade Federal do Rio de Janeiro

URLLC - *Ultra-Reliable Low-Latency Communication*

YAML - *Yaml Ain't Markup Language*

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Tecnologias atuais de redes móveis de quinta geração (5G) e Internet das coisas (*Internet of Things* - IoT) possuem requisitos desafiadores em termos de latência, banda passante e disponibilidade [1, 2, 3, 4]. Uma das formas de atender tais demandas é posicionar as aplicações o mais perto possível dos usuários finais utilizando o conceito de computação de borda de multiacesso (*Multi-access Edge Computing* - MEC). Como o nome sugere, a MEC consiste em levar a computação para múltiplos pontos na borda da rede com o intuito de aproximar o usuário das aplicações, assim reduzindo a latência e possibilitando uma conexão consistente. Através das técnicas de virtualização e migração de serviços utilizadas na MEC, uma aplicação pode ser movida para pontos da borda próximos ao usuário, enquanto ele se move. Porém, o uso dessas tecnologias nas aplicações traz um alto custo de gerenciamento e dificuldade de implantação, pois os ambientes são heterogêneos em localidade, provedores de nuvem e/ou centro de dados. Além disso, a complexidade da gestão dessas aplicações cresce ainda mais quando utilizam a arquitetura de microsserviços [5]. Esta arquitetura de software consiste em construir aplicações desmembrando-as em serviços independentes, comumente implantados com uma virtualização leve baseada em contêineres. O uso de microsserviços facilita a manutenção da aplicação e permite o uso de diferentes linguagens para determinado serviço.

Um dos métodos para simplificar a implantação e gestão das aplicações 5G é por meio de automações em código. Nesse método, a configuração e a implantação dos aplicativos são feitas sem nenhuma ou com a menor interferência humana possível, assim evitando possíveis erros humanos. Esse método, no entanto,

não abrange a etapa inicial e igualmente importante, a criação e configuração da *infraestrutura* necessária para hospedar essas aplicações. A má configuração ou o desconhecimento da infraestrutura podem causar erros indesejados e afetar o funcionamento correto das aplicações. Logo, é necessário um meio de automatizar todo o processo desde a criação da infraestrutura até a implantação e gestão das aplicações para abstrair a complexidade dos ambientes, evitar a necessidade de interferência humana e atender os requisitos das aplicações 5G.

Um outro método para facilitar a gestão das aplicações 5G é por meio do Kubernetes[6], uma plataforma de código aberto para orquestração automática de contêineres e seus recursos. Ou seja, o Kubernetes gerencia o ciclo de vida das aplicações, o que envolve sua implantação, manutenção e, por fim, destruição. O uso do Kubernetes é essencial para aplicações conteinerizadas na arquitetura de microsserviços, visto que tais cenários envolvem milhares de pequenos serviços, tornando praticamente inviável uma administração manual de cada um dos serviços. Entretanto, o Kubernetes não atende com eficiência ambientes geodistribuídos e multinuvem característicos de aplicações 5G. Portanto, é necessário um meio de utilizar os benefícios da plataforma do Kubernetes sem prejudicar a eficiência e atendendo aos requisitos das aplicações.

Este projeto propõe uma ferramenta que automatiza fim-a-fim a criação e configuração da infraestrutura, e a implantação de aplicações geodistribuídas em uma arquitetura de microsserviços com múltiplos *clusters* Kubernetes[1] (também chamado de *Multicluster* Kubernetes). Diferente de outras propostas que automatizam o processo a partir da configuração do *cluster* Kubernetes ou requerem um passo manual inicial, a ferramenta proposta neste projeto faz a automação completa sem passos manuais, desde a criação da infraestrutura até a implantação da aplicação. A ferramenta de automação utiliza o conceito de infraestrutura como código (*Infrastructure as Code* - IaC) com o Terraform [7] e o Ansible [8]. O Terraform é uma ferramenta de provisionamento de infraestrutura a partir de código, utilizada neste projeto para provisionar toda a infraestrutura dos *clusters* e abstrair a complexidade dos ambientes heterogêneos. O Ansible [8] é uma ferramenta de gestão, automação e configuração de infraestrutura a partir de código, utilizada neste projeto para

[1]Disponível em `https://kubernetes.io`.

configurar os *clusters* Kubernetes e implantar as aplicações 5G. O uso conjunto dessas duas ferramentas permite uma automação completa da criação, configuração e implantação de aplicativos sem a necessidade de interferência humana. O projeto também utiliza uma plataforma de controle logicamente centralizada para gerenciar os vários *clusters* Kubernetes de forma simples, coordenada e sem afetar a disponibilidade das aplicações. A plataforma de controle é composta de componentes do Linkerd [9], ferramenta para criação de *multiclusters* Kubernetes, e fluxos de integração e entrega continua (Continuos Integration/Continuos Delivery - CI/CD) desenvolvidos no GitHub Actions [10], ferramenta de automação de fluxos de tarefas.

O projeto implementa 6 módulos para compor a ferramenta de automação utilizando ferramentas de código aberto, sendo eles: (i) módulo Terraform para criação de *cluster* Kubernetes na nuvem da Azure[2]; (ii) módulo Terraform para criação de *cluster* Kubernetes na nuvem da AWS[3] (*Amazon Web Services*); (iii) módulo Terraform para criação de cofres de chaves para armazenamento de credenciais sensíveis do *clusters*; (iv) módulo Terraform de inventário dos *clusters* e suas credenciais para conexão com a ferramenta Ansible; (v) módulo Ansible para configuração do *multicluster* Kubernetes; e (vi) módulo Ansible para implantação das aplicações 5G em todos os *clusters* Kubernetes. O projeto, então, avalia o desempenho da ferramenta de automação com os módulos e das aplicações de teste para certificar que a estrutura atende aos requisitos das aplicações 5G. Um protótipo foi desenvolvido e avaliado em duas provedoras de nuvem quanto ao provisionamento e configuração da infraestrutura. Resultados de avaliação de desempenho do protótipo desenvolvido mostram qual das provedoras de nuvem melhor atende o *multicluster* Kubernetes, e que a ferramenta proposta atende parcialmente aos requisitos de uma aplicação 5G, cumprindo com o requisito de vazão de *download* e ficando bem próximo do requisito de latência.

1.1 Delimitação

O objeto de estudo deste trabalho é um cenário com múltiplos *clusters* Kubernetes para aplicações 5G em diferentes localidades, utilizando um controle cen-

[2]Disponível em `https://azure.microsoft.com`.

[3]Disponível em `https://aws.amazon.com`.

tralizado dos *clusters* para facilitar sua gestão pelo administrador sem prejudicar sua disponibilidade. O ideal é que o administrador consiga verificar a saúde, carga, comunicação e gerenciar as aplicações dos *clusters* Kubernetes em um ponto único, abstraindo a complexidade da gestão de múltiplos *clusters* separadamente. O cenário avaliado considera a utilização de *clusters* Kubernetes fornecidos por grandes provedores de nuvem conectados de forma segura pela Internet. O intuito é demonstrar que a solução suporta diversos cenários, tais como: (1) todo o *multicluster* Kubernetes instanciado em uma única provedora de nuvem e (2) o *multicluster* Kubernetes instanciado em diferentes provedores de nuvem simultaneamente. Em cada um dos cenários, um módulo de automação é desenvolvido para compor a automação fim-a-fim proposta neste projeto.

1.2 Objetivos

O objetivo geral deste projeto é desenvolver uma ferramenta que garanta a criação, configuração e implantação de aplicações em múltiplos *clusters* Kubernetes de forma automática e como código sem nenhuma intervenção humana, desde a infraestrutura até a aplicação. Dessa forma, tem-se como objetivos específicos:

1. desenvolver módulos com Terraform e Ansible para criar e configurar os *clusters* Kubernetes;
2. automatizar o processo de implantação de aplicações para retirar a necessidade de o usuário e/ou administrador ter acesso direto e frequente nos *clusters*;
3. atender os requisitos de latência, disponibilidade e vazão das aplicações 5G;
4. disponibilizar uma plataforma de controle centralizada para facilitar a gestão dos múltiplos *clusters*.

1.3 Metodologia

Este projeto possui três etapas principais: (1) o estudo do estado da arte das arquiteturas *multicluster* Kubernetes, soluções de automação e sua afinidade com 5G; (2) a proposta de uma abordagem completamente automática e como

código para criação, configuração e implantação de aplicações nos *clusters* e (3) a implementação e avaliação da plataforma de controle centralizado, e dos módulos Terraform e Ansible para criação dos *clusters* em diferentes localidades e provedores de nuvem.

A primeira etapa busca verificar as tecnologias existentes de provisionamento de infraestrutura, configuração de componentes e aplicações de forma automática e como código. Além disso, a primeira etapa prevê a pesquisa de artigos científicos que utilizem a arquitetura de *multicluster* Kubernetes em cenários com diferentes provedores de nuvem para aplicações 5G, com ou sem automações, e uma discussão das propostas encontradas.

A segunda etapa visa propor uma ferramenta de automação fim-a-fim em código para criação de *clusters* Kubernetes, desde o provisionamento da infraestrutura necessária até a sua configuração e implantação das aplicações a partir do estado da arte realizado na etapa anterior. Esta etapa detalha as tecnologias, estratégias e arquitetura da automação fim-a-fim a ser proposta.

A última etapa do projeto implementa e avalia a plataforma de controle centralizado e a ferramenta de automação fim-a-fim propostas na etapa anterior. Esta etapa prevê a utilização das ferramentas de código aberto Terraform [7] para o provisionamento da infraestrutura como código e o Ansible [8] para a configuração dos *clusters* Kubernetes e implantação das aplicações. O Terraform permite criar e gerenciar infraestrutura a partir de código usando uma linguagem declarativa, enquanto o Ansible permite gerenciar, automatizar, configurar e implantar aplicativos a partir de um ponto central, também utilizando linguagem declarativa. Essas ferramentas foram escolhidas porque permitem gerenciar a infraestrutura em código simples e de fácil leitura devido à natureza declarativa de suas linguagens. Além disso, ambas são idempotentes, garantindo sempre o mesmo resultado em qualquer execução. Por fim, a etapa inclui uma avaliação de desempenho para verificar o quão rápida é a automação da criação, configuração e implantação de aplicações nos *clusteres* Kubernetes, validar sua disponibilidade e velocidade de resposta de algumas aplicações de teste para certificar que a estrutura é adequada para o 5G.

1.4 Organização do Texto

O restante deste trabalho está organizado em quatro capítulos. O Capítulo 2 apresenta a fundamentação teórica da tecnologia de Kubernetes, os métodos de automação de infraestrutura de TI e as aplicações 5G, detalhando como funcionam, suas arquiteturas com seus pontos positivos e negativos, e suas usabilidades. O Capítulo 3 apresenta a arquitetura proposta para a ferramenta de automação fim-a-fim de criação e configuração do *multicluster* Kubernetes, detalhando cada um dos seus módulos, as ferramentas utilizadas e desenvolvidas, e os resultados do protótipo desenvolvido com avaliações para validar a viabilidade da arquitetura para aplicações 5G. O Capítulo 4 apresenta os trabalhos relacionados com este projeto e o estado da arte do *multicluster* Kubernetes. Por fim, o Capítulo 5 conclui o trabalho discutindo e apresentando a direção para trabalhos futuros.

Capítulo 2

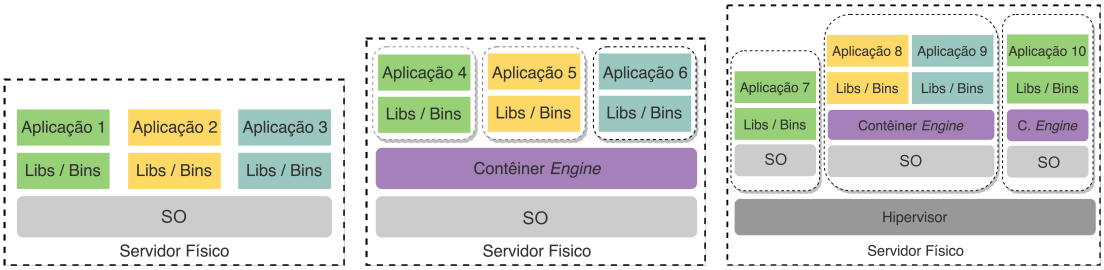
Fundamentação Teórica

Este capítulo apresenta os principais conceitos da tecnologia de Kubernetes, descreve a automação de infraestrutura de Tecnologia da Informação (TI) e as aplicações 5G, destacando suas definições, requisitos e ferramentas envolvidas no desenvolvimento do trabalho.

2.1 A Tecnologia de Kubernetes

O uso da virtualização baseada em contêiner traz desafios de controle, manutenção e monitoramento das aplicações que utilizam essa tecnologia. Conforme ilustrado na Figura 2.1, o contêiner é uma alternativa mais leve à máquina virtual (*Virtual Machine* - VM) baseada em hipervisor, camada de *software* que abstrai e controla o *hardware* para uma VM [11]. Diferentemente de uma VM que abstrai uma máquina completa, desde o *hardware* até o sistema operacional que executa na máquina física (hospedeiro, ou *host*), a virtualização por contêiner funciona no nível do próprio sistema operacional (SO) do hospedeiro, fornecendo as abstrações diretamente aos processos/aplicações através de chamadas de sistema (*system calls*) direcionadas a um mesmo *kernel* (núcleo) de sistema operacional, compartilhado entre os contêineres. Apesar dessa abordagem de compartilhamento do *kernel* prover um menor isolamento se comparado a uma VM tradicional, esse método de virtualização fica bem mais leve ao alocar apenas os recursos necessários para aquele processo/aplicação, como memória, CPU, bibliotecas e/ou funções, e não precisar abstrair uma máquina completa. Essa característica de leveza dos contêineres permite desenvolver novas aplicações de forma mais rápida e em escala [12], além de

permitir que a aplicação seja facilmente replicada em qualquer lugar pela funcionalidade de empacotamento do contêiner.



(a) (b) (c)

Figura 2.1: Servidor físico (a). Virtualização baseada em contêiner implementada no Sistema Operacional (SO) de um servidor físico (b). Virtualização baseada em hipervisor com máquinas virtuais que hospedam aplicações e outras que hospedam contêineres (c).

O Kubernetes (K8s) surgiu da necessidade de atender esse crescimento de escala e complexidade na gestão de múltiplas aplicações conteinerizadas. Criada pela Google, o Kubernetes é uma plataforma de código aberto para gerenciar recursos e ciclo de vida de aplicações em contêineres [6]. O ciclo de vida de uma aplicação envolve sua implantação, manutenção e, por fim, destruição. A gestão desse ciclo e dos recursos envolvidos é chamada de orquestração na literatura, o que explica algumas fontes se referirem ao Kubernetes como um orquestrador de contêineres. Vale ressaltar que o Kubernetes funciona em cima de um virtualizador de contêiner, cujo mais famoso é o Docker [13], mesmo ele não sendo mais usado nas versões atuais do Kubernetes que usam o Containerd [14] ou CRI-O (*Container Runtime Interface plus Open container initiative*) [15]. Além da orquestração, o Kubernetes possui as seguintes funcionalidades: (i) descoberta de serviços para encontrar automaticamente onde a aplicação está hospedada; (ii) balanceamento de carga; (iii) escalonamento automático conforme a carga atual da aplicação; (iv) autocura em caso de falhas em contêineres ou seus hospedeiros e (v) gerenciamento de *tokens* e senhas.

Um *cluster* Kubernetes é um conjunto de nós, máquinas, que executam os serviços do Kubernetes para orquestrar e executar os contêineres. A arquitetura padrão recomendada para um ambiente de produção do Kubernetes consiste de um plano de controle, responsável por gerenciar o *cluster*, e um plano de trabalho, que

executa as aplicações e concentra a maior parte do processamento do *cluster*, conforme representado na Figura 2.2. O Kubernetes é composto por nós de controle (NC), pertencentes ao plano de controle e que executam funções de gerenciamento e orquestração do *cluster* e das aplicações, e nós de trabalho (NT), pertencentes ao plano de trabalho e que executam praticamento todo o processamento do *cluster*, instanciando as aplicações em *pods*. O termo *pod*[1], no contexto de *clusters* Kubernetes, representa um processo em execução no *cluster*, sendo ele um conjunto de um ou mais contêineres agrupados em um mesmo pacote para maximizar os benefícios do compartilhamento de recursos, como armazenamento e rede, e contexto. Como um exemplo, um *pod* representa uma aplicação de um site de vendas. Dentro desse *pod*, tem um contêiner representando a interface do site (*frontend*), outro representando o inventário dos produtos, os serviços de entrega disponíveis e outro representando a área de usuário. Para que uma venda ocorra, todos os contêineres precisam de algum nível de interação entre si, e por isso são implantados no mesmo *pod*. A documentação do Kubernetes recomenda o uso de um número ímpar maior ou igual a 3 de NCs, pois isso evita a ocorrência de empates no protocolo de consenso distribuído baseado em eleição, Raft [16, 17], utilizado para definir o NC ativo na configuração com alta disponibilidade. A sugestão do uso de um número impar de NCs ocorre para evitar a ocorrência de empates durante o funcionamento normal do *cluster* na etapa de eleição do NC líder, sendo o quorum da eleição definido pela maioria simples. Como exemplo, em um cenário com 3 NCs, o quorum é 2, logo, o *cluster* tolera a falha de até 1 NC. No caso de um ambiente com 4 NCs, o quorum aumenta para 3, ou seja, continua tolerando a falha de apenas 1 nó. Por esse motivo é feita a recomendação do uso de um número impar de NCs no *cluster* Kubernetes. Vale ressaltar que o NC e NT também são denominados, respectivamente, de mestre e escravo em algumas documentações e/ou literaturas antigas, porém essa nomenclatura entrou em desuso devido a questões éticas.

O funcionamento do plano de controle é coordenado pelo NC ativo, responsável por atender as requisições dos componentes do *cluster* e o gerenciar, enquanto os outros NCs estão em uma configuração de redundância para caso o NC

[1]O termo *pod* do Kubernetes vem da expressão baleal, coletivo de baleias, que em inglês é *pod of whales*. Como o simbolo do Docker, principal virtualizador de contêineres utilizado no Kubernetes, é uma baleia, a definição de *pod* faz uma analogia de grupo de baleias com grupo de contêineres.

ativo tenha alguma falha física, no seu SO e/ou perda de comunicação com os temporizadores de verificação de saúde (*health check*) que existem nos NC, assim começando uma eleição de um novo líder. O plano de controle possui 4 componentes principais: (i) o controlador, elemento que monitora a saúde e estado dos NCs e NTs do *cluster*, e das aplicações, controla as conexões entre serviços, objetos, contêineres e gerencia os *tokens* e credenciais do *cluster*; (ii) o agendador, que monitora os *pods*, e seleciona o NT em que ele será executado; (iii) o etcd[2], armazenamento do tipo chave-valor distribuído que guarda todos os dados do *cluster*; e (iv) a API Kubernetes, que é o ponto central de comunicação do plano de controle com seus componentes e com o plano de trabalho.

O plano de trabalho é composto pelos NTs e possui 2 componentes principais, o kubelet, um agente que adiciona o NT no *cluster* por meio da Kubernetes API e garante a execução e saúde dos contêineres, e o serviço de *proxy*, que garante a comunicação dos *pods* com elementos dentro ou fora do *cluster*. Além disso, o NT possui um virtualizador de contêiner para instanciar os *pods* das aplicações. Apesar do NC também possuir um virtualizador de contêiner, ele é usado para instanciar componentes de controle, monitoramento, rede do *cluster*. É importante pontuar que existem diversas arquiteturas de Kubernetes, como um modelo sem alta disponibilidade com apenas um NC e alguns NT, ou até mesmo um cenário com apenas um nó que executa simultaneamente a função de NC e NT, porém nenhuma dessas arquiteturas é recomendada para ambientes de produção, pois elas possuem pouca, ou nenhuma, tolerância a falhas.

Outras arquiteturas possíveis para o Kubernetes são as de *multicluster*. O *multicluster* Kubernetes é um ambiente com *n clusters* Kubernetes com comunicação entre si para coordenar o gerenciamento do ciclo de vida das aplicações de forma distribuída. O *multicluster* é uma abordagem recente com o diferencial de aumentar a tolerância a falhas e isolamento, aproximar as aplicações dos usuários finais em múltiplas localidades reduzindo a latência na comunicação, aumentar a disponibilidade e disponibilizar um alto poder computacional dos *clusters* nas bordas da rede, características desejáveis para algumas aplicações 5G, em especial as aplicações que

[2]A sigla etcd é derivada de uma convenção de nomes da estrutura de diretórios do Linux. Em sistemas UNIX, todos os arquivos de configuração do sistema ficam no diretório **/etc**. O "d"de etcd significa distribuído.

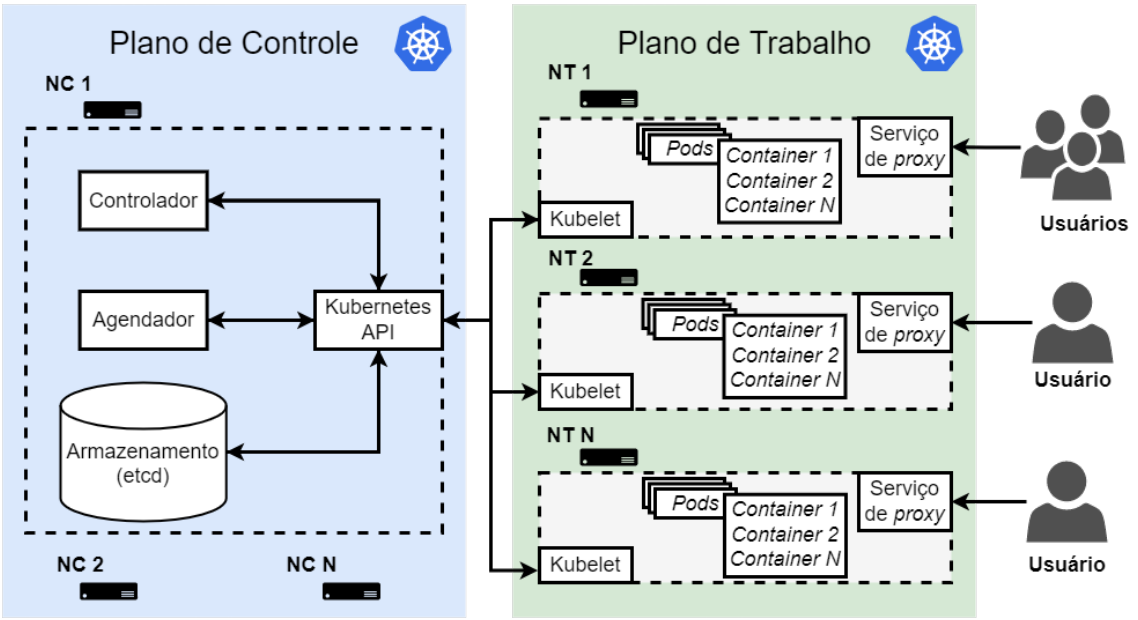


Figura 2.2: Arquitetura modelo para ambientes produtivos de Kubnernetes. O *cluster* é composto de no mínimo 3 nós de controle (NC) pertencentes ao plano de controle que gerencia todo o *cluster*, orquestra as aplicações e armazena seus dados, e 3 nós de trabalho (NT) que executam os *pods*, abstração de componentes de uma aplicação que representa um ou mais contêineres. As regiões pontilhadas delimitam componentes internos aos nós.

exigem ultra baixa latência. As propostas de arquitetura para o *multicluster* são baseadas principalmente em duas metodologias, a centrada no controlador (*controller-centric*) e a centrada na rede (*network-centric*) [18, 19]. A arquitetura **centrada no controlador**, representada na Figura 2.3, consiste de um *Cluster* de Controle (CC) responsável por configurar, orquestrar, controlar e monitorar todo o ambiente, e *n Clusters* de Trabalho (CT) que executam as aplicações e cargas de trabalho. A comunicação do CC com o CT ocorre através da Kubernetes API do CC, ou Kubernetes API global, com a Kubernetes API do CT. Essa comunicação corresponde ao monitoramento, temporizador de saúde dos *clusters* e comandos de controle para o CT. Esse tipo de arquitetura também é chamado de Kubernetes federado na literatura, oriundo do nome da ferramenta utilizada para construção do *multicluster*, o Kubernetes *Federation* (KubeFed) [20]. A ferramenta KubeFed coordena as configurações de múltiplos *clusters* Kubernetes, gerencia aplicações geodistribuídas e fornece mecanismos para recuperação de desastres de um ou mais *clusters* a partir de um conjunto de APIs do CC. Um ponto negativo dessa arquitetura é que o CC

torna-se um ponto único de falha, em caso de falhas a nível do CC como um todo ou da região em que os NCs são hospedados. Um ponto discutível sobre essa afirmação é que um CC pode ter seus NCs espalhados em diferentes regiões, porém essa configuração nem sempre é ideal. NCs instanciados em diferentes localidades acarretam maior latência na comunicação entre os nós, o que gera a necessidade de aumentar o temporizador de verificação de saúde dos NCs e o temporizador de comunicação com os NTs do *cluster*. Por consequência desse aumento nos temporizadores, o tempo de detecção de uma falha também aumenta, o que pode ser algo indesejável em um ambiente de produção. Assim, recomenda-se que um *cluster* tenha todos os seus NCs e NTs instanciados na mesma região para garantir a menor latência possível na comunicação entre os seus nós.

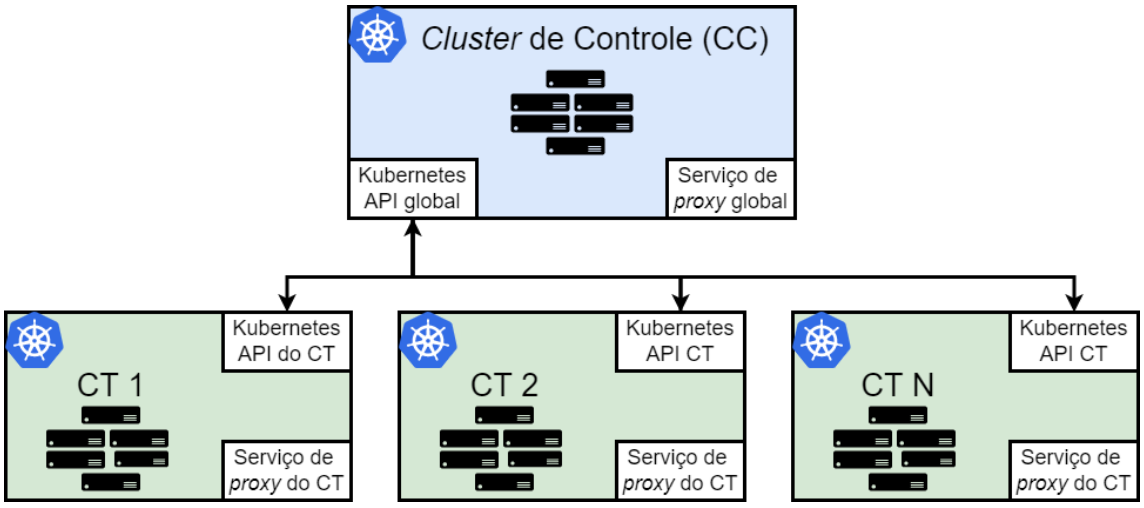


Figura 2.3: Arquitetura de *multicluster* Kubernetes centrada no controlador (*controller-centric*). O *Cluster* de Controle (CC) configura, orquestra, monitora e controla todos os *Cluster* de Trabalho (CT) de diferentes tamanhos e/ou localidades.

Já a arquitetura **centrada na rede**, representada na Figura 2.4, consiste de um componente logicamente centralizado que atua na interconexão dos *clusters* pela rede. Nessa configuração, todos os CTs são entidades independentes que podem possuir comunicações contêiner-contêiner através de uma solução de rede *intercluster*, denominada de malha de rede ou serviço (*network/service mesh*). A malha de rede define quais contêineres podem ou não abrir conexões diretas com outros contêineres, como uma espécie de *firewall*. Essa malha pode ser um componente instanciado em um CC ou em n CTs para garantir um ambiente tolerante a falhas e redundante.

A diferença da comunicação dos *cluster* Kubernetes desta arquitetura para a centrada no controlador é que todos os *clusters* podem estabelecer conexões entre si, uma comunicação n:n que lembra a imagem da malha de rede/serviço, enquanto a comunicação dos *clusters* na arquitetura centrada no controlador é centralizada no CC, uma comunicação 1:n do CC para todos os CTs ou n:1 de todos os CTs para o CC. Alguns exemplos de malhas de rede são:

- **Linkerd** [9] é uma ferramenta de código aberto para criação de malhas de serviços no Kubernetes. Os diferenciais do Linkerd são sua leveza, facilidade de implantação e a entrega de interfaces de visualização e ferramentas de monitoração integradas à solução. A leveza do Linkerd é adquirida por utilizar um plano de controle segregado de um plano de dados. O plano de controle possui três componentes: (i) um serviço de identificação de destinos utilizado pelos *proxies* do plano de dados para verificar se a comunicação é permitida, adquirir informações de rotas, entre outros; (ii) um serviço de identidade que atua como uma entidade certificadora para validar as comunicações *proxy-proxy* e garantir sua segurança; e (iii) um injetor de *proxy* que insere um contêiner de *proxy* nos *pods* indicados com uma etiqueta (*label*) específica para extrair métricas para o *multicluster* Kubernetes e gerenciar as conexões externas a esses *pods*. Já o plano de dados é composto por micro *proxies* implantados como contêineres de *sidecar*[3] dentro dos *pods* das aplicações. A funcionalidade do contêiner de *sidecar* é atuar de forma isolada no *pod* da aplicação para adicionar alguma funcionalidade ou realizar seu monitoramento. No caso do micro *proxy*, ele intercepta, de forma transparente, as conexões TCP internas e externas ao *pod* para exportar métricas para a ferramenta de monitoramento, atuar como um *proxy* de rede, atuar como balanceador de carga de camadas 4 e 7 e aplicar criptografia com protocolo TLS. Devido à leveza da ferramenta e serviços de monitoramento integrados, o Linkerd é utilizado neste projeto para configuração do *multicluster* Kubernetes e seu monitoramento.

- **Consul** [21] é um sistema complexo, dinâmico e de código aberto de malhas e

[3]Um *sidecar* é um dispositivo acoplado a um dispositivo principal para estender ou adicionar funcionalidade, como o utilizado em motocicletas.

descoberta de serviços para Kubernetes. Sua arquitetura é baseada em cliente-servidor, onde os clientes são os NTs dos CTs e os servidores, os NCs dos CCs. Essa configuração de redundância garante uma alta disponibilidade para o *multicluster* Kubernetes. O controle e distribuição de mensagens para os *clusters* Kubernetes são feitos utilizando o protocolo de propagação de boatos (*gossip*). O protocolo é utilizado tanto para comunicação cliente-servidor, quanto entre clientes ou entre servidores em *clusters* Kubernetes distintos. Assim como o Linkerd, o Consul utiliza *proxies* de *sidecar* para monitorar a aplicação e permitir a comunicação entre *pods* de qualquer *cluster* Kubernetes.

A desvantagem dessa arquitetura é a dificuldade de gestão do *multicluster*, uma vez que todos os *clusters* são independentes entre si e não existe um componente de monitoramento e controle centralizado dos outros recursos além da rede, como os nós dos *clusters*, memória RAM, CPU, aplicações, entre outros. Apesar disso, existem algumas ferramentas que amenizam esse ponto negativo ao implantar os recursos de controle e monitoramento diretamente nos *clusters*.

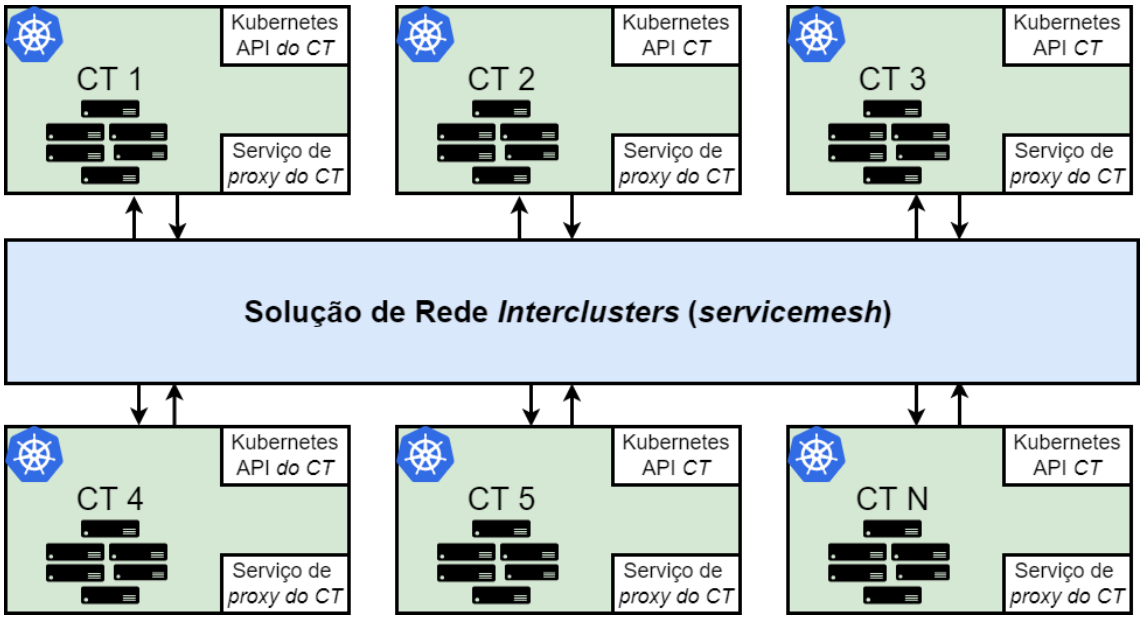


Figura 2.4: Arquitetura de *multicluster* Kubernetes centrada na rede (*network centric*). A solução de rede *interclusters*, malha de rede ou serviço (network/service mesh), permite ou não a comunicação contêiner-contêiner de *clusters* independentes entre si, similar a um *firewall*. A malha de rede pode ser um componente instanciado em um *Cluster* de Controle (CC) ou em N *Clusters* de Trabalho (CT).

Este trabalho irá utilizar noções das arquiteturas centrada na rede e no controlador simultaneamente, propondo uma arquitetura híbrida. O intuito da arquitetura híbrida é centralizar, de forma lógica com redundância, os componentes de controle, monitoramento e visualização do *multicluster* Kubernetes para facilitar sua administração, e distribuir os componentes de rede e processamento para garantir uma maior disponibilidade das aplicações. O Linkerd é utilizado para criar a malha e descoberta de serviços, além de proporcionar ferramentas de monitoramento do *multicluster* Kubernetes. A parte de controle do *multicluster* Kubernetes é feita através de fluxos de tarefas no GitHub Actions, uma solução de integração e entrega contínua (*Continuos Integrations/Continuos Delivery* - CI/CD). Os fluxos de tarefas executam o Terraform [7] e o Ansible [8], apresentados na Seção 2.2 a seguir, para escalar e/ou modificar o *multicluster* Kubernetes provisionando e alterando recursos de infraestrutura, e aplicar e/ou modificar configuração do *multiclusters* Kubernetes e das aplicações 5G.

2.2 Automação da Infraestrutura de TI

No escopo deste projeto e no âmbito da programação, o termo automação refere-se a códigos, *softwares* e/ou sistemas que substituem processos repetitivos e manuais. Ao substituir tais processos, a automação acelera a esteira de entregas de um negócio, permitindo que ele escale com mais facilidade. Além disso, garante a execução padronizada de tarefas e evita possíveis erros humanos gerados nas intervenções manuais, consequentemente reduzindo o custo operacional e de mão de obra. Mais especificamente, este projeto utiliza a automação no nível de infraestrutura e na implantação de aplicações.

A automação de infraestrutura é associada ao conceito de infraestrutura como código (*Infrastructure as Code* - IaC). Nesse paradigma, os recursos são descritos em forma de códigos reutilizáveis [22], assim diminuindo a necessidade de tarefas manuais e demoradas, tipicamente realizadas por especialistas em infraestrutura de TI, e substituindo-as por códigos e automações de ferramentas de IaC. A partir do momento que a infraestrutura está definida em código, o provisionamento, atualização e execução dos recursos, ou também chamado de gestão do ciclo de vida da infraestrutura, são abstraídos e controlados por ferramentas de IaC.

As ferramentas de IaC são definidas e agrupadas em 2 tipos, de acordo com sua abordagem de definição do estado da infraestrutura: o tipo declarativo ou funcional, e o tipo imperativo ou procedural [23]. No tipo **declarativo**, o código define o estado final desejado da infraestrutura, suas propriedades e seus requisitos, não importando as etapas e processos para chegar a esse estado. Dadas essas características, o tipo declarativo é reconhecido pelo seu código simples e de fácil leitura, uma vez que as complexas etapas de criação da infraestrutura são abstraídas pela ferramenta. Já no tipo **imperativo**, o código define todos os passos ou comandos para se alcançar o estado final desejado. Apesar disso geralmente tornar o código mais complexo, o tipo imperativo permite uma maior flexibilidade, pois as etapas podem ser definidas de n formas diferentes para se chegar no mesmo estado final. A escolhas e definição dessas etapas são uma decisão de projeto que pode ter seu foco, por exemplo, na velocidade de provisionamento da infraestrutura, na integridade da infraestrutura, na redução da taxa de erros, entre outros. Seguem alguns exemplos das ferramentas de IaC mais utilizadas e seus tipos, desconsiderando aquelas de uso restrito a infraestrutura de uma provedora de nuvem:

- **Terraform** [7] é uma ferramenta de código aberto para automação de infraestrutura com suporte a diversos provedores de nuvem, e também a plataformas famosas como o repositório de códigos GitHub e o orquestrador de contêineres Kubernetes. A ferramente é do tipo declarativo e utiliza o HCL (*Hashicorp Configuration Language*), uma linguagem de alto nível para IaC. Por meio dos códigos em HCL, o Terraform identifica o estado final desejado para a infraestrutura, verifica o estado atual para identificar as alterações necessárias e montar um plano de ação, e depois executa esse plano aplicando todas as mudanças previstas para alcançar o estado final. No caso do Terraform, o estado da infraestrutura é armazenado em um arquivo que é sempre comparado com o estado atual e o declarado em código para identificar as mudanças necessárias, e também validar possíveis alterações da infraestrutura fora do escopo do código. Além disso, a própria ferramenta identifica todas as dependências entre os recursos e paralisa o provisionamento ou configuração do que for possível, abstraindo a complexidade da programação paralela e reduzindo erros humanos. Entretanto, sua principal característica é a idempotência, ou

seja, um código HCL sempre produz o mesmo resultado independentemente de quantas vezes ele seja executado. Vale ressaltar que o Terraform é uma das ferramentes utilizadas neste projeto, pois ela suporta ambientes multinuvem, consegue monitorar o estado atual do ambiente possibilitando a criação de mecanismos de autocura de infraestrutura, possui integrações e módulos para diversas plataformas, denominados de provedores (*providers*) pela ferramenta, além de ser de fácil uso devido a sua característica declarativa, o que facilita e acelera o desenvolvimento. Também são os provedores do Terraform que se encarregam da comunicação com os provedores de nuvem. Essa comunicação com as nuvens é feita via API e é abstraída nos provedores do Terraform, sendo transparente para o desenvolvedor que precisa apenas fornecer uma credencial de acesso à nuvem. Alguns exemplos de códigos HCL utilizados neste trabalho estão no Apêndice A.

- **Ansible** [8] provê uma forma simples de configurar e gerenciar aplicações e infraestrutura em escala. Sendo uma ferramenta do tipo declarativa, a execução do Ansible utiliza arquivos de configuração escritos em YAML (*Yaml Ain't Markup Language*), uma linguagem de serialização de dados, amplamente utilizada por aplicações e de fácil leitura. O principal arquivo de configuração do Ansible é denominado *playbook*, arquivo em YAML com a declaração do estado desejado para cada recurso. Ao ler os *playbooks*, o Ansible verifica sequencialmente, em tempo de execução, cada recurso com o estado declarado para identificar se é preciso ou não alterar a infraestrutura. O Ansible é similar ao Terraform no quesito de verificação e alteração do estado somente quando necessário, porém a diferença é que o Ansible não armazena o estado da infraestrutura em um arquivo, ele sempre verifica o estado atual em tempo de execução e o compara ao estado desejado declarado no código. Além disso, o Ansible é utilizado para configuração de infraestrutura, enquanto o Terraform para provisionamento de infraestrutura. Vale ressaltar que o Ansible é uma ferramenta de código aberto utilizada neste projeto, pois a ferramenta funciona muito bem em escala, o que possibilita a automação e gestão da configuração dos n nós dos N *clusters* Kubernetes e suas aplicações. Alguns exemplos de *playbooks* do Ansible utilizados neste trabalho estão no Apêndice A.

- **Chef** [24] é uma ferramenta de código aberto para implantação e modelagem escalável e segura de processos de automação de infraestrutura, compatível com diferentes ambientes. O Chef é do tipo imperativo, ou seja, o código define todas as etapas para se alcançar o estado desejado. Para tal, o Chef utiliza as receitas (*recipes*) e os livros de receitas (*cookbooks*) para codificar detalhadamente cada etapa necessária para atingir o estado desejado da infraestrutura e suas aplicações. Essas receitas e livros de receitas são escritos em Ruby, uma linguagem de programação multiparadigma, de tipagem forte e dinâmica.

- **Puppet** [25] é uma ferramenta de código aberto para gerenciar e automatizar a configuração de servidores. O seu funcionamento se dá através da plataforma Puppet (*Puppet platform*), um conjunto de pacotes para automatizar a infraestrutura e gerenciar seu estado através de uma arquitetura servidor-agente. O servidor Puppet (*Puppet server*) armazena os códigos com a definição do estado desejado da infraestrutura e controla os agentes. Por sua vez, os agentes são os servidores em que se deseja automatizar e gerenciar suas configurações. Inicialmente, os agentes executam uma ferramenta de inventário para coletar algumas informações dos servidores, como IP, sistema operacional e nome do servidor (*hostname*). Em seguida, os agentes reportam esse inventário ao servidor Puppet que cria vários catálogos descrevendo o estado desejado para cada agente e os armazena em um banco de dados (*Puppet DataBase* - PuppetDB). Por fim, cada agente recebe seu catálogo e aplica em si todas as mudanças necessárias para alcanças seu estado desejado. Além disso, os agentes periodicamente reportam seu estado para o servidor Puppet com intuito de identificar qualquer alteração indevida e retornar ao estado desejado descrito nos códigos. Os códigos são escritos em uma linguagem declarativa, também chamada Puppet, desenvolvida pelos próprios criadores da ferramenta Puppet.

2.3 Aplicações 5G

O uso das redes móveis de quinta geração (5G) possibilita melhorar ainda mais a qualidade dos serviços ao aumentar taxa de transferência de dados (*through-*

put) e reduzir a latência [26]. Entretanto, prover conexões com esse nível de qualidade é um desafio para os provedores de Internet e de serviços. Uma solução muito usada para tal desafio é levar a computação para múltiplos pontos próximos dos usuários finais, conceito conhecido como computação de borda de multiacesso (*Multi-access Edge Computing - MEC*). Ter a computação perto do usuário significa uma menor distância para que os dados trafeguem da origem ao destino, ou do destino até a origem, assim reduzindo a latência que é medida utilizando o RTT (*Round Time Trip*), tempo total que um pacote de rede leva para ir de uma origem até um destino e depois voltar à origem.

O 5G é caracterizado por três casos de uso principal [1], sendo eles: (i) banda larga móvel melhorada (*enhanced Mobile BroadBand* - eMBB) que consiste na entrega de serviços de banda móvel com conexões mais rápidas, maiores taxas de transferência de dados e maiores capacidades, tipicamente útil para serviços de *streaming* de vídeo, vídeos em 3D, realidade virtual; (ii) comunicação ultraconfiável de baixa latência (*Ultra-Reliable Low-Latency Communications* - URLLC), usada em aplicações críticas que requerem conexões robustas e sem interrupções, onde falhas de milissegundos podem significar perdas irreversíveis, como cirurgias remotas, monitoramento dos sinais vitais de pacientes, carros autônomos; e (iii) comunicações maciças de tipo de máquina a máquina (*massive Machine Type Communications* - mMTC) que consiste na conexão de uma quantidade elevada de dispositivos com baixos poder computacional e custo, e distribuídos em uma grande área, tipicamente sensores, câmeras e medidores de aplicações de IoT, fábricas e casas inteligentes [27]. A Tabela 2.1 apresenta resumidamente os requisitos do 5G por cada caso de uso descrito anteriormente [1].

Para o escopo deste projeto, aplicações 5G são aplicações que atendem ao requisito de latência e taxa de transferência de dados do 5G. Vale ressaltar que, por motivos de custo, não está no escopo deste projeto simular uma rede 5G, e sim propor e avaliar uma arquitetura baseada em MEC, no caso o *multicluster* Kubernetes, que seja compatível com o 5G e consiga instanciar aplicações 5G.

Este capítulo abordou os principais fundamentos de contêineres, Kubernetes e suas diferentes arquiteturas, automações de infraestrutura de TI e aplicações 5G, discutindo suas definições, benefícios e possíveis pontos negativos. A seguir, este projeto apresenta a arquitetura da ferramenta de automação fim-a-fim de provisiona-

Requisito	**Valor Máximo/Mínimo**	**Caso de Uso**
Taxa máxima de dados - *download*	20 Gbit/s	eMBB
Taxa máxima de dados - *upload*	10 Gbit/s	eMBB
Taxa de *download* do usuário em 95% do tempo	100 Mbit/s	eMBB
Taxa de *upload* do usuário em 95% do tempo	50 Mbit/s	eMBB
Latência	1 ms	URLLC
Latência	4 ms	eMBB
Número máximo de dispositivos por área	$10^6/km^2$	mMTC

Tabela 2.1: Relação dos requisitos do 5G de forma resumida e de acordo com o caso de uso, conforme estipulados no IMT-2020 [1].

mento e configuração de *multicluster* Kubernetes, detalha o protótipo desenvolvido e avalia seu desempenho.

Capítulo 3

Automação Fim-a-Fim do *Multicluster* Kubernetes

O objetivo deste capítulo consiste em detalhar a ferramenta proposta para automação fim-a-fim de criação e configuração de *multicluster* Kubernetes, assim como descrever os aspectos de arquitetura e de segurança da informação incorporados na ferramenta. Além disso, é avaliado o desempenho da ferramenta quanto ao tempo de criação dos *clusters* e do *multicluster* Kubernetes nas nuvens da Azure e da AWS, a vazão e latência observadas nas aplicações 5G instanciadas no *multicluster*.

A ferramenta de automação fim-a-fim de criação e configuração de *multicluster* Kubernetes é implementada de forma modular, como mostra a Figura 3.1. A abordagem modular permite uma maior facilidade de manutenção e flexibilidade na escolha do conjunto de plataformas, provedores de nuvens e outras ferramentas para construção da solução proposta neste projeto. O cenário da proposta considera ambientes em múltiplas nuvens e/ou centros de dados (*data centers*) distintos, sem haver a necessidade da infraestrutura estar em um único provedor de nuvem ou domínio. Devido à heterogeneidade do cenário, é importante automatizar tudo o que for possível para abstrair a complexidade do ambiente e minimizar erros humanos.

O objetivo da ferramenta proposta é automatizar em código todos os processos necessários para criação e manutenção de um ambiente ideal para aplicações 5G. O uso das automações em código evita configurações erradas ou fora dos padrões de segurança, pois o ambiente heterogêneo aumenta sua complexidade e a possibilidade de erros humanos em processos manuais. A ferramenta abstrai completamente

o processo de provisionamento de toda infraestrutura necessária para o *multicluster* Kubernetes, além de prover mecanismos de monitoração do ambiente e controle simplificado. Além disso, ter os processos e a infraestrutura automatizados em código auxilia a documentar e mapear o ambiente, o que facilita as manutenções, atualizações e investigações de erros. Já o uso do *multicluster* Kubernetes permite uma solução robusta a indisponibilidades de região e/ou de centros de dados e aproxima as aplicações dos usuários, pois os *clusters* Kubernetes do *multicluster* estão instanciados em regiões independentes uma das outras e aumenta os pontos possíveis de conexão dos usuários com as aplicações, assim atendendo o requisito de alta disponibilidade, alta vazão, e baixa latência das aplicações 5G [1].

A arquitetura da ferramenta é dividida em 4 partes, como mostra a Figura 3.1: (i) o sistema de inicialização e manutenção, responsável pelo provisionamento de todo o *multicluster* Kubernetes, sua configuração e conservação da saúde da infraestrutura; (ii) a interface *web* de gestão do *multicluster* para facilitar o gerenciamento dos *n clusters* do ambiente; (iii) os *clusters* de controle, pontos logicamente centralizados que monitoram e mantêm todo o *multicluster* interconectado; e (iv) os *clusters* de trabalho, que executam as aplicações 5G em diferentes regiões. Visando facilitar a gestão dos *n clusters*, a ferramenta propõe um *multicluster* Kubernetes híbrido, combinando aspectos da arquitetura *multicluster* centrada na rede, como a interconexão dos *n clusters* para permitir a comunicação contêiner-contêiner, e da arquitetura centrada no controlador, como um ponto logicamente centralizado para administrar e visualizar todo o *multicluster* de forma simples.

O **Sistema de Inicialização e Manutenção** do *multicluster* Kubernetes cria toda a infraestrutura dos *n clusters* do ambiente e resolve todas as suas dependências. O *multicluster* em si não necessita estritamente deste sistema, mas o seu uso previne possíveis erros, agiliza a entrega e as mudanças da infraestrutura, e abstrai a complexidade do ambiente heterogêneo. Esse sistema é subdividido em três partes, o Subsistema de Autocura de Infraestrutura, os Módulos de Automação de Infraestrutura e os Módulos de Automação de Aplicações. O **Subsistema de Autocura de Infraestrutura** é responsável por identificar e corrigir de forma autônoma qualquer erro ou mudança indevida a nível de infraestrutura, como perda de conectividade com uma VM, concessão de acesso indevido a um recurso crítico e sensível como um cofre de chaves, alteração no tamanho de uma VM acarretando

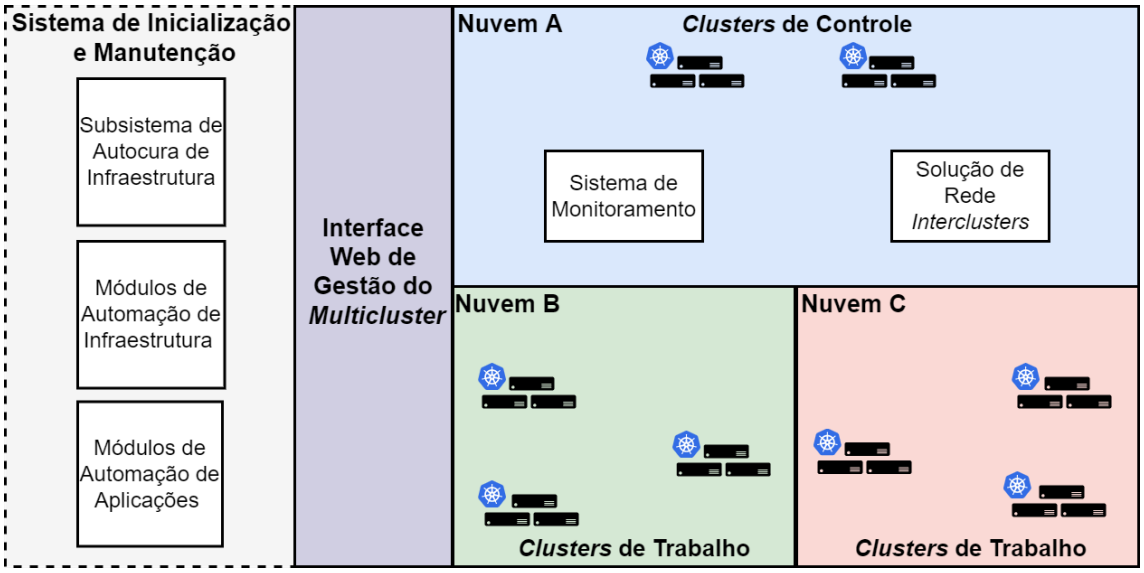


Figura 3.1: Arquitetura geral da ferramenta de automação fim-a-fim de criação e configuração de *multicluster* Kubernetes. A ferramenta é dividida em 4 partes principais, sendo elas: (i) o Sistema de Inicialização e Manutenção responsável pelo provisionamento de todo o *multicluster* Kubernetes, sua configuração e conservação da saúde da infraestrutura. A linha pontilhada que engloba esse sistema representa recursos não obrigatórios para existência do *multicluster* Kubernetes, apesar de trazerem diversos benefícios; (ii) a Interface *Web* de Gestão do *Multicluster* para facilitar o gerencialmente dos *n clusters* do ambiente; (iii) os *Clusters* de Controle (CC), ponto logicamente centralizado que monitora e mantém todo o *multicluster* interconectado. Os CCs podem estar em provedores de nuvens distintos para maior redundância; e (iv) os *Clusters* de Trabalho (CT), que executam as aplicações 5G em diferentes regiões. Os CTs podem estar em provedores de nuvens distintos para maior redundância

em aumento de custo, entre outros. Os **Módulos de Automação de Infraestrutura** são códigos reutilizáveis, similares a bibliotecas, adotando o conceito de infraestrutura como código (*Infrastructure as Code* - IaC) para provisionar e configurar toda a infraestrutura necessária para o *multicluster* Kubernetes. Por último, os **Módulos de Automação de Aplicações** também são códigos reutilizáveis, porém responsáveis por implantar e atualizar as aplicações 5G em contêineres nos *n clusters* Kubernetes. Vale ressaltar que todas as 3 partes do Sistema de Inicialização e Manutenção do *multicluster* Kubernetes são modulares. Dessa forma, qualquer uma das partes pode ser implementada utilizando ferramentas e lingua-

gens de preferência do desenvolvedor, desde que cumpram com as finalidades de cada parte e possuam integrações entre si.

A Interface *Web* de Gestão do *Multicluster* disponibiliza um meio simples de administrar uma grande quantidade de *clusters* a partir de um ponto central ou logicamente centralizado. Dependendo da ferramenta utilizada, a interface se comunica com o *multicluster* Kubernetes através dos CCs ou diretamente com cada um dos *clusters* para coletar seus estados, métricas e enviar comandos ao ambiente. Esse componente da ferramenta de automação fim-a-fim é essencial para viabilizar um controle simples de ambientes heterogêneos de grande porte. A interface pode ser instanciada nos próprios CCs para usufruir da alta disponibilidade do *multicluster*, ou pode ser um recurso externo como uma ferramenta de integração e entrega contínua (*Continuos Integration/Continuos Delivery - CI/CD*). Um ponto importante é que a Interface Web de Gestão do *Multicluster* pode estar integrada com o Sistema de Monitoramento do CC, assim disponibilizando uma visão e controle completos do *multicluster* Kubernetes em um único lugar.

Os *Clusters* de Controle (CC) são elementos característicos de arquiteturas *multicluster* Kubernetes centradas no controlador. Sua funcionalidade é controlar, interconectar e monitorar todos os *clusters* do ambiente. Nesta proposta, os CCs possuem dois componentes internos responsáveis por garantir a funcionalidade do *multicluster*, o Sistema de Monitoramento e a Solução de Rede Interclusters. O **Sistema de Monitoramento** possui visão de todos os *n clusters* do ambiente em tempo real, fornecendo métricas de consumo de recursos, rede, disponibilidade dos *clusters* Kubernetes e das suas aplicações. Alguns exemplos de métricas importantes para esse sistema são: consumo de CPU, memória e latência de cada nó dos *clusters* extraídos de seus SOs, consumo de CPU, memória, latência e vazão de cada contêiner extraídos das APIs Kubernetes de cada *cluster*. Existem várias ferramentas no mercado ou de código aberto capazes de atender as finalidades do Sistema de Monitoramento de extrair todas essas métricas, agrupá-las e construir uma interface de fácil visualização, como o Prometheus [28], Grafana [29], Thanos [30], entre outros. A **Solução de Rede Interclusters**, também chamada de malha de rede ou serviço (*network/service mesh*), é um elemento característico de arquiteturas *multicluster* Kubernetes centradas na rede. A função desse componente é permitir a comunicação das aplicações em *pods* instanciados em diferentes *clusters*,

como se fosse um *firewall*. Além disso, outra função importante desse componente é a descoberta automática de serviços (*service discovery*), pois torna-se complexo e praticamente inviável mapear onde cada aplicação está instanciada em um ambiente heterogêneo e de grande escala com *n clusters* Kubernetes. A descoberta automática de serviços fornece um servidor DNS com os registros constantemente atualizados de todas as aplicações instanciadas no *multicluster* e seus respectivos endereços IPs. Neste projeto, o uso conjunto do CC para controle logicamente centralizado de todo *multicluster* Kubernetes com a Solução de Rede Interclusters para construção da malha de rede/serviço caracteriza uma arquitetura híbrida de *multicluster* Kubernetes.

Os *Clusters* de Trabalho (CT) executam todas as aplicações do *multicluster* Kubernetes. Os CTs podem ser de qualquer tamanho e instanciados em qualquer nuvem ou centro de dados, sendo o único requisito a existência de comunicação entre os CTs e os CCs. Para aplicações 5G, os CTs devem ficar em regiões o mais próximas possíveis dos usuários finais para garantir baixa latência. Outro detalhe importante do *multicluster* Kubernetes é que as requisições dos usuários finais às aplicações 5G podem ser feitas diretamente ao *cluster* hospedando a aplicação, ou a um dos CCs que fará um *proxy* para o CT correspondente. A implantação das aplicações 5G nos CTs é feita através do Sistema de Inicialização e Manutenção.

3.1 Desenvolvimento do Protótipo da Ferramenta

Um protótipo da ferramenta proposta foi desenvolvido[1] utilizando um ambiente multinuvem para simular aproximadamente um cenário de aplicações 5G. Inicialmente, é avaliado o desempenho na criação de *clusters* Kubernetes em dois provedores de nuvem, a Azure da Microsoft[2] e a AWS da Amazon[3]. A escolha dos provedores de nuvem foi baseada no relatório de serviços de nuvem da Gartner[4], empresa renomada de consultoria na área de tecnologia. Os parâmetros de escolha

[1]A implementação está disponível em `https://github.com/JoltLeo/kubernetes_multicluster`.

[2]Disponível em `https://azure.microsoft.com`.

[3]Disponível em `https://aws.amazon.com`.

[4]Disponível em `https://www.gartner.com/reviews/market/public-cloud-iaas`.

foram a quantidade de avaliações e notas atribuídas aos provedores de nuvem do relatório da Gartner. Em questão de custo, a Microsoft forneceu uma assinatura gratuita de estudante para Azure com escopo reduzido e saldo de 100 dólares para o desenvolvimento da proposta. Já no caso da AWS foi necessário utilizar verba pessoal no total de 100 reais, pois a Amazon não oferece nenhuma assinatura gratuita que englobe o uso do seu serviço de Kubernetes. Na nuvem da Azure, foi utiliza o serviço de Kubernetes da Azure (*Azure Kubernetes Service* - AKS), enquanto na nuvem da AWS, o serviço elástico de Kubernetes (*Elastic Kubernetes Service* - EKS). Para um *cluster* Kubernetes com 2 nós com CPU Intel Xeon E5-2673 v4 CPU 2,3 GHz, 7 GB de memória RAM e disco SSD de 100 GB, o uso do AKS custa 140,16 dólares/mês e o EKS 196,71 dólares/mês. Da arquitetura do *multicluster* Kubernetes, apresentada anteriormente na Figura 3.1, foram utilizadas as seguintes tecnologias, conforme representadas na Figura 3.2.

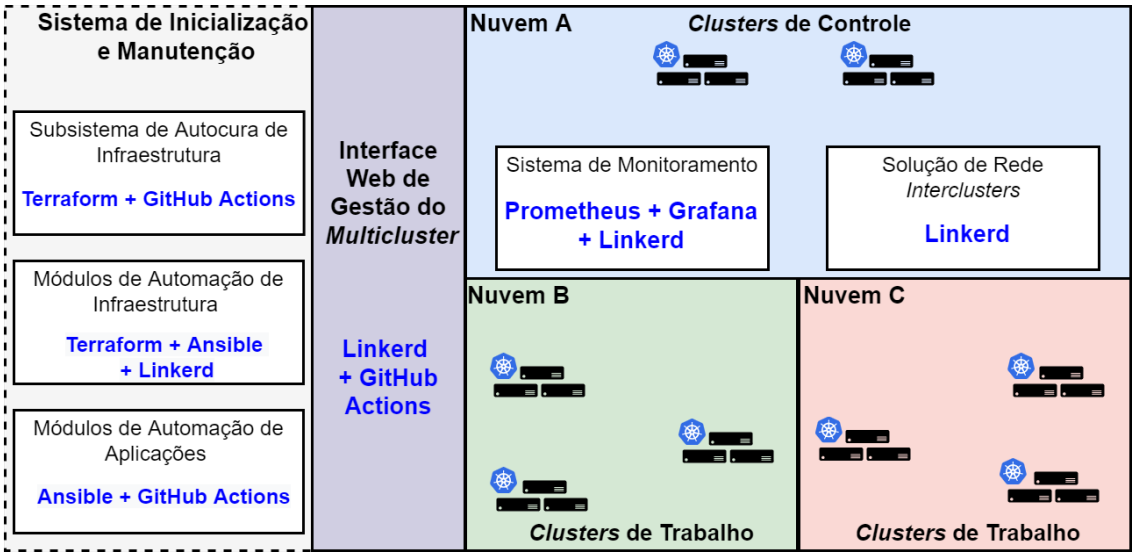


Figura 3.2: Arquitetura do protótipo da ferramenta de automação fim-a-fim de criação e configuração de *multicluster* Kubernetes. O protótipo utiliza as seguintes tecnologias: (i) no Sistema de Inicialização e Manutenção, utiliza o Terrafomn, Ansible, Linkerd e GitHub Actions; (ii) na Interface *Web* de Gestão do *Multicluster* utiliza Linkerd em conjuto com o GitHub Actions; (iii) e nos *Clusters* de Controle (CC), utiliza o Prometheus e Grafana para monitoramento, e Linkerd para solução de rede interclusters e também monitoramento.

- No **Sistema de Inicialização e Manutenção** do *multicluster* Kubernetes, foram utilizados o Terraform [7], Ansible [8], Github Actions [10], ferramen-

tas gratuitas de automação, e o Linkerd [9]. Os Módulos de Automação de Infraestrutura são feitos e executados em Terraform e Ansible. Os módulos Terraform são responsáveis por criar toda a infraestrutura necessária para o *multicluster* Kubernetes, e sua escolha foi motivada pela ferramenta possuir uma linguagem de programação declarativa de fácil entendimento, armazenar o estado de toda a infraestrutura, o que pode ser usado para monitoramento, e provisionar a infraestrutura de forma automática e a partir de código, assim tirando sua complexidade. Foram desenvolvidos 4 módulos Terraform, sendo eles: (i) módulo de criação de *clusters* Kubernetes na Azure, utilizando o serviço *Azure Kubernetes Service* - AKS; (ii) módulo de criação de *clusters* Kubernetes na AWS, utilizando o serviço *Elastic Kubernetes Service* - EKS; (iii) módulo de criação de cofre de chaves na Azure para armazenar as credenciais de conexão dos *clusters* Kubernetes instanciados em qualquer provedor de nuvem de forma segura. Vale ressaltar que o cofre de chaves poderia ser criado na AWS também, porém este projeto optou utilizar o cofre de chaves na Azure por questão de custo e por não influenciar no desempenho da ferramenta; e (iv) módulo de criação do inventário do Ansible, arquivo declarando todos os valores e senhas necessárias para configurar e conectar nos n *clusters*. Já os módulos Ansible fazem a configuração de todo o *multicluster* Kubernetes ao instalar o Linkerd e seus componentes em todos os *clusters*. O Linkerd é utilizado como solução rede interclusters (*service mesh*) para construir o *multicluster* Kubernetes. A justificativa da utilização do Ansible é similar à do Terraform, porém as diferenças são que o Ansible faz a parte de configuração de infraestrutura e não armazena nenhum estado. Os Módulos de Automação de Aplicações também utilizam Ansible, mas em conjunto com o GitHub Actions que implementa um gatilho para implantação das aplicações 5G no *multicluster* Kubernetes de forma automática. Baseado na noção de CI/CD, esse gatilho é ativado quando há qualquer alteração no diretório de aplicações 5G do repositório do código do protótipo no GitHub. Por último, o Subsistema de Autocura de Infraestrutura utiliza o Terraform e é baseado no estado armazenado pela ferramenta. O Terraform, executado através de fluxos de tarefas automáticos do GitHub Actions de 10 em 10 minutos, valida, identifica e corrige todas as divergências do estado armazenado, da infraestrutura

declara no código e do estado atual. Todos os módulos Terraform e Ansible, e os fluxos de tarefas automáticos do GitHub Actions estão no Apêndice A.

- Na **Interface *Web* de Gestão do *Multicluster***, foram utilizados os painéis de visualização (*dashboards*) do Linkerd em conjunto com fluxos de tarefas automáticos do GitHub Actions. Os painéis, representados na Figura 3.3, expõem o estado atual do *multicluster* Kubernetes contendo a relação de todas as aplicações em *pods* instanciadas no ambiente, e disponibilizam métricas dos *clusters*, como consumo de CPU, memória RAM, taxas de leitura e escrita em disco, latência e vazão de rede entre os usuários e as aplicações. O GitHub Actions implementa um gatilho para atualizar qualquer configuração do *multicluster* Kubernetes ou criar novos *clusters*. Esse gatilho é ativado quando é feita qualquer alteração no repositório do protótipo, que então executa fluxos de tarefas utilizando Terraform e o Ansible para realizar as alterações. Ambos os painéis e o GitHub Actions foram escolhidos por proporcionarem um ponto central com redundância para administrar e visualizar o *multicluster* Kubernetes.

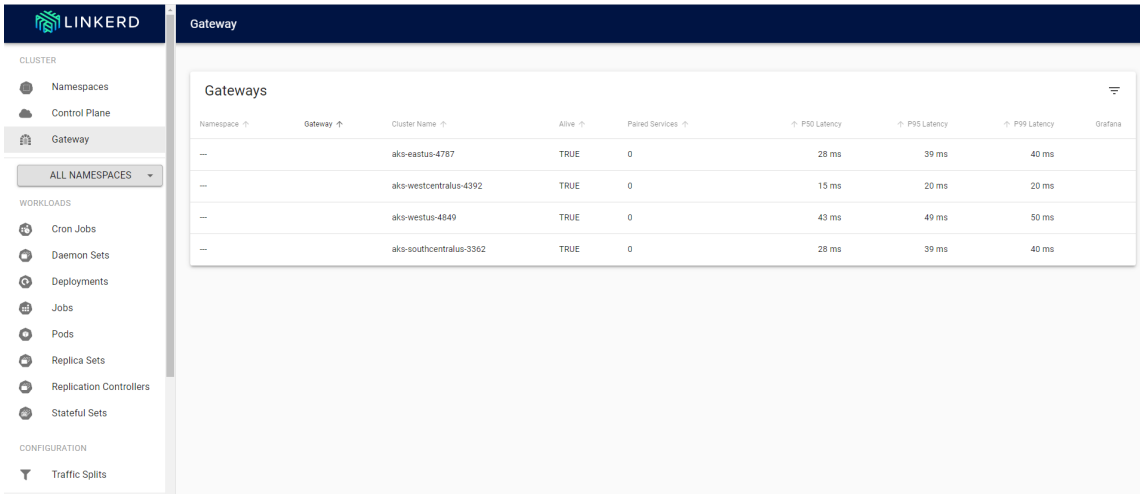
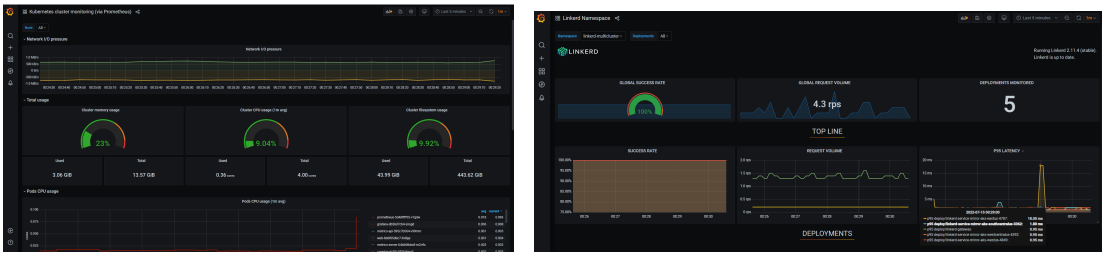


Figura 3.3: Interface web logicamente centralizada do Linkerd para visualização do *multicluster* Kubernetes. No painel é possível visualizar o estado atual de todos os *n clusters* Kubernetes e quais aplicações estão em execução em cada um dos *clusters*.

- Nos ***Clusters* de Controle (CC)**, o seu Sistema de Monitoramento foi implementado com o Prometheus [28], ferramenta de monitoração, e o Grafana [29], ferramenta de visualização e criação de *dashboards*. A escolha dessas ferra-

mentas foi motivada pela sinergia que há entre elas, pois as duas possuem integrações nativas entre si e com o Kubernetes. A Figura 3.4 representa o resultado do uso conjunto do Prometheus com o Grafana em um *dashboard*. A Solução de Rede Interclusters (*service mesh*) foi feita com o Linkerd por ser de código aberto, bem documentado e adotado pela comunidade, e por também possuir integrações nativas com o Prometheus e o Grafana.



(a) (b)

Figura 3.4: Painel de monitoração, feito com Prometheus e Grafana, de um *cluster* Kubernetes fornecendo informações de consumo de CPU, memória RAM, rede e disco (a). Painel de monitoração de todo o *multicluster* Kubernetes, também feito com Prometheus e Grafana (b).

3.2 Avaliação de Desempenho

Nesta seção, é avaliado o desempenho do protótipo da ferramenta proposta, em termos de tempo médio de criação dos *cluster* Kubernetes, tempo de execução do Terraform, tempo de configuração do *multicluster* Kubernetes, latência e vazão entre um cliente e aplicações 5G em diferentes localizações geográficas.

3.2.1 Tempo Médio de Criação dos Clusters Kubernetes

Experimento 1: O primeiro experimento consiste em avaliar o comportamento de dois provedores de nuvem utilizados neste projeto, Azure e AWS, na criação dos *clusters* e suas dependências com o Terraform. Para tal, foi medido o tempo médio de criação dos *clusters* ao variar o número de *clusters* criados em paralelo. Devido a limitações de verba e limitação de quantidade de serviços em uso das assinaturas de ambas as nuvens, só foi possível chegar no máximo de 5 *clusters* criados em paralelo, o que é um número razoável para um cenário de *multicluster* regional se comparado

à quantidade de serviços empresariais de centros de dados da Vivo[5]. Cada *cluster* Kubernetes possui 2 nós com uma CPU Intel Xeon E5-2673 v4 CPU 2,3 GHz, 7 GB de memória RAM e disco SSD de 100 GB. O experimento é feito a partir de um agente de execução do GitHub Actions com Intel Xeon E5-2673 v4 CPU 2,3 GHz, 7 GB de memória RAM e disco SSD de 14 GB. A Figura 3.5(a) apresenta o resultado do tempo médio de criação apenas dos *clusters* Kubernetes, desconsiderando suas dependências com outros recursos. O tempo médio de criação dos *clusters* Kubernetes foi adquirido dos arquivos de registro (*logs*) de execução do Terraform, sendo que o Terraform foi executado $K = 3$ vezes para cada medida. O cálculo do tempo médio de criação dos *clusters Kubernetes* está definido na Equação 3.1, onde T_n é o tempo médio da criação de n *clusters* Kubernetes em paralelo da execução k, e t_{ki} é o tempo de criação do *cluster* Kubernetes i da execução k.

$$T_n = \frac{\sum_{i=1}^{n}\sum_{i=1}^{K} t_{ki}}{3n} \tag{3.1}$$

O erro do tempo médio de criação dos *clusters* Kubernetes é calculado utilizando o erro quadrático médio, definido na Equação 3.2, onde e_{T_n} é o erro quadrático médio do tempo médio T_n de criação de n *clusters* Kubernetes em paralelo, K é o número de execuções e t_{ki} é o tempo de criação do *cluster* Kubernetes i da execução k. O erro quadrático médio é utilizado devido ao seu uso em outros trabalhos relacionados na literatura sobre *multicluster* Kubernetes.

$$e_{T_n} = \sqrt{\frac{1}{K}\sum_{i=1}^{K}(t_{ki} - T_n)^2} \tag{3.2}$$

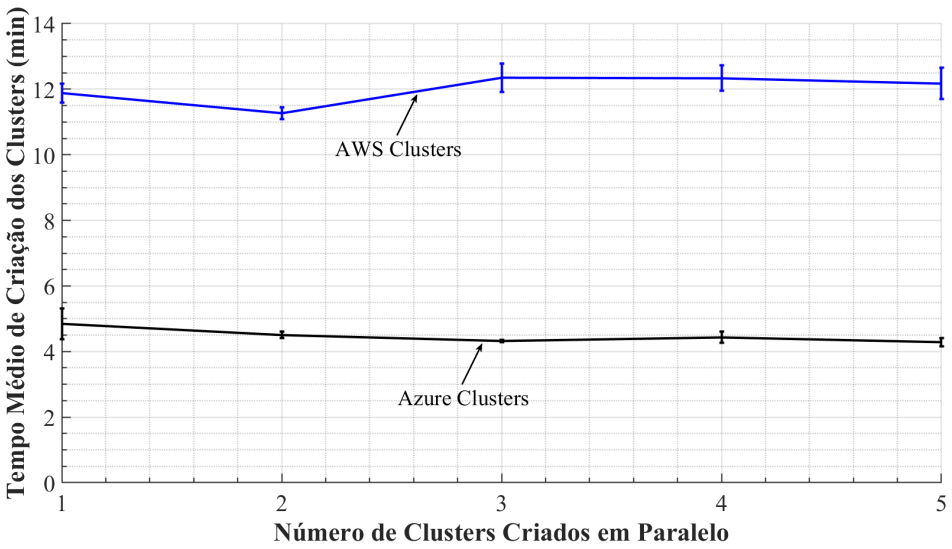
Ambas as nuvens lidam bem com o paralelismo na criação dos *clusters* Kubernetos, pois as curvas estão quase constantes. Além disso, é evidente que a criação de *clusters* na Azure é mais rápida do que na AWS, chegando a ser até 3 vezes mais rápida. A grande diferença nos tempos médios pode estar relacionada com alguma otimização da Azure para o Terraform ou abordagens diferentes de criação de *clusters* Kubernetes. A Figura 3.5(b) apresenta o resultado do tempo médio de execução do Terraform para avaliar o impacto das dependências dos *clusters* em cada nuvem, como a rede e sub-rede dos *clusters* Kubernetes, regras de *firewall*, roteadores interno

[5]Informação retirada de `https://www.vivo.com.br/para-empresas/produtos-e-servicos/digitais/data-center/data-centers`.

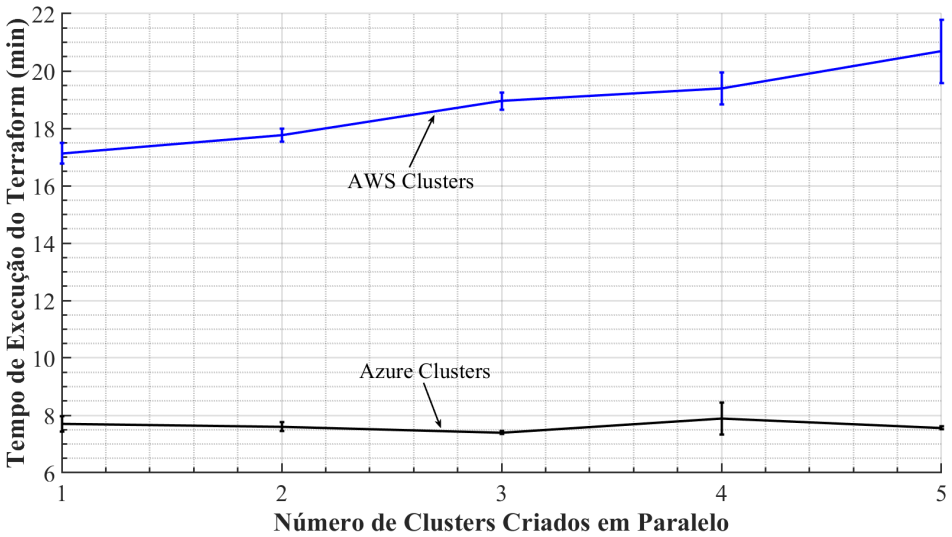
e externo, entre outros. O gráfico da Figura 3.5(b) considera a execução completa do Terraform, incluindo a criação dos *cluster* Kubernetes cujo tempo de criação é apresentado na Figura 3.5(a). É possível identificar novamente um tempo bem elevado na execução do Terraform para AWS, o que gera fortes indícios de que a AWS possui uma abordagem mais complexa na criação dos *clusters* Kubernetes do que a Azure em termos da quantidade de dependências dos *clusters*, fato observável nos *logs* de execução do Terraform, apresentados na Figura 3.6, que lista todos os recursos sendo criados ou destruídos. Outro ponto importante é que as dependências dos *clusters* Kubernetes na AWS impactam o desempenho do Terraform à medida que se aumenta o paralelismo, comportamento não identificado na Azure, que apresenta a curva praticamente constante.

3.2.2 Tempo de Configuração do Multicluster Kubernetes

Experimento 2: O segundo experimento consiste em medir o tempo de configuração do *multicluster* Kubernetes com Ansible à medida que o número n de *clusters* aumenta. Foi utilizado o mesmo cenário do experimento **1** com 2 nós por *clusters* Kubernetes e executado no agente do GitHub Actions, porém é apenas utilizada a nuvem da Azure. A configuração do *multicluster* Kubernetes é transparente à nuvem onde seus *clusters* estão hospedados, tendo apenas influência da execução do Kubernetes e do Linkerd. Portanto, foi utilizada apenas a nuvem da Azure por ter um resultado superior à AWS no experimento **1**, e por possuir assinatura gratuita de até 100 dólares para o projeto. A Figura 3.7 apresenta o resultado do tempo de configuração do *multicluster* Kubernetes. Vale pontuar que não existe *multicluster* com apenas um *cluster* Kubernetes, portanto o número de *clusters* em paralelo varia de 2 até 5. O resultado demonstra que, apesar de pouca diferença, o Ansible não possui um bom desempenho na configuração paralela dos *clusters* do *multicluster* Kubernetes. Entretanto, é possível que essa diferença seja irrelevante ou nula ao aplicar checagens de estado dos *clusters* antes de prosseguir com a configuração, pois existe dependências entre os passos da configuração via Ansible. Alguns dos passos do Ansible são comandos assíncronos que não aguardam a finalização da configuração do recurso no Kubernetes, então o código atual utiliza temporizadores para contornar esse problema e evitar erros de execução.



(a) Tempo médio de criação de clusters Kubernetes variando o número de *clusters* criados simultaneamente e desconsiderando suas dependências.

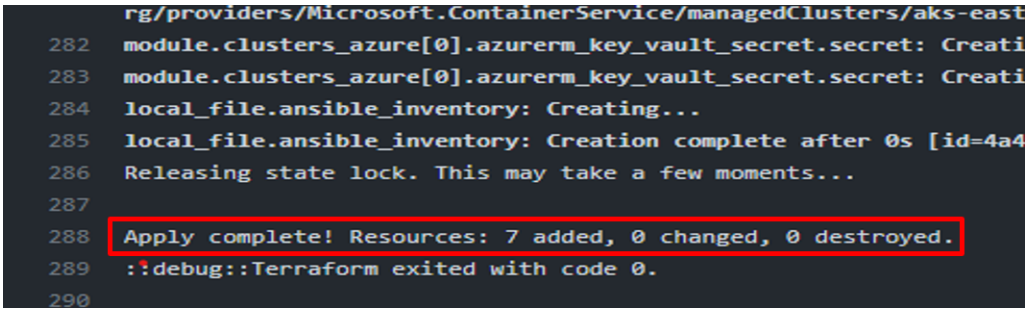


(b) Tempo médio de execução do Terraform variando o número de *clusters* criados simultaneamente para avaliar o impacto das dependências do *clusters*.

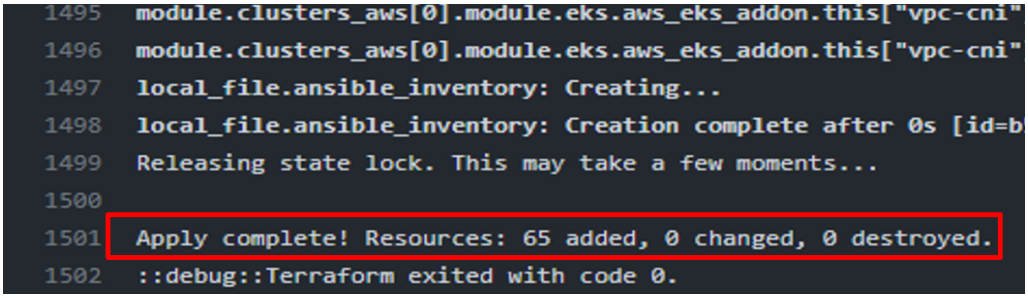
Figura 3.5: Tempo médio de criação de *clusters* Kubernetes e suas dependências na Azure e na AWS. O número de *clusters* criados simultaneamente é variado para avaliar o quão eficiente o protótipo é com paralelismo.

3.2.3 Vazão e Latência das Aplicações 5G Hospedadas no Multicluster Kubernetes

Experimento 3: O último experimento consiste em avaliar o quão próximo o protótipo está de atender os requisitos de vazão e latência de uma aplicação 5G. Foi



(a) *Log* de execução do Terraform para criação de 1 *cluster* Kubernetes na nuvem da Azure.



(b) *Log* de execução do Terraform para criação de 1 *cluster* Kubernetes na nuvem da AWS.

Figura 3.6: *Logs* de execução do Terraform para criação de 1 *cluster* Kubernetes na nuvem da Azure e AWS. Os *logs* indicam que o *cluster* Kubernetes na Azure precisa de 7 recursos de infraestrutura para ser provisionado, enquanto o *cluster* Kubernetes na AWS precisa de 65 recursos.

utilizado o mesmo cenário do experimento **2** para o *multicluster* Kubernetes com 2 nós por *cluster* e toda infraestrutura hospedada na Azure. Para este experimento, os 5 *clusters* Kubernetes foram distribuídos em 5 diferentes centros de dados da Azure nos Estados Unidos da América (EUA). O objetivo da distribuição é simular o mais próximo possível um ambiente 5G. Uma máquina virtual Intel Xeon E5-2673 v4 CPU 2,3 GHz com 7 GB de memória RAM e disco SSD de 100 GB foi criada na Azure para atuar como cliente e realizar chamadas HTTP para os 5 *clusters* do *multicluster* Kubernetes. O cenário do experimento está ilustrado no mapa da Figura 3.8. A Tabela 3.1 apresenta os resultados de latência entre um cliente no sul da Virgínia e cada um dos *clusters* em diferentes regiões dos EUA. Como esperado, a latência é menor em regiões próximas ao cliente. Mesmo quando o cliente e um dos *clusters* estão no mesmo estado, a Virgínia, a latência não atingiu o valor esperado de menor ou igual a 4 *ms* de uma aplicação 5G, porém esse valor deve ser atingido

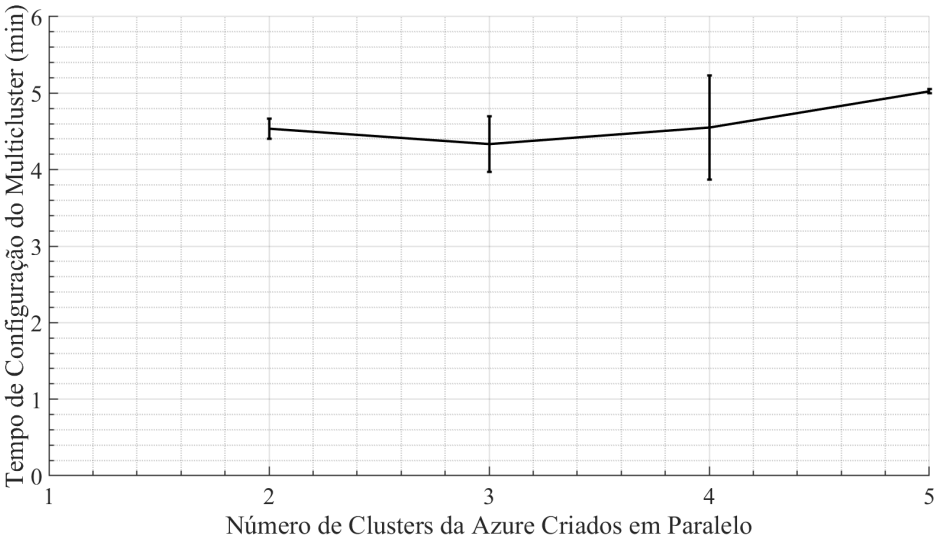


Figura 3.7: Tempo de configuração do *multicluster* Kubernetes com Ansible ao variar o números de *clusters* pertencentes ao *multicluster* para avaliar o paralelismo do protótipo.

caso o cliente estivesse mais próximo ainda do centro de dados da Azure da região Leste EUA. A vazão de *download* foi medida para 5 tamanhos de arquivos diferentes, sendo eles 1 kB, 1 MB, 5 MB, 10 MB e 20 MB. A Figura 3.9 apresenta os resultados de vazão de *download* para arquivos de diferentes tamanhos entre um cliente no sul da Virgínia e cada um dos *clusters* do *multicluster* Kubernetes em diferentes regiões. A vazão de *download* é maior para os arquivos baixados de regiões mais próximas do cliente, chegando a valores 10 vezes maiores para o cliente situado no mesmo estado do *cluster* Kubernetes. Além disso, é notado um aumento na vazão de *download* à medida que o tamanho do arquivo aumenta. Apesar de não ser visível na Figura 3.9, existe um limiar em que a vazão de *download* fica constante, pois estará atingindo o limite da banda da rede. Para apenas o *cluster* situado no Leste dos EUA, a vazão de *download* para o cliente situado no mesmo estado do *cluster*, Virgínia, atende ao requisito de taxa de *download* do usuário de 100 Mbit/s (12,5 MB/s) para uma aplicação 5G eMBB, estipulado no IMT-2020 [1] e apresentado na Tabela 2.1. Isto demonstra a importância de ter as aplicações os mais próximas possíveis dos usuários finais, uma vez que a latência é proporcional e a vazão de *download* inversamente proporcional à distância entre cliente e aplicação.

Este capítulo apresentou a arquitetura modular da ferramenta de automação

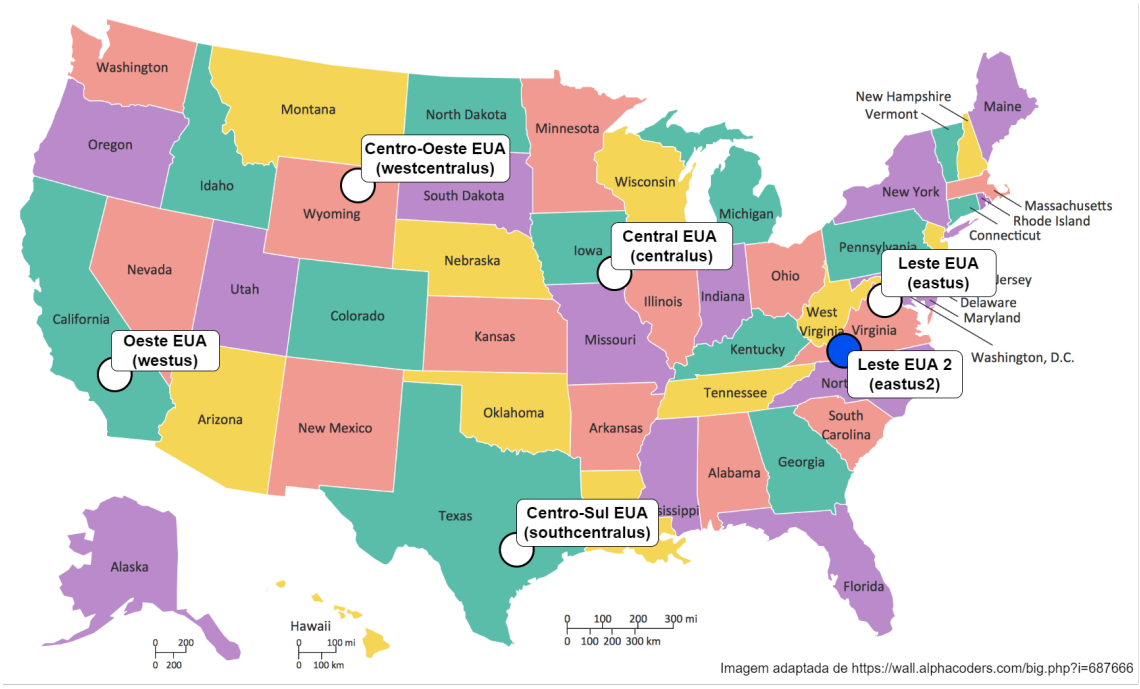


Figura 3.8: Mapa dos Estados Unidos da América com a localidade dos 5 *clusters* do *multicluster* Kubernetes utilizados no experimento *3* apresentado na Seção 3.1. Os círculos brancos representam o centro de dados onde apenas um *cluster* Kubernetes está hospedado, enquanto o círculo azul representa a máquina cliente utilizada para realizar as requisições HTTP para as aplicações 5G.

Região da Aplicação	**Estado da Região**	**Latência (ms)**
Leste dos EUA	Virgínia	$11,9 \pm 0,7$
Centro-Sul dos EUA	Texas	$55,4 \pm 2,9$
Centro dos EUA	Iowa	$56,8 \pm 5,2$
Centro-Oeste dos EUA	Wyoming	$83,6 \pm 4,1$
Oeste dos EUA	Califórnia	$123,1 \pm 3,8$

Tabela 3.1: Latência entre um cliente na região Leste dos EUA 2, em Virgínia, para cada aplicação instanciada em cada *cluster* em diferentes regiões do *multicluster* Kubernetes geodistribuído.

fim-a-fim de provisionamento e configuração de *multicluster* Kubernetes, e de seu protótipo. O protótipo utilizou diferentes ferramentas, como Terraform, Ansible, Linkerd, GitHub Action com fluxos CI/CD, Prometheus e Grafana, e construiu integrações complexas entre elas. O projeto implementa 4 módulos Terraform para provisionamento dos *clusters* Kubernetes na Azure e AWS, criação de cofre de cha-

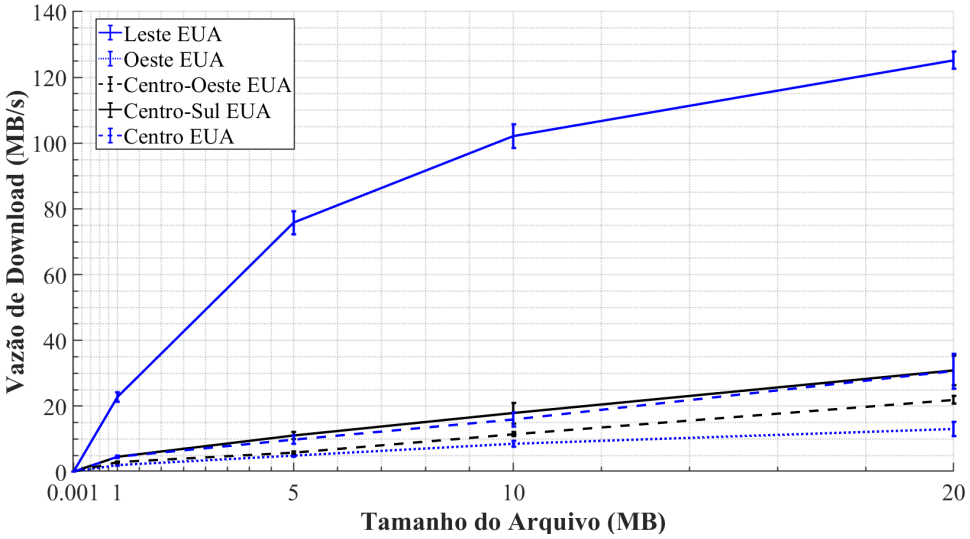


Figura 3.9: Vazão de *download* entre um cliente na região Leste dos EUA 2, em Virgínia, para cada aplicação 5G instanciada em cada *cluster* em diferentes regiões do *multicluster* Kubernetes.

ves e do arquivo de inventário para o Ansible; 2 módulos Ansible para configuração do *multicluster* Kubernetes e implantação automática das aplicações 5G no ambiente; 2 dos fluxos CI/CD para criação do *multicluster* Kubernetes e implantação e alterações automáticas das aplicações 5G no ambiente. A avaliação de desempenho demonstrou que o provisionamento dos *clusters* Kubernetes em paralelo é 3 vezes mais rápido no provedor de nuvem da Azure do que na AWS, e que ambos os provedores de nuvem não são praticamente afetadas com o aumento do número de *clusters* Kubernetes criados em paralelo. Além disso, observou-se que o tempo de execução do Terraform na Azure é mais rápido do que na AWS, indicando que os *clusters* Kubernetes criados na AWS possuem mais dependências de recursos do que aqueles criados na Azure, dado confirmado com a análise dos *logs* de execução do Terraform. Após as avaliações dos provedores de nuvem, este capítulo avaliou a configuração do *multicluster* Kubernetes com Ansible, Linkerd e GitHub Actions na Azure. Os resultados mostram que o Ansible é levemente afetado pelo aumento do número de *clusters* Kubernetes configurados em paralelo. Entretanto, é possível melhorar o desempenho do Ansible ao implementar checagens de estado dos *clusters* antes de prosseguir com os próximos passos de configuração, assim tratando as dependências entre as etapas assíncronas de configuração do *multicluster* Kubernetes que podem causar erros em etapas seguintes por não terem sido concluídas. Por

fim, este capítulo avalia a latência e vazão de *download* de arquivos de diferentes tamanhos entre um cliente na Virginia, EUA, e uma aplicação 5G conteinerizada no *multicluster* Kubernetes. O resultado de latência não atinge o valor de 4 ms requisitado pelo caso de uso eMBB do 5G, porém chega relativamente próximo a ele, permitindo concluir que a latência de 4 ms pode ser atingida caso o cliente estivesse mais próximo ainda da aplicação 5G. Já o resultado de vazão de *download* atinge o requisito de 100 $Mbit/s$ ($12,5$ MB/s) de uma aplicação 5G eMBB. A seguir, o projeto apresenta os trabalhos relacionados e o estado da arte do *multicluster* Kubernetes, realizando comparações com a ferramenta de automação fim-a-fim proposta.

Capítulo 4

Trabalhos Relacionados

O uso da arquitetura de *multicluster* Kubernetes tem sido frequente na literatura em diferentes cenários, como IoT [31], MEC [27], redes 5G [32], entre outros. A arquitetura *multicluster* Kubernetes também tem sido usada em conjunto com automações a nível de infraestrutura e de aplicação.

Neste capítulo, será apesentado o estado da arte do *multicluster* Kubernetes através de trabalhos da literatura organizados em três categorias: (i) gestão e monitoramento do *multicluster*; (ii) automações de provisionamento e configuração do *multicluster*; e (iii) uso do *multicluster* para aplicações 5G. Vale ressaltar que os trabalhos podem pertencer a mais de uma categoria, porém estão descritos neste texto apenas na categoria em que sua proposta mais se destaca.

4.1 Gestão e Monitoramento do *Multicluster* Kubernetes

Nesta seção, serão descritos trabalhos relacionados à gestão e monitoramento dos *clusters*, destacando como cada proposta lida com as dificuldades impostas pela natureza distribuída do ambiente do *multicluster*, possíveis instabilidades na rede, falhas de *hardware* nos *clusters*, falhas nas aplicações, variações de carga de requisições de clientes, necessidade de expandir ou reduzir o tamanho dos *clusters* ou das aplicações, além de destacar como as soluções acompanham o estado atual de todo o *multicluster*.

Tamiru *et al.* propõem o uso do Kubernetes *Federation* (KubeFed) para

gestão centralizada de *clusters* Kubernetes geodistribuídos em um único plano de controle, e realizam uma análise de erros causados por instabilidades de rede [33]. Elevadas perdas de pacotes e latência, e/ou perdas momentâneas de conexão, falhas transientes, em conjunto com uma má configuração dos temporizadores de verificação de saúde (*health checks*) do *multicluster* podem gerar falsos positivos sobre o estado de um *cluster*, fazendo com que o *cluster* de controle realoque as aplicações hospedadas no *cluster* aparentemente em falha. Esse evento se repete até que a configuração dos *clusters* sejam alteradas, causando indisponibilidades nas aplicações. Dessa forma, os autores identificam as principais configurações que impactam a disponibilidade dos *clusters* e das aplicações, e desenvolvem um controle por realimentação para adaptar as configurações dos *clusters* em tempo real. Antes de construírem o sistema de controle, os autores avaliam um conjunto randômico de 705 combinações de configurações do KubeFed para identificar o parâmetro que mais afeta a estabilidade dos *clusters*, assim identificando o temporizador de verificação de saúde (*health check*) do *cluster* como o mais importante. Com isso, os autores definem uma métrica de estabilidade baseada no número configurado de réplicas das aplicações, no número de réplicas das aplicações em execução no momento, na quantidade de *clusters* e no tempo do experimento. Essa métrica é então controlada via realimentação do valor do temporizador da verificação de saúde do *cluster*. Apesar de permitir uma configuração dinâmica dos *clusters*, apenas considerar aspectos de rede no controle não é suficiente para garantir a saúde e estabilidade dos *clusters* [34]. Além disso, conforme representado na Tabela 4.1, a proposta não utiliza nenhuma forma de automação de infraestrutura, e mesmo utilizando uma configuração geodistribuída, os autores não apresentam resultados que comprovem a aptidão da solução para aplicações 5G.

Lee *et al.* propõem um processo baseado em monitoração para otimizar a configuração de recursos de cada tipo de aplicação em ambientes *multicloud* e *multicluster* Kubernetes [35]. O intuito da configuração ótima é evitar o uso ineficiente dos recursos. Para tal, os autores avaliam o uso de CPU, memória, leitura/escrita em disco, latência e taxas de transmissão e recepção de pacotes para 3 tipos de aplicações, sendo elas: (i) aprendizado de máquina, (ii) web com consultas a bancos de dados, e (iii) IoT em nuvem. Para cada tipo de aplicação é considerado um parâmetro diferente para análise: para aprendizado de máquina é considerada a

quantidade de épocas (*epochs*), para web com consultas a bancos de dados é variado o número de clientes, e para IoT em nuvem, a quantidade de dispositivos. O processo envolve três etapas: (i) etapa de *profiling* para ajustar as configurações dos aplicativos a serem testados; (ii) etapa de *benchmarking* para coletar as métricas de consumo dos *clusters*, nós dos *clusters* e contêineres para avaliar comparativamente seus valores; e (iii) etapa de visualização para exportar os dados e resultados em gráficos. Os autores utilizam as ferramentas Prometheus e Nvidia *System Management Interface* (Nvidia-SMI) para monitorar e coletar métricas, respectivamente, do *multicluster* Kubernetes e de consumo de GPU baseada na utilização dos seus núcleos (*cores*). O trabalho não descreve a criação e gestão do *multicluster* Kubernetes. Trata-se de um método de estudo prévio da configuração ótima de recursos para as aplicações. Logo, não atende a ambientes reais suscetíveis a falhas dos nós Kubernetes e/ou intermitências na rede, pois não realiza uma configuração dinâmica baseada no estado atual do *multicluster*. A proposta não utiliza automações de infraestrutura ou das aplicações e, apesar de destacar o ambiente *multicloud*, os autores não informam se utilizam um ambiente geodistribuído, conforme destacados na Tabela 4.1.

Ao contrário dos artigos citados, este projeto de fim de curso propõe o uso da ferramenta gratuita e de código aberto Linkerd [9] para prover uma gestão centralizada da configuração da rede do *multicluster* Kubernetes, sem criar um ponto único de falha no caso do *cluster* de controle apresentar uma falha. Isto é possível ao se utilizar configurações de redundância disponíveis no Linkerd. Além disso, a ferramenta provê um sistema de descoberta automática das aplicações e um serviço de DNS para as mesmas. O Linkerd consulta as APIs Kubernetes de todos *clusters* para coletar o nome da aplicação e o endereço IP do contêiner da aplicação. Então, a ferramenta trata essas informações e as disponibiliza em um serviço de DNS para todo o *multicluster* Kubernetes, assim abstraindo a complexidade da infraestrutura e a localização de cada aplicação. Já para o monitoramento dos *clusters*, este projeto propõe o uso conjunto do Prometheus [28] com o Grafana [29] para monitorar e visualizar as métricas dos *clusters*, o uso de *Dashboards* do Linkerd para possibilitar controlar as aplicações e ter uma visão detalhada de cada cluster individualmente, e uma adaptação de uso do Terraform [7] para monitorar e garantir o mesmo estado da configuração de cada infraestrutura do *multicluster* Kubernetes a qualquer momento.

4.2 Automação do Provisionamento e Configuração do *Multicluster* Kubernetes

Nesta seção, são descritos trabalhos relacionados a automações a nível de infraestrutura e/ou de aplicação. As soluções de automação analisadas englobam o provisionamento e configuração da infraestrutura, e a implantação e configuração das aplicações.

Em um segundo trabalho, Tamiru *et al.* propõem uma plataforma de orquestração de aplicações para um *multicluster* Kubernetes geodistribuído, o mck8s [36]. A proposta é uma extensão do KubeFed por meio de recursos customizados (*custom resources*) e abstrações para criar políticas de agendamento, escalonamento automático de contêineres, provisionamento e escalonamento automático de *clusters* Kubernetes na nuvem, e rebalanceamento das aplicações do *multicluster* Kubernetes. O uso conjunto dessas abstrações proporciona a otimização do consumo de recursos físicos, alocação automática de aplicações nos *clusters* Kubernetes e escalonamento automático de aplicações para atender às variações nas requisições dos clientes. Entretanto, os autores não demostram resultados referentes ao provisionamento automático de *clusters* Kubernetes na nuvem e a solução ainda exige alguns passos de configuração manual para a criação do *cluster* de gerenciamento. Além disso, conforme representado na Tabela 4.1, os autores não apresentam resultados que comprovem a aptidão da solução para aplicações 5G.

Mfula *et al.* propõem uma plataforma para implantar, escalar e gerenciar *clusters* de contêineres [27]. A proposta utiliza Kubernetes para orquestrar os contêineres de cada *cluster*, Prometheus e Grafana para monitoramento e visualização das métricas do *multicluster* Kubernetes, a pilha de *software* da Elastic [37] para gerenciar e visualizar os registros (*logs*) de forma centralizada, e uma API RESTful em NodeJS, um ambiente de execução JavaScript, para servir como uma interface de controle centralizada de todos os *clusters*. Além disso, os autores automatizam a criação da infraestrutura dos *clusters* com o Terraform, enquanto a configuração e instalação de pacotes são feitas com o Ansible em conjunto com Helm [38], uma ferramenta de implantação automática de aplicações no Kubernetes. O único ponto não automatizado nesse trabalho é o chamado *cluster* de serviço, ou *cluster* de gerenciamento, que é o ponto central de administração de todo o *multicluster*

Kubernetes.

Ao contrário dos artigos citados, este projeto de fim de curso propõe uma automação fim-a-fim, sem necessitar de intervenções manuais que podem acarretar em falhas ou erros de configuração. A automação engloba o provisionamento da infraestrutura via código com o Terraform, a criação do *cluster* inicial de gerenciamento e a configuração do *multicluster* com Ansible. Qualquer mudança na infraestrutura ou implantação de aplicações são feitas de forma automática através de fluxos de trabalho (*workflows*) no GitHub Actions [10], apresentados no Apêndice A. Além disso, gatilhos (*triggers)* são configurados no repositório do git para permitir a execução automática desses fluxos de trabalho quando atendidas certas condições, baseada nos conceito de integração e entrega contínua (*Continuos Integration/Continuos Delivery - CI/CD*).

4.3 *Multicluster* Kubernetes para Aplicações 5G

Nesta seção, serão descritos trabalhos relacionados a *multicluster* Kubernetes aptos para aplicações 5G. No escopo deste projeto, um *multicluster* apto para aplicações 5G deve apresentar um ambiente geodistribuído e comprovar, mediante resultados de latência, disponibilidade e/ou velocidade das aplicações, que atende aos requisitos das aplicações 5G definidos no Capítulo 2. Dessa forma, as propostas que possuem ambientes geodistribuídos, porém não comprovam que atendem aos requisitos das aplicações 5G são marcadas como "impossível determinar" na Tabela 4.1.

Osmani *et al.* propõem integrar o KubeFed com o *Network Service Mesh* (NSM), uma ferramenta de rede para ambientes *multicloud* [32]. O uso do NSM no *multicluster* Kubernetes automatiza a configuração de rede dos contêineres, além de permitir a criação de conexões ponto a ponto (*Peer-to-Peer* - P2P) entre contêineres em diferentes *clusters* e o uso de encadeamento de funções de rede (*Service Function Chaining* - SFC. Porém, os autores não testam e apresentam resultados de um ambiente *multicloud* e geodistribuído, comum em cenários de 5G, e não utilizam automações de infraestrutura e de aplicações.

Ungureanu *et al.* propõem uma API declarativa para criação, configuração e gerenciamento de aplicações 5G na borda (*edge*), e comparam duas ferramentas

de automação de criação de *clusters* Kubernetes, o Kind [39] e o K3S [40] [41]. O *multicluster* Kubernetes proposto possui sua infraestrutura na VMware [42], uma plataforma privada de virtualização, e na AWS. Além disso, a solução utiliza o Prometheus e o Grafana para monitorar a saúde dos *clusters*, e o Linkerd para monitorar e rotear a comunicação entre serviços. Os pontos negativos da proposta são a utilização de uma nuvem privada que não possui contas de estudo ou pesquisa, a VMware, e o processo manual de criação do *cluster* inicial. Além disso, os autores não deixam claro se a proposta utiliza um ambiente geodistribuído, apesar de comprovarem com os resultados a aptidão de sua solução para aplicações 5G.

Ao contrário dos artigos citados, este projeto de fim de curso propõe utilizar o Linkerd como solução de roteamento e descoberta automática de serviços entre *clusters*, avaliando seu desempenho em um *multicluster* Kubernetes *multinuvem* e geodistribuído. Exemplos de aplicações 5G são criadas no ambiente para avaliar suas métricas de desempenho e validar se a estrutura cumpre com os requisitos de latência, disponibilidade e/ou vazão de aplicações 5G.

A Tabela 4.1 resume as características principais deste projeto em comparação com outras iniciativas correlacionadas encontradas na literatura.

	Este Projeto	**[33]**	**[36]**	**[27]**	**[35]**	**[31]**	**[32]**	**[41]**
Ambiente geodistribuído	✓	✓	✓	✓	-	✓	X	-
Ambiente *Multicloud*	✓	✓	✓	✓	✓	✓	X	✓
Apto para Aplicações 5G	✓	-	-	✓	X	-	✓	✓
Automação da Configuração das Aplicações	✓	✓	✓	✓	X	✓	X	✓
Automação do Provisionamento do *Cluster* Inicial	✓	X	X	X	X	X	X	X
Automação do Provisionamento da Infraestrutura	✓	X	✓	✓	X	X	X	✓
Gestão Centralizada dos *Clusters*	✓	✓	✓	✓	X	✓	✓	✓
Monitoramento dos *Clusters*	✓	✓	✓	✓	✓	✓	X	✓
Utiliza KubeFed	X	✓	✓	X	X	-	✓	X

Tabela 4.1: Comparação entre a literatura e a proposta deste projeto. O ✓significa que o trabalho possui a funcionalidade, **X** não possui e **-** que não é possível determinar por falta de informações.

Este capítulo apresentou os trabalhos relacionados com este projeto, discu-

tindo e comparando o estado da arte do *multicluster* Kubernetes com a ferramenta de automação fim-a-fim proposta neste projeto. Os trabalhos relacionados são analisados em 3 categorias referentes ao *multicluster* Kubernetes, sendo elas: (i) gestão e monitoramento do *multicluster*; (ii) automações de provisionamento e configuração do *multicluster*; e (iii) uso do *multicluster* para aplicações 5G. Uma tabela comparativa é apresentada resumindo os principais pontos e diferenças de cada trabalho relacionado e deste projeto. A seguir, o projeto apresenta as conclusões deste trabalho e discute trabalhos futuros.

Capítulo 5

Conclusões

A automação é fundamental para reduzir erros humanos e tempos de entrega. Grande parte das soluções atuais de *clusters* e *multiclusters* Kubernetes não possuem tais soluções ou automatizam parcialmente o processo de provisionamento e configuração da infraestrutura e das aplicações conteinerizadas. Em cenários multinuvem de escala, típicos de ambientes 5G, é inaceitável a execução manual de processos devido à alta complexidade do ambiente heterogêneo, necessidades de alta disponibilidade e respostas rápidas que podem ser prejudiciais pela ocorrência de erros, além do número elevado de microsserviços em contêineres que dificulta ainda uma gestão manual.

Este projeto de fim de curso apresentou uma arquitetura de ferramenta para automatizar fim-a-fim a criação e configuração de *multicluster* Kubernetes. A ferramenta proposta utiliza componentes modulares para facilitar manutenções e permitir uma ampla escolha de ferramentas, sistemas e linguagens para construção da solução. Este trabalho desenvolveu 4 módulos Terraform, sendo eles: (i) módulo de criação de *clusters* Kubernetes na Azure; (ii) módulo de criação de *clusters* Kubernetes na AWS; (iii) módulo de criação de cofre de chaves na Azure para armazenar as credenciais de conexão dos *clusters* Kubernetes instanciados em qualquer provedora de nuvem de forma segura; e (iv) módulo de criação do inventário do Ansible. Para configuração do *multicluster* Kubernetes e das aplicações 5G, este projeto desenvolveu 2 módulos Ansible: (i) módulo de configuração do *multicluster* Kubernetes utilizando o Linkerd, solução para interconexão dos n *clusters* para construção da malha de rede/serviço; e (ii) módulo de implantação automática das aplicações 5G

em todos os *clusters* do *multicluster* Kubernetes.

Além disso, o projeto utiliza um cenário geodistribuído com diferentes provedores de nuvem para posicionar o poder de processamento perto dos usuários finais e simular aplicações 5G. Um protótipo utilizando infraestrutura como código com Terraform e Ansible, CI/CD com GitHub Actions, sistema de monitoração com Prometheus e Grafana, e o controle e visualização do *multicluster* Kubernetes com o Linkerd foi desenvolvido e avaliado nas nuvens da Azure e da AWS. Os resultados comprovam que a nuvem da Azure provisiona a infraestrutura do *multicluster* Kubernetes em até 3 vezes mais rápido do que a nuvem da AWS. Entretanto, apesar do *multicluster* Kubernetes provisionado na Azure ter obtido uma vazão de *download* acima do requisitado, a solução cumpriu parcialmente os requisitos das aplicações 5G ao apresentar uma latência maior do que esperado. O resultado de latência ainda pode ser melhorado se avaliado um cenário cujo cliente esteja exatamente na mesma região da aplicação 5G.

Como trabalhos futuros, pretende-se melhorar os módulos Ansible implementando ações de validações em tempo real para evitar falhas nas etapas de configuração do *multicluster* Kubernetes e necessidade de uma nova tentativa de execução, assim garantindo maior robustez e velocidade no processo. O uso do *multicluster* Kubernetes na nuvem da Google, *Google Cloud Platform* (GCP), será avaliado e comparados com a Azure. Além disso, a abordagem de criação do ambiente na nuvem da AWS será analisada em detalhe para propor um método mais ágil e simples de provisionamento da infraestrutura. Por fim, pretende-se simular uma rede 5G com dispositivos Raspberry Pi como micro *clusters* Kubernetes com K3S [40], visando criar um cenário de Internet das coisas (*Internet of Things* - IoT) para 5G.

Referências Bibliográficas

[1] M.2150, R. I.-R., "Detailed specifications of the terrestrial radio interfaces of International Mobile Telecommunications-2020", 2021, Disponível em `https://www.itu.int/rec/R-REC-M.2150/en`. Acessado em 30 de Junho de 2022.

[2] CHIN, W. H., FAN, Z., HAINES, R., "Emerging technologies and research challenges for 5G wireless networks", *IEEE Wireless Communications*, v. 21, n. 2, pp. 106–112, 2014.

[3] XU, Y., GUI, G., GACANIN, H., *et al.*, "A Survey on Resource Allocation for 5G Heterogeneous Networks: Current Research, Future Trends, and Challenges", *IEEE Communications Surveys Tutorials*, v. 23, n. 2, pp. 668–695, 2021.

[4] VARGA, P., PETO, J., FRANKO, A., *et al.*, "5G support for Industrial IoT Applications — Challenges, Solutions, and Research gaps", *Sensors*, v. 20, n. 3, 2020.

[5] NEWMAN, S., *Building Microservices: Designing Fine-Grained Systems*. 1 ed. O'Reilly Media, February 2015.

[6] FOUNDATION, C. N. C., "Kubernetes: Production-Grade Container Orchestration", 2014, Disponível em `https://kubernetes.io`. Acessado em 30 de Junho de 2022.

[7] HASHICORP, "Unlocking the Cloud Operating Model: Cloud Compliance and Management", 2019, Disponível em `https://www.datocms-assets.com/2885/1597080117-terraform-com-cloud-compliancemanagement-whitepaper-v2-digital.pdf`. Acessado em 30 de Junho de 2022.

[8] HAT, R., "Ansible in Depth", 2017, Disponível em https://www.ansible.com/hubfs/pdfs/Ansible-InDepth-WhitePaper.pdf. Acessado em 30 de Junho de 2022.

[9] BUOYANT, I., "Linkerd: A Different Kind of Service Mesh", 2017, Disponível em https://linkerd.io. Acessado em 30 de Junho de 2022.

[10] MICROSOFT, "GitHub Actions: Automate your Workflow from Idea to Production", 2018, Disponível em https://github.com/features/actions. Acessado em 30 de Junho de 2022.

[11] XAVIER, M. G., NEVES, M. V., ROSSI, F. D., *et al.*, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments". In: *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 233–240, 2013.

[12] SOLTESZ, S., PÖTZL, H., FIUCZYNSKI, M. E., *et al.*, "Container-Based Operating System Virtualization: A Scalable, High-Performance Alternative to Hypervisors", v. 41, n. 3, pp. 275–287, mar 2007.

[13] INC., D., "Introduction to Container Security: Understanding the isolation properties of Docker", 2016, Disponível em https://www.docker.com/wp-content/uploads/2022/03/WP_IntrotoContainerSecurity_08.19.2016.pdf. Acessado em 30 de Junho de 2022.

[14] FOUNDATION, C. N. C., "Containerd: An Industry-standard Container Runtime with an Emphasis on Simplicity, Robustness and Portability", 2015, Disponível em https://containerd.io. Acessado em 30 de Junho de 2022.

[15] FOUNDATION, C. N. C., "CRI-O: Lightweight Container Runtime for Kubernetes", 2017, Disponível em https://cri-o.io. Acessado em 30 de Junho de 2022.

[16] ONGARO, D., OUSTERHOUT, J., "In Search of an Understandable Consensus Algorithm". In: *2014 USENIX Annual Technical Conference (USENIX ATC 14)*, pp. 305–319, Philadelphia, PA, 2014.

[17] REBELLO, G., CAMILO, G., SILVA, L., *et al.*, "Correntes de Blocos: Algoritmos de Consenso e Implementação na Plataforma Hyperledger Fabric", *38º Jornada de Atualização em Informática (JAI) do XXXIX Congresso da Sociedade Brasileira de Computação (CSBC 2019)*, pp. 93–148, 2019.

[18] ESPINEL SARMIENTO, D., LEBRE, A., NUSSBAUM, L., *et al.*, "Decentralized SDN Control Plane for a Distributed Cloud-Edge Infrastructure: A Survey", *IEEE Communications Surveys Tutorials*, v. 23, n. 1, pp. 256–281, 2021.

[19] LARSSON, L., GUSTAFSSON, H., KLEIN, C., *et al.*, "Decentralized Kubernetes Federation Control Plane". In: *IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, pp. 354–359, 2020.

[20] GROUP, K. M. S. I., "Kubernetes Cluster Federation (KubeFed)", 2018, Disponível em `https://github.com/kubernetes-sigs/kubefed`. Acessado em 30 de Junho de 2022.

[21] HASHICORP, "Service Mesh and Microservices Networking", 2018, Disponível em `https://www.datocms-assets.com/2885/1536681707-consulwhitepaperaug2018.pdf`. Acessado em 30 de Junho de 2022.

[22] RAHMAN, A., MAHDAVI-HEZAVEH, R., WILLIAMS, L., "A Systematic Mapping Study of Infrastructure as Code Research", *Information and Software Technology*, v. 108, pp. 65–77, 2019.

[23] MORRIS, K., *Infrastructure as Code*. O'Reilly Media, 2020.

[24] PROGRESS, "Chef: Automation Software for Continuous Delivery of Secure Applications and Infrastructure", 2009, Disponível em `https://www.chef.io`. Acessado em 30 de Junho de 2022.

[25] PUPPET, "Puppet: Make your Infrastructure Work for You", 2009, Disponível em `https://puppet.com`. Acessado em 30 de Junho de 2022.

[26] HASSAN, N., YAU, K.-L. A., WU, C., "Edge Computing in 5G: A Review", *IEEE Access*, v. 7, pp. 127276–127289, 2019.

[27] MFULA, H., YLÄ-JÄÄSKI, A., NURMINEN, J. K., "Seamless Kubernetes Cluster Management in Multi-Cloud and Edge 5G Applications". In: *International Conference on High Performance Computing & Simulation*, 2021.

[28] SOUNDCLOUD, "Prometheus: From Metrics to Insight", 2012, Disponível em `https://prometheus.io`. Acessado em 30 de Junho de 2022.

[29] ÖDEGAARD, T., "Grafana", 2014, Disponível em `https://grafana.com`. Acessado em 30 de Junho de 2022.

[30] FOUNDATION, C. N. C., "Thanos: Open Source, Highly Available Prometheus Setup With Long Term Storage Capabilities", 2018, Disponível em `https://thanos.io`. Acessado em 30 de Junho de 2022.

[31] JAVED, A., ROBERT, J., HELJANKO, K., *et al.*, "IoTEF: A Federated Edge-Cloud Architecture for Fault-Tolerant IoT Applications", *Journal of Grid Computing*, v. 18, 2020.

[32] OSMANI, L., KAUPPINEN, T., KOMU, M., *et al.*, "Multi-Cloud Connectivity for Kubernetes in 5G Networks", *IEEE Communications Magazine*, v. 59, n. 10, pp. 42–47, 2021.

[33] TAMIRU, M. A., PIERRE, G., TORDSSON, J., *et al.*, "Instability in Geo-Distributed Kubernetes Federation: Causes and Mitigation". In: *28th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pp. 1–8, 2020.

[34] SAYFAN, G., *"Mastering Kubernetes"*. Packt Publishing Ltd, 2017.

[35] LEE, S., SON, S., HAN, J., *et al.*, "Refining Micro Services Placement over Multiple Kubernetes-orchestrated Clusters employing Resource Monitoring". In: *IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, pp. 1328–1332, 2020.

[36] TAMIRU, M., PIERRE, G., TORDSSON, J., *et al.*, "MCK8S: An Orchestration Platform for Geo-distributed Multi-cluster Environments". In: *30th International Conference on Computer Communications and Networks (ICCCN)*, 2021.

[37] ELASTIC, "Elastic Stack", 2010, Disponível em `https://www.elastic.co/pt/elastic-stack`. Acessado em 30 de Junho de 2022.

[38] FOUNDATION, C. N. C., "Helm: The Package Manager for Kubernetes", 2016, Disponível em `https://helm.sh`. Acessado em 30 de Junho de 2022.

[39] FOUNDATION, C. N. C., "Kind", 2018, Disponível em `https://kind.sigs.k8s.io`. Acessado em 30 de Junho de 2022.

[40] FOUNDATION, C. N. C., "K3S: Lightweight Kubernetes", 2018, Disponível em `https://k3s.io`. Acessado em 30 de Junho de 2022.

[41] UNGUREANU, O.-M., VLADEANU, C., KOOIJ, R., "Collaborative Cloud - Edge: A Declarative API orchestration model for the NextGen 5G Core". In: *2021 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pp. 124–133, 2021.

[42] VMWARE, "VMware Infrastructure 3 Pricing, Packaging and Licensing Overview", 2008, Disponível em `https://www.vmware.com`. Acessado em 30 de Junho de 2022.

Apêndice A

Código-fonte da ferramenta proposta

Código do módulo Terraform de criação de *cluster* Kubernetes na Azure.

```
1
2  variable "env" {
3    type = string
4    validation {
5      condition     = contains(["dev", "hom", "prd"], var.env)
6      error_message = "Allowed values are 'dev', 'hom' and 'prd'."
7    }
8    description = "Cluster environment. It will be used to tag all
       resources."
9  }
10
11 variable "clusters_region" {
12   type        = string
13   default     = "brazilsouth"
14   description = "Azure region of the cluster"
15 }
16
17 variable "vault_id" {
18   type        = string
19   description = "Key vault to store kube-config"
20 }
21
```

```
22 variable "node_size" {
23   type         = string
24   default      = "Standard_D2_v2"
25   description = "Cluster VM node size."
26 }
27
28 variable "kubernetes_version" {
29   type         = string
30   default      = "1.22.6"
31   description = "Kubernetes cluster version"
32 }
33
34 variable "number_nodes_per_cluster" {
35   type     = number
36   default = 2
37 }
38
39 terraform {
40   required_providers {
41     azurerm = {
42       source  = "hashicorp/azurerm"
43       version = ">=2.29.0"
44     }
45
46     random = {
47       source  = "hashicorp/random"
48       version = ">=3.1.0"
49     }
50   }
51   required_version = ">= 1.0.0"
52 }
53
54 resource "random_integer" "random_id" {
55   min = 1
56   max = 5000
57 }
58
59 resource "azurerm_resource_group" "rg_cluster" {
60   name     = "aks-${var.clusters_region}-${random_integer.random_id.
```

```
      result}-rg"
61    location = var.clusters_region
62  }
63
64  resource "azurerm_kubernetes_cluster" "az_cluster" {
65    name                = "aks-${var.clusters_region}-${random_integer.
        random_id.result}"
66    location            = azurerm_resource_group.rg_cluster.location
67    resource_group_name = azurerm_resource_group.rg_cluster.name
68    dns_prefix          = "aks${var.clusters_region}${random_integer.
        random_id.result}"
69    kubernetes_version  = var.kubernetes_version
70
71    default_node_pool {
72      name       = "akspool${random_integer.random_id.result}"
73      node_count = var.number_nodes_per_cluster
74      vm_size    = var.node_size
75    }
76
77    identity {
78      type = "SystemAssigned"
79    }
80
81    tags = {
82      env = var.env
83    }
84  }
85
86  resource "azurerm_key_vault_secret" "secret" {
87    name         = "aks-${var.clusters_region}-${random_integer.random_id
        .result}"
88    value        = azurerm_kubernetes_cluster.az_cluster.kube_config_raw
89    key_vault_id = var.vault_id
90
91    tags = {
92      env = var.env
93    }
94  }
95
```

```
96 output "cluster_name" {
97   description = "AKS cluster name"
98   value       = "aks-${var.clusters_region}-${random_integer.random_id.
       result}"
99 }
100
101 output "cluster_endpoint" {
102   description = "Endpoint for AKS control plane."
103   value       = azurerm_kubernetes_cluster.az_cluster.fqdn
104 }
105
106 output "client_certificate" {
107   value = azurerm_kubernetes_cluster.az_cluster.kube_config.0.client_
       certificate
108 }
109
110 output "kube_config" {
111   value     = azurerm_kubernetes_cluster.az_cluster.kube_config_raw
112   sensitive = true
113 }
114
115 output "region" {
116   description = "AKS region"
117   value       = var.clusters_region
118 }
```

Listing A.1: Código-fonte do módulo Terrafom para criação de *clusters* Kubernetes na nuvem da Azure.

Código do módulo Terraform de criação de *cluster* Kubernetes na AWS.

```
1  variable "cluster_region" {
2    type        = string
3    default     = "us-east-2"
4    description = "AWS region of the cluster"
5  }
6
7  variable "vpc_cidr" {
8    type        = string
9    default     = "10.0.0.0/16"
10   description = "CIDR for cluster VPC"
```

```
}

variable "vpc_private_ip" {
  type        = list(string)
  default     = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  description = "CIDR list for private subnet"
}

variable "vpc_public_ip" {
  type        = list(string)
  default     = ["10.0.4.0/24", "10.0.5.0/24", "10.0.6.0/24"]
  description = "CIDR list for public subnet"
}

variable "kubernetes_version" {
  type        = string
  default     = "1.21"
  description = "Kubernetes cluster version"
}

variable "node_size" {
  type        = string
  default     = "t2.large"
  description = "Cluster VM node size."
}

variable "vault_id" {
  type        = string
  description = "Key vault to store kube-config"
}

variable "number_nodes_per_cluster" {
  type    = number
  default = 2
}

variable "env" {
  type = string
  validation {
```

```
50      condition     = contains(["dev", "hom", "prd"], var.env)
51      error_message = "Allowed values are 'dev', 'hom' and 'prd'."
52    }
53    description = "Cluster environment. It will be used to tag all
          resources."
54  }
55
56  data "aws_availability_zones" "available" {}
57
58  module "vpc" {
59    source                = "terraform-aws-modules/vpc/aws"
60    version               = ">=3.2.0"
61    name                  = "eks-${random_string.suffix.result}-${var.env
          }-vpc"
62    cidr                  = var.vpc_cidr
63    azs                   = data.aws_availability_zones.available.names
64    private_subnets       = var.vpc_private_ip
65    public_subnets        = var.vpc_public_ip
66    enable_nat_gateway    = true
67    single_nat_gateway    = true
68    enable_dns_hostnames = true
69
70    tags = {
71      "kubernetes.io/cluster/${local.cluster_name}" = "shared"
72    }
73
74    public_subnet_tags = {
75      "kubernetes.io/cluster/${local.cluster_name}" = "shared"
76      "kubernetes.io/role/elb"                      = "1"
77    }
78
79    private_subnet_tags = {
80      "kubernetes.io/cluster/${local.cluster_name}" = "shared"
81      "kubernetes.io/role/internal-elb"             = "1"
82    }
83  }
84
85  resource "aws_security_group" "worker_group_mgmt_one" {
86    name_prefix = "worker_group_mgmt_one_${random_string.suffix.result
```

```
      }-${var.env}"
  vpc_id       = module.vpc.vpc_id

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"

    cidr_blocks = [
      var.vpc_cidr,
      "0.0.0.0/0",
    ]
  }
}

terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "3.74.0"
    }
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">=2.29.0"
    }
    random = {
      source  = "hashicorp/random"
      version = ">=3.1.0"
    }

    local = {
      source  = "hashicorp/local"
      version = ">=2.1.0"
    }

    null = {
      source  = "hashicorp/null"
      version = ">=3.1.0"
    }
```

```
125
126     kubernetes = {
127       source  = "hashicorp/kubernetes"
128       version = ">=2.0.1"
129     }
130   }
131   required_version = ">= 1.0.0"
132 }
133
134 resource "random_string" "suffix" {
135   length  = 4
136   special = false
137 }
138
139 locals {
140   cluster_name = "eks-${random_string.suffix.result}-${var.env}"
141   kubeconfig = yamlencode({
142     apiVersion      = "v1"
143     kind            = "Config"
144     current-context = "terraform"
145     clusters = [{
146       name = module.eks.cluster_id
147       cluster = {
148         certificate-authority-data = module.eks.cluster_certificate_
              authority_data
149         server                     = module.eks.cluster_endpoint
150       }
151     }]
152     contexts = [{
153       name = "terraform"
154       context = {
155         cluster = module.eks.cluster_id
156         user    = "terraform"
157       }
158     }]
159     users = [{
160       name = "terraform"
161       user = {
162         token = data.aws_eks_cluster_auth.cluster.token
```

```
163         }
164       }]
165     })
166 }
167
168 /* Creating EKS*/
169
170 module "eks" {
171   source          = "terraform-aws-modules/eks/aws"
172   version         = ">=17.24.0"
173   cluster_name    = local.cluster_name
174   cluster_version = var.kubernetes_version
175   subnet_ids      = module.vpc.private_subnets
176
177   cluster_endpoint_private_access = true
178   cluster_endpoint_public_access  = true
179
180   vpc_id = module.vpc.vpc_id
181
182   cluster_addons = {
183     coredns = {
184       resolve_conflicts = "OVERWRITE"
185     }
186     kube-proxy = {}
187     vpc-cni = {
188       resolve_conflicts = "OVERWRITE"
189     }
190   }
191
192   eks_managed_node_group_defaults = {
193     disk_size       = 50
194     instance_types = [var.node_size]
195   }
196
197   eks_managed_node_groups = {
198     blue = {}
199     green = {
200       min_size      = var.number_nodes_per_cluster
201       max_size      = var.number_nodes_per_cluster
```

```
202         desired_size = var.number_nodes_per_cluster
203
204         instance_types = [var.node_size]
205         capacity_type  = "SPOT"
206       }
207     }
208 }
209
210 data "aws_eks_cluster_auth" "cluster" {
211   name = module.eks.cluster_id
212 }
213
214 output "cluster_id" {
215   description = "EKS cluster ID."
216   value       = module.eks.cluster_id
217 }
218
219 output "cluster_name" {
220   description = "EKS cluster name."
221   value       = local.cluster_name
222 }
223
224 output "cluster_endpoint" {
225   description = "Endpoint for EKS control plane."
226   value       = module.eks.cluster_endpoint
227 }
228
229 output "cluster_security_group_id" {
230   description = "Security group ids attached to the cluster control
        plane."
231   value       = module.eks.cluster_security_group_id
232 }
233
234 output "kube_config" {
235   description = "kubectl config as generated by the module."
236   value       = local.kubeconfig
237 }
238
239 output "region" {
```

```
240   description = "AWS region"
241   value       = var.cluster_region
242 }
```

Listing A.2: Código-fonte do módulo Terrafom para criação de *clusters* Kubernetes na nuvem da AWS.

Código do módulo Terraform de criação de cofre de chaves na Azure.

```
1  variable "project_key" { type = string }
2  variable "vault_identifier" {
3    type = string
4    validation {
5      condition     = length(var.vault_identifier) < 16
6      error_message = "The vault_identifier length must be shorter than
           15."
7    }
8  }
9  variable "env" {
10   type = string
11   validation {
12     condition     = contains(["dev", "hom", "prd"], var.env)
13     error_message = "Allowed values are 'dev', 'hom' and 'prd'."
14   }
15 }
16
17 variable "secret_admins" {
18   type    = list(string)
19   default = []
20 }
21 variable "area_name" {
22   type    = string
23   default = ""
24 }
25 variable "secret_readers_apps" {
26   type        = list(string)
27   default     = []
28   description = "List of service principals with ``get`` and ``list``
       permissions on secrets. Defaults to empty list"
29 }
30 variable "certificate_managers_apps" {
```

```
31   type        = list(string)
32   default     = []
33   description = "List of service principals with ``management``
         permissions on certificates. Defaults to empty list"
34 }
35 variable "secret_readers_groups" {
36   type        = list(string)
37   default     = []
38   description = "List of AD groups with ``get`` and ``list``
         permissions on secrets. Defaults to empty list"
39 }
40
41 terraform {
42   required_providers {
43     azurerm = {
44       source  = "hashicorp/azurerm"
45       version = ">=2.29.0"
46     }
47     azuread = {
48       source  = "hashicorp/azuread"
49       version = ">1.0.0"
50     }
51   }
52   required_version = ">= 1.0.0"
53 }
54
55 data "azurerm_client_config" "current" {}
56
57 data "azurerm_resource_group" "rg" {
58   name = var.project_key
59 }
60
61 resource "azurerm_key_vault" "key_vault" {
62   name                        = "pf-${var.vault_identifier}-${var.env}"
63   location                    = data.azurerm_resource_group.rg.location
64   resource_group_name         = data.azurerm_resource_group.rg.name
65   enabled_for_disk_encryption = true
66   tenant_id                   = data.azurerm_client_config.current.
         tenant_id
```

```
67    purge_protection_enabled     = false
68
69    sku_name = "standard"
70
71    network_acls {
72      default_action = "Allow"
73      bypass         = "AzureServices"
74    }
75
76    tags = {
77      env = var.env
78    }
79  }
80
81  resource "azurerm_key_vault_access_policy" "devops_agent" {
82    key_vault_id = azurerm_key_vault.key_vault.id
83    tenant_id    = data.azurerm_client_config.current.tenant_id
84    object_id    = data.azurerm_client_config.current.object_id
85    secret_permissions = [
86      "Set",
87      "Get",
88      "List",
89      "Delete",
90      "Purge",
91      "Recover",
92    ]
93    certificate_permissions = [
94      "Get",
95      "List",
96      "GetIssuers",
97      "ListIssuers",
98    ]
99  }
100
101 resource "azurerm_key_vault_access_policy" "secret_admins" {
102   for_each = toset(var.secret_admins)
103
104   key_vault_id = azurerm_key_vault.key_vault.id
105   tenant_id    = data.azurerm_client_config.current.tenant_id
```

```
106   object_id     = each.key
107   secret_permissions = [
108     "Set",
109     "Get",
110     "List",
111     "Delete",
112     "Purge",
113     "Recover",
114   ]
115 }
116
117 /* Assign kv readers */
118 data "azuread_service_principal" "secret_readers" {
119   for_each     = toset(var.secret_readers_apps)
120   display_name = each.key
121 }
122
123 data "azuread_group" "secret_readers" {
124   for_each     = toset(var.secret_readers_groups)
125   display_name = each.key
126 }
127
128 data "azuread_service_principal" "certificate_managers" {
129   for_each     = toset(var.certificate_managers_apps)
130   display_name = each.key
131 }
132
133 resource "azurerm_role_assignment" "readers" {
134   for_each = merge(data.azuread_service_principal.secret_readers, data.
        azuread_group.secret_readers, data.azuread_service_principal.
        certificate_managers)
135
136   scope                = azurerm_key_vault.key_vault.id
137   role_definition_name = "Reader"
138   principal_id         = each.value.object_id
139 }
140
141 resource "azurerm_role_assignment" "admin_readers" {
142   for_each = toset(var.secret_admins)
```

```
143
144   scope                = azurerm_key_vault.key_vault.id
145   role_definition_name = "Reader"
146   principal_id         = each.key
147 }
148
149 resource "azurerm_key_vault_access_policy" "readers" {
150   for_each = merge(data.azuread_service_principal.secret_readers, data.
          azuread_group.secret_readers)
151
152   key_vault_id = azurerm_key_vault.key_vault.id
153   tenant_id    = data.azurerm_client_config.current.tenant_id
154   object_id    = each.value.object_id
155
156   secret_permissions = [
157     "Get",
158     "List",
159   ]
160 }
161
162 resource "azurerm_key_vault_access_policy" "certificate_managers" {
163   for_each = data.azuread_service_principal.certificate_managers
164
165   key_vault_id = azurerm_key_vault.key_vault.id
166   tenant_id    = data.azurerm_client_config.current.tenant_id
167   object_id    = each.value.object_id
168
169   certificate_permissions = [
170     "Get",
171     "Backup",
172     "Create",
173     "Delete",
174     "DeleteIssuers",
175     "Import",
176     "ManageContacts",
177     "ManageIssuers",
178     "Purge",
179     "Recover",
180     "Restore",
```

```
181       "Update",
182       "SetIssuers",
183       "List",
184       "GetIssuers",
185       "ListIssuers",
186     ]
187 }
188
189 output "key_vault_id" {
190   value = azurerm_key_vault.key_vault.id
191 }
192
193 output "key_vault_name" {
194   value = azurerm_key_vault.key_vault.name
195 }
```

Listing A.3: Código-fonte do módulo Terrafom para criação de cofre de chaves (*key vaults*) na nuvem da Azure.

Código do módulo Terraform de criação do inventário do Ansible.

```
1  variable "triggers" {
2    type        = map(any)
3    description = "Triggers to run ansible"
4    default     = {}
5  }
6
7  variable "hosts" {
8    description = "Hosts List"
9    sensitive   = true
10   type        = list(any)
11 }
12
13 variable "extra_vars" {
14   type        = map(any)
15   description = "Extra Parameters to ansible inventory"
16   default     = {}
17 }
18
19 variable "run_playbook" {
20   type        = string
```

```
21   default     = ""
22   description = "Playbook to run"
23 }
24
25 variable "inventory_filename" {
26   type    = string
27   default = "ansible-inventory"
28 }
29
30 variable "python_interpreter" {
31   type    = string
32   default = "/usr/bin/python3"
33 }
34
35 variable "galaxy_install_collections" {
36   type    = list(string)
37   default = []
38 }
39
40 variable "galaxy_install_roles" {
41   type    = list(string)
42   default = []
43 }
44
45 variable "galaxy_requirements" {
46   type    = string
47   default = ""
48 }
49
50
51 locals {
52   inventory = templatefile(
53     "${path.module}/ansible-inventory.tmpl", {
54       hosts      = var.hosts
55       extra_vars = var.extra_vars
56     }
57   )
58   inventory_command = "echo '${base64encode(local.inventory)}' | base64
       -d  > ${var.inventory_filename}"
```

```
59 }
60
61 resource "null_resource" "inventory_file" {
62   count = var.run_playbook != "" ? 1 : 0
63
64   triggers = merge({
65     inventory = local.inventory
66     playbook  = file(var.run_playbook)
67   }, var.triggers)
68
69   provisioner "local-exec" {
70     command = local.inventory_command
71   }
72 }
73
74 resource "null_resource" "run_ansible" {
75   count = var.run_playbook != "" ? 1 : 0
76
77   triggers = merge({
78     inventory = local.inventory
79     playbook  = file(var.run_playbook)
80   }, var.triggers)
81
82   provisioner "local-exec" {
83     command = <<EOT
84       export ANSIBLE_HOST_KEY_CHECKING=False
85       export ANSIBLE_PIPELINING=1
86       export ANSIBLE_SSH_RETRIES=3
87       export ANSIBLE_TIMEOUT=20
88       export ANSIBLE_SSH_EXTRA_ARGS="-o StrictHostKeyChecking=no -o
             UserKnownHostsFile=/dev/null"
89       export ANSIBLE_PYTHON_INTERPRETER=${var.python_interpreter}
90       export ANSIBLE_COLLECTIONS_PATHS=./
91       export ANSIBLE_ROLES_PATH=./ansible_roles
92
93       %{if var.galaxy_requirements != ""}
94       ansible-galaxy install -r ${var.galaxy_requirements} --force
95       %{endif}
96
```

```
97        %{for collection in var.galaxy_install_collections~}
98        ansible-galaxy collection install ${collection}
99        %{endfor~}
100
101       echo "Check connection on ansible hosts"
102       ansible -o -i ${var.inventory_filename} -m wait_for_connection
              all -a "delay=10 timeout=180"
103
104
105       %{for role in var.galaxy_install_roles~}
106       ansible-galaxy install ${role} --force
107       %{endfor~}
108
109       ansible-playbook -i ${var.inventory_filename} ${var.run_playbook}
110     EOT
111   }
112
113   depends_on = [null_resource.inventory_file]
114 }
115
116 output "inventory" {
117   value     = local.inventory
118   sensitive = true
119 }
120
121 output "host_ip_list" {
122   value = flatten(var.hosts.*.ip_address)
123 }
124
125 ### ansible-inventory.tmpl
126 [all]
127 %{ for host in hosts }
128 ${host.name} ansible_host=${host.ip_address} ${lookup(host, "extra_vars
      ", "")}
129 %{ endfor }
130
131 %{ for group in coalescelist(distinct(flatten(hosts.*.group)), ["group
      "])~}
132 [${group}]
```

```
%{ for host in hosts ~}
%{ if host.group == "${group}" && host.group != "all" ~}
${host.name}
%{ endif ~}
%{~ endfor ~}
%{ endfor ~}

[all:vars]
ansible_ssh_common_args="-o StrictHostKeyChecking=no -o
    UserKnownHostsFile=/dev/null"
%{for var_name, var_value in extra_vars~}
${var_name}="${var_value}"
%{endfor~}

### ansible-inventory.tpl
[machines]
%{for host in hosts~}
${host.name} ansible_host=${host.ip_address}
%{endfor~}

[machines:vars]
ansible_ssh_common_args="-o StrictHostKeyChecking=no -o
    UserKnownHostsFile=/dev/null"
%{for var_name, var_value in extra_vars~}
${var_name}="${var_value}"
%{endfor~}
```

Listing A.4: Código-fonte do módulo Terrafom para criação do inventário do Ansible.

Código do módulo Ansible para configuração do *multicluster* Kubernetes.

```
- hosts: all
  become: yes
  environment:
    KUBECONFIG: "{{ kubeconfig_env }}"

  pre_tasks:

    - name: Create kubeconfig files
```

```
9         ansible.builtin.copy:
10          dest: "{{ index }}.yml"
11          content: |
12            {{ item | b64decode }}
13        loop: "{{ kubeconfigs_b64.split(',') }}"
14        loop_control:
15          index_var: index
16        no_log: true
17
18    - name: Get all kubernetes cluster contexts
19      command: |
20        kubectl config get-contexts
21      register: contexts
22
23    - name: Show all kubernetes cluster contexts
24      debug: var=contexts.stdout
25
26  tasks:
27
28    - name: Generate Linkerd certificates
29      command: "step certificate create root.linkerd.
          cluster.local root.crt root.key --profile
          root-ca --no-password --insecure"
30      register: result_1
31      retries: 2
32      delay: 10
33      until: result_1 is not failed
34
35    - name: Generate Linkerd issuer credentials
        certificates
36      command: "step certificate create identity.
          linkerd.cluster.local issuer.crt issuer.key
```

```
      --profile intermediate-ca --not-after 8760h
      --no-password --insecure --ca root.crt --ca-
      key root.key"
37    register: result_2
38    retries: 2
39    delay: 10
40    until: result_2 is not failed
41
42  - name: Install Linkerd in each cluster
43    shell: "/home/runner/.linkerd2/bin/linkerd
      install --identity-trust-anchors-file root.
      crt --identity-issuer-certificate-file
      issuer.crt --identity-issuer-key-file issuer
      .key --context='{{ item }}' | kubectl --
      context='{{ item }}' apply -f -"
44    register: result_3
45    retries: 2
46    delay: 60
47    until: result_3 is not failed
48    with_items: "{{ clusters_name.split(',') }}"
49
50  - name: Check clusters
51    shell: "/home/runner/.linkerd2/bin/linkerd
      check --context='{{ item }}'"
52    register: "check_clusters1"
53    with_items: "{{ clusters_name.split(',') }}"
54    retries: 2
55    delay: 60
56    until: check_clusters1 is not failed
57
58  - name: Show Check clusters
59    debug: "msg={{ item.stdout_lines }}"
```

```
60        with_items: "{{ check_clusters1.results }}"
61
62      - name: Install Linkerd viz cluster
63        shell: "/home/runner/.linkerd2/bin/linkerd viz
            install --context='{{ item }}' | kubectl --
            context='{{ item }}' apply -f -"
64        register: result_4
65        retries: 2
66        delay: 60
67        until: result_4 is not failed
68        with_items: "{{ clusters_name.split(',') }}"
69
70      - name: Check clusters viz
71        shell: "/home/runner/.linkerd2/bin/linkerd
            check --context='{{ item }}'"
72        register: "check_clusters"
73        retries: 2
74        delay: 90
75        until: check_clusters is not failed
76        with_items: "{{ clusters_name.split(',') }}"
77
78      - name: Show Check clusters viz
79        debug: "msg={{ item.stdout_lines }}"
80        with_items: "{{ check_clusters.results }}"
81
82      - name: Install Linkerd multicluster
83        shell: "/home/runner/.linkerd2/bin/linkerd
            multicluster install --context='{{ item }}'
            | kubectl --context='{{ item }}' apply -f -"
84        register: result_5
85        retries: 2
86        delay: 60
```

```
87        until: result_5 is not failed
88        with_items: "{{ clusters_name.split(',') }}"
89
90      - name: Check gateway on clusters
91        shell: "kubectl --context='{{ item }}' -n
            linkerd-multicluster rollout status deploy/
            linkerd-gateway"
92        register: "check_gateway"
93        retries: 2
94        delay: 60
95        until: check_gateway is not failed
96        with_items: "{{ clusters_name.split(',') }}"
97
98      - name: Show Check gateways
99        debug: "msg={{ item.stdout_lines }}"
100       with_items: "{{ check_gateway.results }}"
101
102     - name: Link Linkerd clusters
103       shell: |
104         /home/runner/.linkerd2/bin/linkerd
              multicluster link --cluster-name {{
              master_cluster }} --context='{{
              master_cluster }}' --set '
              enableHeadlessServices=true' | kubectl --
              context='{{ item }}' apply -f -
105       /home/runner/.linkerd2/bin/linkerd
            multicluster link --cluster-name {{ item
            }} --context='{{ item }}' --set '
            enableHeadlessServices=true' | kubectl --
            context='{{ master_cluster }}' apply -f -
106       loop: "{{ clusters_name.split(',') }}"
107       when: item != master_cluster
```

```
108         register: result_6
109         retries: 2
110         delay: 60
111         until: result_6 is not failed
112
113       - name: Install nginx on clusters
114         shell: "kubectl --context='{{ item }}' apply -f
                https://raw.githubusercontent.com/
              kubernetes/ingress-nginx/controller-v1.2.1/
              deploy/static/provider/cloud/deploy.yaml"
115         register: "check_nginx"
116         with_items: "{{ clusters_name.split(',') }}"
117         retries: 2
118         delay: 60
119         until: check_nginx is not failed
120
121       - name: Show Check nginx
122         debug: "msg={{ item.stdout_lines }}"
123         with_items: "{{ check_nginx.results }}"
124
125       - name: Expose dashboards
126         shell: "kubectl --context='{{ item }}' apply -f
                /home/runner/work/kubernetes_multicluster/
              kubernetes_multicluster/terraform/ansible/
              deployments/expose_dashboard.yml"
127         with_items: "{{ clusters_name.split(',') }}"
128         register: result_7
129         retries: 2
130         delay: 60
131         until: result_7 is not failed
```

Listing A.5: Código-fonte do módulo Ansible para configuração do *multicluster* Kubernetes.

Código do módulo Ansible para implantação das aplicações 5G no *multicluster* Kubernetes.

```
- hosts: all
  become: yes
  environment:
    KUBECONFIG: "{{ kubeconfig_env }}"

  pre_tasks:

    - name: Create kubeconfig files
      ansible.builtin.copy:
        dest: "{{ index }}.yml"
        content: |
          {{ item | b64decode }}
      loop: "{{ kubeconfigs_b64.split(',') }}"
      loop_control:
        index_var: index
      no_log: true

    - name: Get all kubernetes cluster contexts
      command: |
        kubectl config get-contexts
      register: contexts

    - name: Show all kubernetes cluster contexts
      debug: var=contexts.stdout

  tasks:
    - name: Get namespace files to deploy
      ansible.builtin.find:
        paths: /home/runner/work/
          kubernetes_multicluster/
          kubernetes_multicluster/terraform/ansible/
```

```
            applications
30          file_type: file
31          recurse: yes
32          patterns: 'namespace.yml'
33        register: namespace_files
34
35    - name: Deploy namespaces
36        shell: "kubectl --context='{{ item[1] }}' apply
              -f {{ item[0].path }}"
37        with_nested:
38          - "{{ namespace_files.files }}"
39          - "{{ clusters_name.split(',') }}"
40
41    - name: Get app files to deploy
42        ansible.builtin.find:
43          paths: /home/runner/work/
              kubernetes_multicluster/
              kubernetes_multicluster/terraform/ansible/
              applications
44          file_type: file
45          recurse: yes
46          patterns: '*.yml'
47          exclude: 'namespace.yml'
48        register: app_files
49
50    - name: Deploy apps
51        shell: "kubectl --context='{{ item[1] }}' apply
              -f {{ item[0].path }}"
52        with_nested:
53          - "{{ app_files.files }}"
54          - "{{ clusters_name.split(',') }}"
```

Listing A.6: Código-fonte do módulo Ansible para implantação das aplicações 5G

nos *n clusters* do *multicluster* Kubernetes.

Workflow do GitHub Actions para criação do *multicluster* Kubernetes.

```
name: Create Multicluster

env:
  RESOURCE_GROUP: "tfstate-rg"
  STORAGE_ACCOUNT: "tfstatepfaccount"
  CONTAINER_NAME: "tfstate-container"
  KEY: "main.tfstate"
  APP_SECRET: ${{ secrets.AZURE_APP_SECRET }}
  AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY }}
  AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_KEY
    }}

on:
  push:
    branches: [ main ]
  schedule:
    - cron: '10 * * * *'

  workflow_dispatch:


jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Replace Tokens
        uses: cschleiden/replace-tokens@v1.1
```

```
30          with:
31            files: '["**/*"]'
32
33        - name: Setup Python
34          uses: actions/setup-python@v4.0.0
35          with:
36            # Version range or exact version of Python
                to use, using SemVer's version range
                syntax. Reads from .python-version if
                unset.
37            python-version: '3.x'
38
39        - name: Setup pip3
40          run: |
41            curl https://bootstrap.pypa.io/get-pip.py -
                o get-pip.py
42            python3 get-pip.py --user
43          working-directory: ${{ github.workspace}}/
              terraform
44
45        - name: Install Ansible
46          run: |
47            python3 -m pip install --user ansible
48            ansible --version
49            ansible-galaxy collection install
                kubernetes.core
50          working-directory: ${{ github.workspace}}/
              terraform
51
52        - name: Install Terraform
53          uses: hashicorp/setup-terraform@v1
54          with:
```

```
55            terraform_version: latest
56
57        - name: Install Kubectl
58          uses: Azure/setup-kubectl@v2.1
59          with:
60            version: latest
61
62        - name: Install Linkerd
63          run: |
64            curl --proto '=https' --tlsv1.2 -sSfL
                https://run.linkerd.io/install | sh
65            export PATH=$PATH:/home/runner/.linkerd2/
                bin
66          working-directory: ${{ github.workspace}}/
              terraform
67
68        - name: Install step-cli
69          run: |
70            wget https://dl.step.sm/gh-release/cli/docs
                -cli-install/v0.20.0/step-cli_0.20.0
                _amd64.deb
71            sudo dpkg -i step-cli_0.20.0_amd64.deb
72          working-directory: ${{ github.workspace}}/
              terraform
73
74
75        - name: Terraform Init
76          run: terraform init
77          working-directory: ${{ github.workspace}}/
              terraform
78
79        - name: Terraform Plan
```

```
80          run: terraform plan #-target module.
                clusters_azure -target module.clusters_aws
81          working-directory: ${{ github.workspace}}/
                terraform
82
83        - name: Terraform Apply Full
84          run: time terraform apply --auto-approve
85          working-directory: ${{ github.workspace}}/
                terraform
86
87        - name: Set KUBECONFIG env and run ansible
88          run: |
89            CONFIG_NAMES=$(cat inventory | grep
                  clusters_name)
90            CONFIG_NAMES=${CONFIG_NAMES##*=}
91            CONFIG_NAMES=$(echo "$CONFIG_NAMES" | sed
                  -r 's/"//g')
92            CONFIG_NAMES=$(echo "$CONFIG_NAMES" | sed
                  -r 's/,/.yml:\/home\/runner\/work\/
                  kubernetes_multicluster\/
                  kubernetes_multicluster\/terraform\//g')
93            PWD=$(pwd)
94            CONFIG_NAMES=$(echo "$PWD/$CONFIG_NAMES")
95            export KUBECONFIG="${CONFIG_NAMES}.yml"
96            echo "$KUBECONFIG"
97            time ansible-playbook ./ansible/
                  install_multicluster_playbook.yml -i
                  inventory -e "kubeconfig=$KUBECONFIG"
98          working-directory: ${{ github.workspace}}/
                terraform
```

Listing A.7: Código-fonte do *workflow* do GitHub Actions para criação do *multicluster* Kubernetes.

Workflow do GitHub Actions para implantação das aplicações 5G no *multi-cluster* Kubernetes.

```yaml
name: Deploy Applications

env:
  RESOURCE_GROUP: "tfstate-rg"
  STORAGE_ACCOUNT: "tfstatepfaccount"
  CONTAINER_NAME: "tfstate-container"
  KEY: "main.tfstate"
  APP_SECRET: ${{ secrets.AZURE_APP_SECRET }}
  AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY }}
  AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_KEY
    }}

on:
  push:
    branches: [ main ]
  schedule:
    - cron: '10 * * * *'

  workflow_dispatch:


jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Replace Tokens
        uses: cschleiden/replace-tokens@v1.1
        with:
```

```
31            files: '["**/*"]'
32
33        - name: Setup Python
34          uses: actions/setup-python@v4.0.0
35          with:
36            # Version range or exact version of Python
                to use, using SemVer's version range
                syntax. Reads from .python-version if
                unset.
37            python-version: '3.x'
38
39        - name: Setup pip3
40          run: |
41            curl https://bootstrap.pypa.io/get-pip.py -
                o get-pip.py
42            python3 get-pip.py --user
43          working-directory: ${{ github.workspace}}/
              terraform
44
45        - name: Install Ansible
46          run: |
47            python3 -m pip install --user ansible
48            ansible --version
49            ansible-galaxy collection install
                kubernetes.core
50          working-directory: ${{ github.workspace}}/
              terraform
51
52        - name: Install Terraform
53          uses: hashicorp/setup-terraform@v1
54          with:
55            terraform_version: latest
```

```
56
57        - name: Install Kubectl
58          uses: Azure/setup-kubectl@v2.1
59          with:
60            version: latest
61
62        - name: Terraform Init
63          run: terraform init
64          working-directory: ${{ github.workspace}}/
              terraform
65
66        - name: Terraform Plan
67          run: terraform plan #-target module.
              clusters_azure -target module.clusters_aws
68          working-directory: ${{ github.workspace}}/
              terraform
69
70        - name: Terraform Apply Full
71          run: time terraform apply --auto-approve
72          working-directory: ${{ github.workspace}}/
              terraform
73
74        - name: Set KUBECONFIG env and run ansible
75          run: |
76            CONFIG_NAMES=$(cat inventory | grep
                clusters_name)
77            CONFIG_NAMES=${CONFIG_NAMES##*=}
78            CONFIG_NAMES=$(echo "$CONFIG_NAMES" | sed
                -r 's/"//g')
79            CONFIG_NAMES=$(echo "$CONFIG_NAMES" | sed
                -r 's/,/.yml:\/home\/runner\/work\/
                kubernetes_multicluster\/
```

```
            kubernetes_multicluster\/terraform\//g')
80          PWD=$(pwd)
81          CONFIG_NAMES=$(echo "$PWD/$CONFIG_NAMES")
82          export KUBECONFIG="${CONFIG_NAMES}.yml"
83          echo "$KUBECONFIG"
84          ansible-playbook ./ansible/
              deploy_apps_playbook.yml -i inventory -e
               "kubeconfig=$KUBECONFIG"
85        working-directory: ${{ github.workspace}}/
            terraform
```

Listing A.8: Código-fonte do *workflow* do GitHub Actions para implantação das aplicações 5G no *multicluster* Kubernetes.