INSTITUTO FEDERAL DE PERNAMBUCO
***CAMPUS* BELO JARDIM**
BACHARELADO EM ENGENHARIA DE SOFTWARE

KAIQUE RIERICKSON TORRES SILVA

Implementação e Orquestração Automatizada de Clusteres Kubernetes com GitOps: Um Estudo de Caso.

Belo Jardim, Pernambuco
28/06/2023

KAIQUE RIERICKSON TORRES SILVA

Implementação e Orquestração Automatizada de Clusteres Kubernetes com GitOps: Um Estudo de Caso.

Trabalho de conclusão de Curso (TCC) apresentado como requisito parcial para obtenção do grau de Bacharel em Engenharia de Software. Primeiro TCC aprovado no curso de Engenharia de Software do IFPE - Campus Belo Jardim

Banca de Qualificação:

João Almeida - IFPE - Campus Belo Jardim
Elton Bezerra Torres - IFPE - Campus Belo Jardim
Daniel leite Viana - UFAPE

Belo Jardim, Pernambuco, 28/06/2023.

Dados Internacionais de Catalogação - CIP

S586i Silva, Kaique Rierickson Torres

Implementação e orquestração automatizada de clusteres kubernets com gitops: um estudo de caso / Kaique Rierickson Torres Silva. – Belo Jardim-PE, 2023.

78f.: il. ; 29 cm.

Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Software) – Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco, Campus Belo Jardim- PE, 2023.

Orientador: Prof.º Elton Bezerra Torres.

Inclui referências.

1. Programação de computadores - software. 2. Gerenciamento de aplicações. 3. Desenvolvimento de aplicações. I. Título. II. Torres, Elton Bezerra. III. Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco.

CDD 005

Ficha catalográfica elaborada pelo Sistema Integrado de Bibliotecas - SIBI/IFPE.
Bibliotecária: Louise Machado Freire Dias CRB – 4 /2267.

KAIQUE RIERICKSON TORRES SILVA

IMPLEMENTAÇÃO E ORQUESTRAÇÃO AUTOMATIZADA DE CLUSTERES KUBERNETES COM GITOPS: UM ESTUDO DE CASO.

Trabalho aprovado. Belo Jardim, 28/06/2023.

Elton Bezerra Torres

Professor Orientador

João Almeida e Silva

Convidado 1

Daniel leite Viana

Convidado 2

AGRADECIMENTOS

A elaboração deste trabalho não teria sido possível sem o apoio e a colaboração de diversas pessoas e instituições, as quais gostaria de agradecer e reconhecer.

Primeiramente, gostaria de agradecer ao meu orientador, Elton Torres, pelo seu constante apoio, orientação e feedbacks construtivos que foram fundamentais para a conclusão deste estudo. De uma forma menos formal, obrigado por aguentar todo o aperreio para o mestre corrigir cada versão desse TCC, obrigado também por me apresentar essa área maravilhosa do DevOps, GitOps e todos os Ops aí. Sem dúvidas o senhor mudou minha vida, muito obrigado!!!

Também gostaria de agradecer à minha família, em especial a minha mãe Maria Rita e meu irmão Kaynan, pela paciência, compreensão (precisou de muita mesmo viu?!) e amor incondicional durante toda a minha trajetória acadêmica. Amo vocês!!

Agradeço ainda às pessoas que participaram direta e indiretamente da pesquisa, obrigado por compartilharam suas experiências, bem como às instituições e empresas que disponibilizaram informações e recursos necessários para a realização deste trabalho.

Agradeço a Weverton, colega de curso, por compartilhar conhecimentos, experiências e momentos inesquecíveis ao longo desta jornada de construção do TCC. Amigo, toda ajuda e apoio nessa caminhada foram essências para que eu pudesse chegar aqui, tamo junto sempre!! Quando estiver escrevendo o seu pode me aperrear para lhe ajudar também. Aaaa e os duos no LoLzin foram extremamente necessários para que eu não pirasse!

A todos vocês, meu sincero e profundo agradecimento. Sem o apoio e a colaboração de vocês, esta conquista não seria possível.

Por fim e mais importante que todos, agradeço a Deus por ter me dado forças e paciência para conseguir terminar esse projeto, só Ele sabe o tamanho da luta!!

“O Senhor é a minha força e o meu escudo; nele o meu coração confia, e dele recebo ajuda. Meu coração exulta de alegria, e com o meu cântico lhe darei graças. Salmos 28:7”

“Aleluia! Deem graças ao Senhor porque ele é bom; o seu amor dura para sempre. Salmos 106:1”

RESUMO

Este trabalho de conclusão de curso (TCC) apresenta um estudo de caso sobre a implementação e orquestração automatizada de clusters Kubernetes utilizando a metodologia GitOps. O Kubernetes é uma plataforma de código aberto amplamente utilizada para o gerenciamento de contêineres e escalabilidade de aplicações. No entanto, configurar e gerenciar um cluster Kubernetes pode ser complexo e exigir um esforço significativo (KUBERNETS, s.d.). Neste estudo de caso, propomos a utilização da abordagem GitOps, que se baseia no uso de repositórios Git para controlar e automatizar o ciclo de vida da infraestrutura e das aplicações em um cluster Kubernetes. Essa metodologia permite uma implantação e uma gestão mais eficientes do ambiente, garantindo uma configuração consistente e rastreável. Durante o estudo de caso, implementamos um cluster Kubernetes em um ambiente de nuvem pública e utilizamos o GitOps para automatizar a implantação de aplicações e atualizações no cluster. Exploramos ferramentas como Rancher e Fleet para sincronizar as definições de configuração do Kubernetes armazenadas em um repositório Git com o estado real do cluster. Ao longo do estudo, avaliamos a eficácia da abordagem GitOps em termos de facilidade de implantação, manutenção e escalabilidade do cluster Kubernetes. Também consideramos aspectos como segurança, confiabilidade e monitoramento contínuo do ambiente. Os resultados do estudo de caso mostram que a implementação e orquestração automatizada de clusters Kubernetes com GitOps oferecem benefícios significativos em termos de eficiência operacional, rastreabilidade e confiabilidade do ambiente. A abordagem GitOps simplifica a gestão do cluster, permitindo uma maior agilidade no desenvolvimento e implantação de aplicações. Este estudo de caso serve como um guia prático para profissionais de TI interessados em adotar a metodologia GitOps para o gerenciamento automatizado de clusters Kubernetes. Ele destaca as melhores práticas, desafios enfrentados e insights relevantes para implementar com sucesso uma abordagem de orquestração automatizada baseada em GitOps em um ambiente Kubernetes.

Palavras-chave: IaC, Kubernets, Terraform, Ansible, GitOps, Rancher, DevOps.

ABSTRACT

This thesis presents a case study on the automated implementation and orchestration of Kubernetes clusters using the GitOps methodology. Kubernetes is an open-source platform widely used for container management and application scalability. However, configuring and managing a Kubernetes cluster can be complex and require significant effort (KUBERNETS, s.d.). In this case study, we propose the use of the GitOps approach, which relies on Git repositories to control and automate the lifecycle of infrastructure and applications in a Kubernetes cluster. This methodology enables more efficient deployment and management of the environment, ensuring consistent and traceable configuration. During the case study, we implemented a Kubernetes cluster in a public cloud environment and utilized GitOps to automate the deployment of applications and updates to the cluster. We explored tools such as Rancher and Fleet to synchronize Kubernetes configuration definitions stored in a Git repository with the actual state of the cluster. Throughout the study, we evaluated the effectiveness of the GitOps approach in terms of ease of deployment, maintenance, and scalability of the Kubernetes cluster. We also considered aspects such as security, reliability, and continuous monitoring of the environment. The results of the case study demonstrate that the automated implementation and orchestration of Kubernetes clusters with GitOps offer significant benefits in terms of operational efficiency, traceability, and reliability of the environment. The GitOps approach simplifies cluster management, allowing for greater agility in application development and deployment. This case study serves as a practical guide for IT professionals interested in adopting the GitOps methodology for automated management of Kubernetes clusters. It highlights best practices, challenges faced, and relevant insights for successfully implementing a GitOps-based automated orchestration approach in a Kubernetes environment.

Keywords: IaC, Kubernets, Terraform, Ansible, GitOps, Rancher, DevOps.

LISTA DE ABREVIATURAS

IaC	infraestrutura como código
K8s	Kubernets
CI	Continuous Integration - integração continua
CD	Continuous Deployment - Deploy continuo
TF	Terraform
T.I.	Tecnologia da Informação
OCI	Oracle Cloud Infrastructure
IaaS	Infrastructure as a Service
PaaS	Platform as a Service
SaaS	Software as a Service
AWS	Amazon Web Services, Inc.
GCP	Google Cloud Platform
GitOps	Git and Operations
DevOps	Developer and Opartions
VM	Virtual Machine
VCN	Virtual Cloud Network
S.O.	Sistema operacional
AD	Availability Domains
OCPU	Oracle CPU

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1

INTRODUÇÃO

Um dos passos no desenvolvimento de aplicações é a implantação do sistema, nessa etapa que o sistema desenvolvido será disponibilizado para o usuário final, e nesse processo são necessárias a aplicação de diversas ferramentas, técnicas e tecnologias para tornar a aplicação ou sistema acessível aos usuários.

Para realizar a implantação e disponibilização do sistema é preciso criar uma infraestrutura que hóspede o sistema desenvolvido. Por muito tempo a forma de implementar os sistemas era feita em ambientes on-premise, o termo on-premise caracteriza ambientes que são construídos localmente no ambiente de T.I, geralmente esses ambientes são data center que tem todos os serviços utilizados instalados localmente (CRONAPPS, 2022). Para maximizar a produtividade e disponibilidade dos sistemas, principalmente os sistemas web, cada vez mais a implantação é feita em infraestruturas na nuvem, caracterizada por: integração das informações, redução de custos com infraestrutura, escalabilidade ao utilizar os serviços, etc. Estimativas apontam que 65 por cento dos aplicativos estarão prontos ou otimizados para serem disponibilizados na nuvem até 2027. (GARTNER, 2023a).

Com essa migração de sistemas para a nuvem, houve uma tendência de mudança na forma de deploy, que era realizado em servidores físicos ou virtuais e com essa alteração passaram a utilizar cada vez mais tecnologias baseadas em contêiner, os quais “são ambientes isolados utilizados para empacotar aplicações. Contêineres têm o objetivo de segregar e facilitar a portabilidade de aplicações em diferentes ambientes.”(GUEDES, 2021) Com o crescimento da utilização de contêineres na infraestrutura e no desenvolvimento de sistemas, surgiu a necessidade de orquestração dos recursos que serão utilizados.

Para realizar essa orquestração uma tendência é a utilização do Kubernets, que consiste em um sistema de orquestração de contêineres com a função de automatizar o dimensionamento dos contêineres, a implantação e a gestão das aplicações que estarão rodando nos contêineres (KUBERNETS, s.d.). No Kubernets temos os clus-

ters, esses clusters são um conjunto de servidores de processamento, neles teremos os nós, que são máquinas que realizam o processamento no cluster Kubernets (KUBERNETS, 2022). Para que os clusters funcionem de forma correta deve existir pelo menos, um plano de controle responsável por manter o estado desejado do cluster e pelo menos uma máquina responsável pela execução das aplicações em contêineres.(REDHAT, 2020b)

Ainda durante o processo de criação da infraestrutura de deployment, a criação dos recursos de forma manual pode ser um ponto de gargalo nesse fluxo (JULIAKM, s.d.). Dessa forma, para evitar que aconteçam divergências na criação dos recursos, já que a criação manual é suscetível a erros, quanto para agilizar todo o processo de criação, torna-se comum a utilização da infraestrutura como código (IaC), consistindo basicamente em transformar a criação e gerenciamento dos recursos da infraestrutura em código(REDHAT, 2022b), utilizando para isso algumas tecnologias como: Terraform e Ansible.

A fim de que todo o processo de criação e administração da infraestrutura seja automático e fluida, sempre que houver uma atualização nessa IaC, ela precisa ser replicada de forma automática no servidor que estivermos utilizando, para executar essa função o GitOps se mostra como a melhor pratica. conforme a Waveworks GitOps é “Um modelo operacional para infraestrutura como código, que fornece um conjunto de práticas recomendadas e unificam a implantação, o gerenciamento e o monitoramento de aplicativos em contêineres.Um caminho para uma experiência de desenvolvimento para gerenciar aplicativos; onde *pipelines* CI/CD ponta a ponta e fluxos de trabalho Git são aplicados a operações e desenvolvimento.” (WAVEWORKS, 2018)

Por fim, com a necessidade de uma agilidade maior na entrega desses sistemas aos usuários finais, os times e departamentos de infraestrutura vem utilizando com maior frequência as práticas DevOps, que consistem em: “Um composto de Dev (desenvolvimento) e Ops (operações), o DevOps é a união de pessoas, processos e tecnologias para fornecer continuamente valor aos clientes.”(AZURE, 2022). Atualmente a filosofia DevOps vem sendo bastante utilizada porque visa uma integração e entrega continua, agilizando o desenvolvimento e a disponibilização dos sistemas para o usuário final.

1.1 OBJETIVOS

O objetivo geral desse trabalho é a criação e disponibilização de uma infraestrutura de deployment de aplicações web na nuvem, baseada em Kubernets e construída com algumas ferramentas de IaC, com foco em aproximar o engenheiro de softwares as tecnologias utilizadas.
Tendo em vista os benéficos da implantação de infraestrutura ágil com a utilização do DevOps, os objetivo mais específicos do trabalho são:
— Criar a infraestrutura de um cluster em uma nuvem publica utilizando IaC;
— Introduzir desenvolvedores a utilização de práticas DevOps no *deployment* dos seus sistemas webs;
— Mostrar a importância do passo de implantação no desenvolvimento de softwares web;
— Instalar e configurar uma ferramenta de gerenciamento de clusters, com foco em facilitar o gerenciamento desses *clusters* pelos desenvolvedores;
— Criar uma infraestrutura simples de *pipeline* de CI/CD integrada a infraestrutura criada.
— Aplicar conceitos de GitOps para versionamento, controle e provisionamento da infraestrutura criada.

2

FUNDAMENTAÇÃO TEÓRICA

Para uma melhor compreensão da implementação e da metodologia, é necessário explicar alguns termos com um maior nível de detalhes, termos esses que estão regendo o direcionamento do trabalho como técnicas e/ou métodos utilizados e ferramentas aplicadas no desenvolvimento.

2.1

COMPUTAÇÃO EM NUVEM

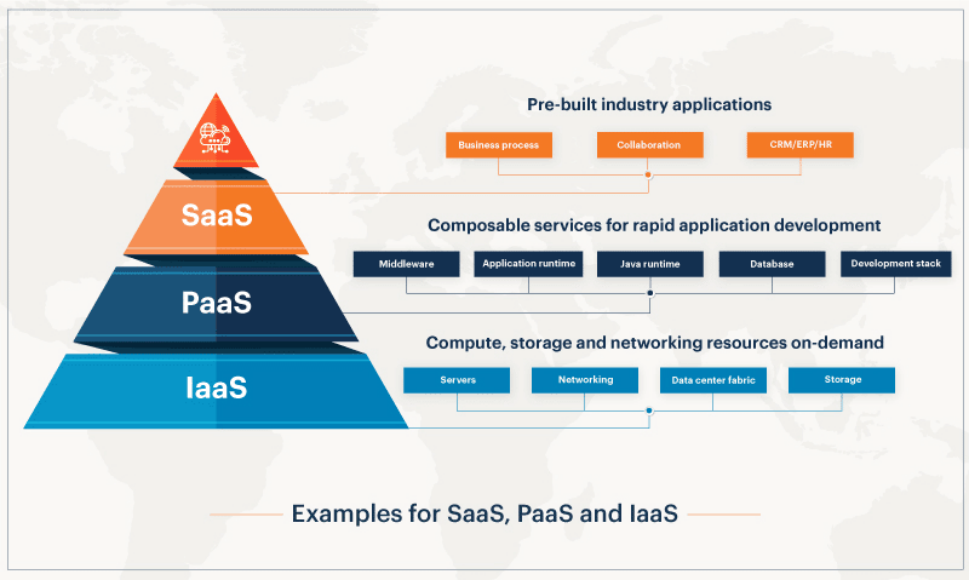
A computação em nuvem é um modelo de computação que permite o acesso a recursos de computação, como servidores, armazenamento e bancos de dados, através da internet. Em outras palavras, a computação em nuvem possibilita que empresas e usuários finais possam ter acesso a recursos computacionais sob demanda, sem a necessidade de investimentos em infraestrutura física, tais como servidores e sistemas de armazenamento, por exemplo.

Segundo Knorr (KNORR; GRUMAN, 2008) “A computação em nuvem entra em foco apenas quando você pensa no que a TI sempre precisa: uma maneira para aumentar a capacidade ou adicionar recursos em tempo real sem investir em nova infraestrutura e treinamento de novos funcionários ou licenciamento de novos softwares”, contudo atualmente entendesse que a computação em nuvem não se limita a apenas esses 3 pontos, ela garante parte dos serviços que uma infraestrutura on-premise oferta a computação em nuvem oferece com algumas melhorias, sendo a mais notável a sua escalabilidade, a escalabilidade na nuvem refere-se à capacidade de aumentar ou diminuir a capacidade de recursos computacionais disponíveis em um ambiente de nuvem, de forma rápida e flexível, para atender às necessidades de demanda em constante mudança, porém não se limitando a isso ela fornece um custo de manutenção menor, ganho de desempenho e disponibilidade de recurso e até um ganho de mobilidade. (RITTINGHOUSE; RANSOME, 2017)

Para que todos esses benefícios cheguem ao cliente, eles precisam utilizar al-

guns serviços, fundamentalmente são 3 serviços: IaaS, PaaS, SaaS (TSAI; HUANG; SHAO, 2011). Algumas bibliotecas consideram DaaS e BaaS como serviços fundamentais(TAURION, 2009), na figura 2.1, segue uma explicação dos modelos utilizados:

Figura 2.1: Saas vs Paas vs Iaas



(a) https://kinsta.com/wp-content/uploads/2022/06/1.-SaaS-vs-IaaS-vs-PaaS.png

- SaaS — Software as a Service, ou *software* como serviço em português. É um modelo de distribuição de *software* no qual o provedor de serviços disponibiliza um aplicativo pela internet e o usuário paga apenas pelo uso, geralmente com base em uma assinatura mensal. Nesse modelo, o provedor de serviços é responsável pelo gerenciamento do *software*, incluindo manutenção, atualizações e segurança. O usuário acessa o *software* por meio de um navegador web ou de um aplicativo dedicado, sem precisar se preocupar com a infraestrutura necessária para hospedar o *software* ou com questões de instalação e configuração (DUBEY; WAGLE, 2007).
- PaaS — um modelo de computação em nuvem que oferece aos usuários uma plataforma completa para desenvolvimento, execução e gerenciamento de aplicativos sem que eles precisem se preocupar com a infraestrutura subjacente. Em outras palavras, o provedor de PaaS oferece aos desenvolvedores uma plataforma pronta para uso, incluindo *hardware*, sistema operacional, *middleware*, banco de dados e outras ferramentas necessárias para criar, implantar e executar aplicativos (TOTVS, 2022).
- IaaS — É um modelo de computação em nuvem que fornece acesso a recursos de computação virtualizados, como servidores, armazenamento e redes, por

meio da internet. Com o IaaS, as empresas podem provisionar, gerenciar e escalar recursos de TI sob demanda, sem precisar investir em infraestrutura física local ou *hardware* (BHARDWAJ; JAIN; JAIN, 2010). Os provedores de IaaS geralmente oferecem aos clientes uma variedade de opções de configuração de infraestrutura, como tamanho e tipo de servidor, quantidade de armazenamento e largura de banda de rede.

É através dessas categorias de serviços que as *cloud*s disponibilizam aos seus clientes um concorrente as infraestruturas on-premise, com essa divisão de categoria fica mais evidente algumas vantagens da nuvem, um cliente não precisa investir em um *hardware* para montar todo um servidor que tenha a função de hospedar apenas um web app e pode somente contratar aquele serviço de plataforma que atende a sua necessidade. Apesar de todas essas vantagens ainda assim podem surgir dúvidas por parte dos clientes: “como eu vou manter um dado sensível privado?”, “como faço para ter um recurso onde ninguém mais possa utilizar?” ou até “Preciso de um serviço onde eu tenha prioridade máxima no acesso?”. Nem todas as nuvens são iguais, exatamente pensando em diversas demandas distintas dos clientes que existem 3 tipos de nuvem (DOCUSIGN, 2019).

2.1.1

NUVEM PRIVADA

A nuvem privada é um ambiente de computação em nuvem exclusivamente usado por uma única organização, geralmente localizada dentro do data center da própria organização ou em um provedor de serviços de hospedagem dedicado. Segundo a NIST (National Institute of Standards and Technology), uma nuvem privada é definida como ”uma infraestrutura de nuvem provisionada para uso exclusivo de uma única organização que compreende vários consumidores (por exemplo, departamentos comerciais e de TI)”(MELL; GRANCE, 2011).

De acordo com um estudo realizado pela Forrester Research, a adoção de nuvens privadas está crescendo rapidamente, especialmente em grandes empresas, que muitas vezes possuem requisitos de segurança mais rigorosos. A pesquisa também sugere que, embora as nuvens públicas sejam mais populares em termos de adoção, as nuvens privadas são mais populares em termos de uso real e gastos com TI (STATEN, 2011). Outra vantagem da nuvem privada é a maior flexibilidade e controle que ela oferece em relação às nuvens públicas.

Como observado por uma pesquisa da Gartner, “a nuvem privada pode fornecer

opções adicionais de personalização e controle, bem como maior segurança e conformidade" (BITTMAN, 2010). No entanto, é importante notar que a implantação de uma nuvem privada pode ser mais cara do que a implantação de uma nuvem pública devido ao investimento inicial necessário para construir e manter a infraestrutura. Existem várias ferramentas populares para a criação e gerenciamento de nuvens privadas. Aqui estão algumas delas: OpenStack, VMware vSphere, Microsoft Azure Stack, Red Hat OpenShift, Eucalyptus, CloudStack, etc.

2.1.2

NUVEM HÍBRIDA

A nuvem híbrida é uma abordagem em que uma organização usa uma combinação de recursos de nuvem pública e privada para atender às suas necessidades de negócios. Segundo a Gartner, "a computação em nuvem híbrida se tornou uma estratégia importante para empresas que buscam a flexibilidade de escolher a melhor opção de hospedagem para cada aplicativo ou serviço, sem perder o controle ou a segurança dos dados" (BITTMAN, 2010).

Segundo o Instituto Nacional de Padrões e Tecnologia dos EUA, a nuvem híbrida é definida como "uma composição de dois ou mais tipos de nuvem (nuvem privada, nuvem comunitária ou nuvem pública) que permanecem entidades únicas, mas são interconectadas por tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicativos" (MELL; GRANCE, 2011).

A Forrester Research, por sua vez, destaca que a nuvem híbrida oferece "a flexibilidade de escolher a melhor opção de hospedagem para cada aplicativo ou serviço, sem perder o controle ou a segurança dos dados" (STATEN, 2011).

2.1.3

NUVEM PÚBLICA

Nuvem pública é um modelo de computação em nuvem que oferece serviços de computação, armazenamento e rede através da internet para uso público geral. Conforme a definição do NIST (National Institute of Standards and Technology), "a nuvem pública é propriedade de uma organização que disponibiliza recursos de computação, armazenamento e rede para o público geral através da internet, usando tecnologias padronizadas de computação em nuvem" (MELL; GRANCE, 2011).

A nuvem pública é um modelo de entrega de serviços de computação em nuvem altamente escalável e flexível, permitindo que os usuários acessem facilmente uma ampla gama de recursos de computação sem precisar investir em *hardware* e *software* próprios. Além disso, a nuvem pública oferece uma maneira eficiente de lidar com picos de demanda, pois os usuários podem simplesmente solicitar mais recursos quando necessário e liberá-los quando não precisam mais.

No entanto, o uso desse modelo também apresenta alguns desafios, especialmente em relação à segurança e privacidade dos dados. Uma pesquisa da IBM(International Business Machines Corporation) mostra que a segurança é a maior preocupação dos usuários da nuvem pública, com 77 por cento dos entrevistados citando a segurança como um obstáculo significativo para a adoção da nuvem pública (IBM, 2012).

Apesar desses desafios, a nuvem pública continua a ser uma opção atraente para muitas empresas que buscam reduzir seus custos de infraestrutura e aumentar a flexibilidade e agilidade em seus negócios. Segundo Gartner[1], os gastos mundiais de usuários finais com serviços de Nuvem Pública crescerão 21,7 por cento em 2023, totalizando 591,8 bilhões de dólares. O número representa crescimento de 18,8 por cento em relação ao volume do ano de 2022, que foi de aproximadamente 490,3 bilhões de dólares (GARTNER, 2023b). A computação em nuvem revolucionou como as empresas provisionam e utilizam recursos de TI. Os provedores de nuvem pública, como a AWS(Amazon Web Services), a Microsoft Azure e o Google Cloud, têm sido fundamentais nessa transformação. Esses provedores oferecem infraestruturas escaláveis e serviços gerenciados para atender às demandas em constante evolução das empresas. A AWS é líder de mercado, fornecendo uma ampla gama de serviços, desde infraestrutura como serviço (IaaS) até plataformas específicas para desenvolvimento de aplicativos. A Microsoft Azure, por sua vez, destaca-se por sua abordagem híbrida, permitindo a integração de infraestruturas locais com a nuvem. O GCP(Google Cloud Platform), oferecido pelo Google, é conhecido por sua experiência em tecnologias emergentes, como aprendizado de máquina e inteligência artificial.

2.2
DEVOPS

A palavra DevOps se da pela junção de duas funções a *“Developer”* que é o desenvolvedor de *software* e *“Operations”* que geralmente se dá ao time de infraestrutura da empresa, apesar de englobar as responsabilidades de cada uma dessas funções,

[1]A Gartner é uma das principais empresas mundiais especializadas pesquisa e consultoria em tecnologia da informação.

o seu significado vai além de apenas uma junção de nomes, a implantação do DevOps em qualquer ecossistema de T.I se trata da mudança de cultura, a adoção de um pensamento de automação e designs de sistemas que trazem um ganho na agilidade e qualidade da entrega dos produtos que estão sendo desenvolvidos. DevOps pode ser categorizado como uma metodologia que traz técnicas para que a ideia saia do processo de desenvolvimento para o processo de implantação o mais rápido possível, assim gerando um valor maior para o cliente (REDHAT, 2022a).

Com a utilização da metodologia DevOps, ocorre uma junção entre as equipes de Desenvolvimento e de Operações (ou infraestrutura), existem alguns modelos de como DevOps é dividido, no mais comum essas duas funções são mescladas em apenas uma, os responsáveis agora participam desde o planejamento do sistema até a hora em que o sistema tem que ser hospedado em algum serviço, sendo responsáveis assim por cada etapa em todo o ciclo do desenvolvimento. Atualmente tem crescido bastante o termo DevSecOps, o qual é a adição das responsabilidades do time de segurança a todo esses processos e técnicas, a partir disso, uma maior visão analítica sobre assuntos relacionados a segurança do sistema passa a existir na equipe de DevOps ou DevSecOps.
(AMAZON, s.d.[b])

A Amazon define a implantação do DevOps em

> "Essas equipes usam práticas para automatizar processos que historicamente sempre foram manuais e lentos. Eles usam uma pilha de tecnologia e ferramentas que os ajudam a operar e desenvolver aplicativos de modo rápido e confiável. Essas ferramentas também ajudam os engenheiros a realizar tarefas independentemente (por exemplo, implantação de código ou provisionamento de infraestrutura) que normalmente exigiriam a ajuda de outras equipes, e isso aumenta ainda mais a velocidade da equipe.".

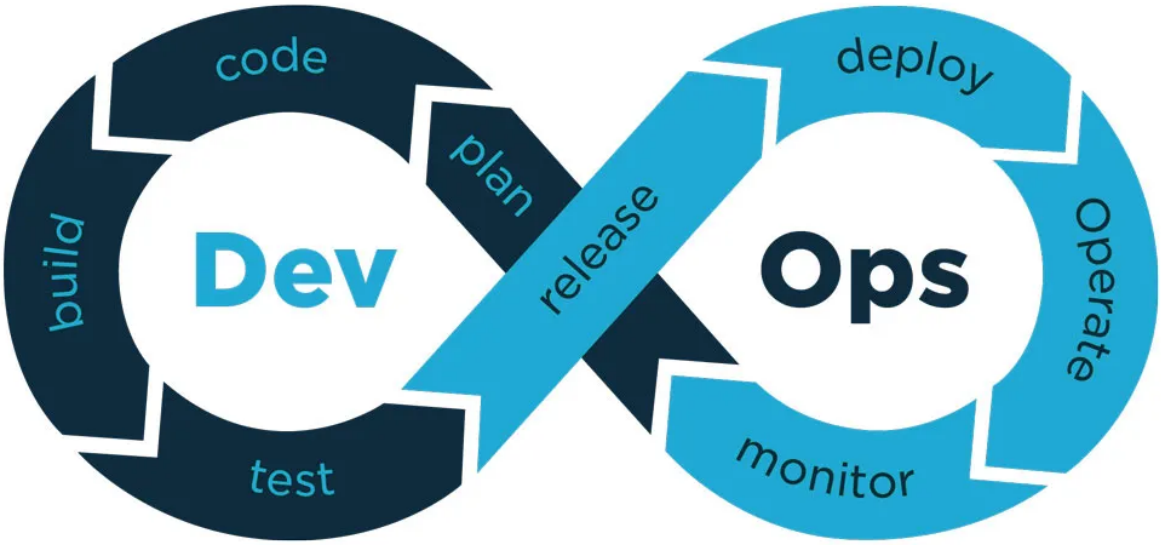
Além disso, listam alguns benéficos na implantação da metodologia: entrega rápida — Para poder entregar inovação aos clientes de forma ágil; confiabilidade — Garantindo a integridade do sistema, evitando falhas; escala — Possibilitando a automação da escalabilidade do sistema. (AMAZON, s.d.[b])

2.2.1

FLUXO DE TRABALHO CONTÍNUO DEVOPS

Para que a metodologia possa ser implantada com sucesso existe um fluxo de trabalho a ser seguido, esse fluxo traz uma simplificação de todos os processos necessários na DevOps, esses fluxos podem ser chamados de estratégias e elas devem ser executadas de forma contínua nos projetos adotados.

Figura 2.2: Fluxo de trabalho contínuo DevOps



(a) Fonte: Jornada para Nuvem

Cada uma dessas fases apresentadas possuem ferramentas e/ou recursos específicos que priorizam a otimização dos resultados de cada etapa e auxiliam na implementação da estratégia. As quatro primeiras fases dizem respeito ao Dev, a parte da codificação em si:

- **Plan** — Para fase de planejamento, uma ferramenta de gerenciamento de projetos é a melhor indicação, como Jira e Trello. (TEAM, 2021)
- **Code** — Na fase de código ferramentas de versionamento de código como o git são essenciais (BERNARDO, 2022).
- **Build** — Para realizar a *build* dos códigos automatizadamente, o Meaven possibilita isso em Java (DEVMEDIA, s.d.), já no Node temos o NPM, para alguns frameworks JavaScript possuem seu próprio *bukd* como o NextJS.
- **Test** — Na fase de testes são utilizadas ferramentas para realizar testes automatizados. Esses teste podem envolver testes unitários, teste de integração, etc. Alguns exemplos de ferramentas utilizadas são JUnit para Java, JavaScript e TypeScript, para .NET temos o XUnit como exemplo

Já as outras quatro fases da estratégia estão relacionadas com o Ops e são tarefas diretamente ligada com a entrega do sistema para o usurário final e sua monitoração.

- **Release** — O release do *software* é basicamente os versionamento do sistema desenvolvido, as ferramentas mais utilizadas dependem muito da arquitetura do

sistema, os contêineres registry podem ser utilizados em sistemas que utilizam contêiner, Python Package Index em sistemas python e vários outros em diversas linguagens. (GAEA, 2019)

- **Deploy** — *Deploy* ou implantação, é a parte antes da publicação do *software* em si, é a montagem do sistema no ambiente de implantação. Ferramentas como o Chef, Puppet, Azure DevOps, Git Actions e mais algumas ferramentas são comumente utilizadas nessa etapa.
- **Operate** — Essa etapa está relacionada ao ambiente que o sistema será implementado, atualmente esses ambientes são em nuvens como Azure, AWS, Google Cloud, Oracle Cloud Infrastructure (OCI). Porém, pode implementado em um ambiente on-premise.
- **Monitor** — A fase do monitoramento começa logo após o sistema ir ao ar, para monitorar os sistemas geralmente usam-se ferramentas como Zabbix, Grafana, Prometheus e alguns outros (GAEA, 2019).

Essas e algumas outras ferramentas podem ser utilizadas para aumentar a qualidade das entregas e agilizar todo o processo dessa estratégia (GAEA, 2019).

2.2.2

PILARES DA FILOSOFIA DEVOPS

Não necessariamente exista uma estratégia correta para a implantação do DevOps em um ambiente de T.I, cada ambiente tem suas características que devem ser respeitadas, então as estratégias devem se adaptar a cada caso, porém o DevOps possuem quatro pilares que também devem ser mantidos (CODEBLOG, 2022).

Comunicação

A comunicação constante entre os membros do time de operações e no time de desenvolvimento é essencial, e no DevOps, junção dos dois mundos, a necessidade de comunicação constante torna-se ainda mais importante, pois os profissionais atuam como uma ponte entre o desenvolvimento e a equipe de operações. Sua função é identificar bugs e implementar soluções por meio do DevOps. Em outras palavras, a comunicação no DevOps foca na criação de processos que facilitem o relacionamento entre as equipes e a integração entre as pessoas, facilitando o conhecimento mútuo das necessidades de cada domínio e a busca de soluções.

“Em outras palavras, a partir do reconhecimento mútuo, a comunicação na DevOps atua na criação de processos que auxiliam na relação entre os times e na integração entre as pessoas.”(CODEBLOG, 2022).

Colaboração

Em uma ordem lógica, a colaboração segue a comunicação. Ou seja, além de compreender os processos e condicionantes da interação entre equipes, a superação de conflitos também é essencial para uma cooperação efetiva.

Essencialmente, isso vai além de simplesmente reconhecer problemas e dificuldades, é fundamental discutir soluções e, principalmente, testá-las para implementá-las. Agora, é particularmente importante manter contato próximo. Para isso, as organizações podem escolher o canal mais conveniente, como chat, aplicativos de mensagens instantâneas ou reuniões de alinhamento. A chave é superar as adversidades e fazer com que ambas as equipes se ajudem para navegar com sucesso na situação (CODEBLOG, 2022).

Automação

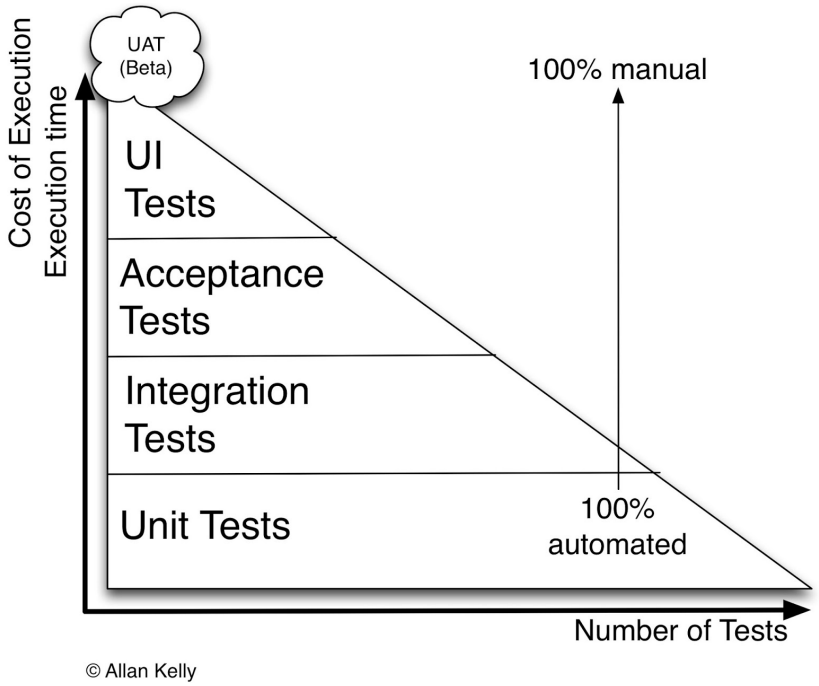
A automação desempenha um papel fundamental em uma transformação DevOps bem-sucedida. É parte fundamental para impulsionar esse movimento, permitindo a integração de ambientes estáveis e o desenvolvimento de processos consistentes para criação, teste e releases bem-sucedidos.

Por meio da automação, a equipe de DevOps pode tornar o processo de lançamento mais previsível e preencher a lacuna entre o design e a produção. As equipes de desenvolvimento se beneficiam do conhecimento operacional, trabalhando juntas para definir o estado ideal da infraestrutura e traduzi-la em código (GAEA, 2019).

- Integração contínua (Continuous Integration - CI): A integração contínua é uma das práticas primarias de desenvolvimento de *software* em projetos que seguem as melhores praticas do DevOps, ela consiste na etapa em que os desenvolvedores centralizam as alterações de código em um repositório. Isso permite que domínios de desenvolvimento, operações e suporte compartilhem informações, conhecimento e apliquem testes automatizados antes de implantar aplicativos (REHKOPF, s.d.).

Como citado anteriormente, uma das principais partes do CI é a automação de testes. Sempre que houver uma alteração no repositório do código do produto, os testes devem ser executados, para isso se faz necessário a utilização de alguma ferramenta que faça a observação do repositório para disparar uma ação assim que essas atualizações chegarem. uma dúvida comum é a escolha de quais testes devem ser automatizados na figura 2.3 Allan Kelly traz a resposta em uma melhoria na pirâmide de teste do Mike Cohn (COHN, 2009):

Figura 2.3: Pirâmide de testes



(a) Fonte: https://www.allankelly.net/archives/628/testing-triangles-pyramids-and-circles/comments

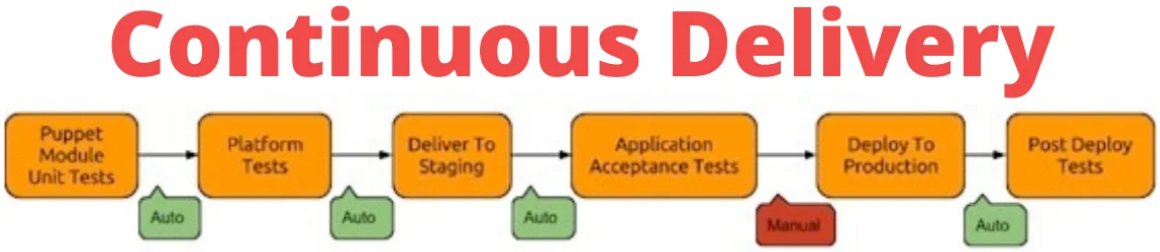
Na base da piramide os testes unitários são colocados como testes que devem ser cem por cento automatizados, segundo Kelly existem de duas a quatro vezes mais testes unitários do que testes de produção em si (KELLY, 2013), por isso ficaria inviável esses testes serem executados manualmente, se perderia muito tempo e isso iria diretamente contra as ideias fundamentais do DevOps e do desenvolvimento ágil.

- Entrega contínua (Continuous Delivery - CD): A entrega contínua refere-se à capacidade de implementar com segurança, rapidez e sustentabilidade qualquer tipo de mudança (como novos recursos, alterações de configuração, correções de *bugs* e experimentos), seja na produção ou nas mãos dos usuários.

O objetivo é tornar a implantação um processo previsível e rotineiro, independentemente do tamanho e da complexidade do sistema — seja um grande sistema distribuído, um ambiente de produção complexo, um sistema integrado ou um aplicativo. Isso é obtido garantindo que o código esteja sempre pronto para ser implantado, mesmo com inúmeros desenvolvedores fazendo alterações diárias. Com o uso do CD, elimina-se a necessidade e as restrições de código das fases de integração, teste e estabilização que tradicionalmente ocorrem após a conclusão do desenvolvimento. (HUMBLE JEZ - FARLEY, 2010)

A figura 2.4 traz a representação visual de processos do Continuous Delivery:

Figura 2.4: Continuous Delivery representação



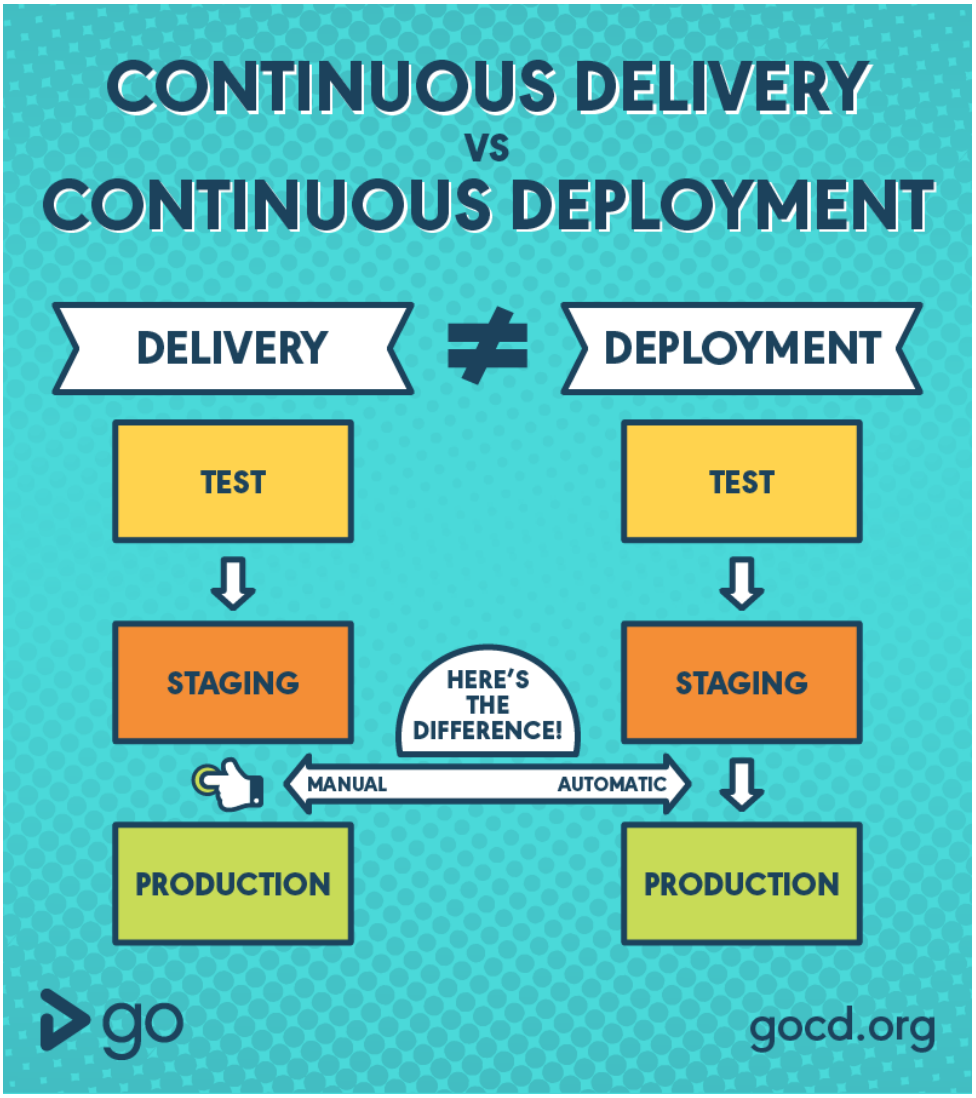
(a) Fonte: https://www.novelvista.com/resources/images/blogs/details/what-is-continuous-deployment-delivery.webp

- Implantação contínua (Continuous Deployment - CD): A implantação continua é como uma continuação da integração continua, ela visa acelerar o *lead time*, diminuindo assim o tempo em que uma alteração no código leva para chegar ao cliente ou usuário final(GAEA, 2022).

  Com a implantação contínua, sempre que uma nova alteração é enviada ao repositório central, um processo é automaticamente iniciado para disponibilizar as mudanças no ambiente desejado. Isso permite que as equipes envolvidas deem *feedbacks* dos processos relacionados ao ciclo de vida do *software* ou serviço.

  Existe uma grande semelhança com o processo do *Continuous Delivery*, eles são tão semelhantes que podem levar até pessoas com certa experiência na área a confundir os processos na hora de realizar uma citação. (GAEA, 2021). Porém, não é complicado de discernir os processos, o Continuous Deployment é totalmente automatizado, já o *Continuous Delivery* possui uma etapa de aprovação que necessita de um trabalho manual, realizado por um integrante do time como podemos observar na imagem 2.5:

Figura 2.5: Diferença entre Deployment e Delivery



(a) Fonte: https://www.gocd.org/assets/images/blog/continous-delivery-vs-deployment-infographic/continuous-delivery-vs-continuous-deployment-infographic-305dd620.png

- DevOps Testing: Esse termo busca o DevOps com o trabalho de controle de qualidade, com o objetivo estabelecer ambientes de testes contínuos que façam parte do processo de desenvolvimento (CODEBLOG, 2022).

Monitoramento

Por último, mas não menos importante, o pilar de monitoramento cobre o rastreamento e a análise de todas as fases do DevOps para determinar o que está funcionando e o que precisa de otimização.

Para isso, diversas ferramentas que permitem a geração automática de relatórios podem ser testadas e utilizadas. Antes, porém, é preciso estabelecer indicadores, de-

finir critérios de controle e, principalmente, monitorar todas as etapas. Afinal, somente coletando e analisando dados é possível obter visões sobre processos e implementar melhorias.(CODEBLOG, 2022)

2.3 GITOPS

GitOps é uma metodologia de gerenciamento de infraestrutura que utiliza práticas de versionamento de código, como Git[1], revisão de código e CI/CD, para automatizar processos e garantir a estabilidade e confiabilidade da infraestrutura de TI. Em vez de tratar a infraestrutura como algo estático, o GitOps permite que ela seja dinâmica e evolua ao longo do tempo, respondendo às necessidades em constante mudança dos usuários e do negócio.(GITLAB, s.d.)

O GitOps é uma extensão da prática de infraestrutura como um código(infrastructure as a code - IaC), sendo a abordagem de gerenciamento de infraestrutura que trata a infraestrutura como código e a controla usando ferramentas de controle de versão, como Git (REDHAT, 2023). No GitOps, todas as configurações da infraestrutura são armazenadas como código e qualquer mudança é gerenciada por meio de um processo de solicitação de *pull request (solicitação de pull)* e revisão de código. Isso ajuda a garantir que todas as alterações sejam verificadas antes de serem implementadas, reduzindo o risco de falhas e problemas de segurança.

Com o GitOps, o processo de implantação é automatizado usando *pipelines* de CI/CD. Isso significa que todas as alterações de infraestrutura são validadas automaticamente em um ambiente de teste antes de serem implantadas em produção. Isso ajuda a reduzir os erros humanos e melhora a confiabilidade do sistema (HARRER, s.d.). Outra vantagem do GitOps é que ele permite que as equipes de operações de TI e desenvolvimento trabalhem juntas de maneira mais eficaz. Como as configurações da infraestrutura são tratadas como código, a equipe de desenvolvimento pode dar *feedback* sobre as alterações propostas para a infraestrutura, permitindo que a equipe de operações de TI responda rapidamente às necessidades dos usuários finais. O GitOps tem se tornado cada vez mais popular, especialmente entre as equipes de DevOps, por permitir que as organizações implementem mudanças de infraestrutura de maneira mais rápida e eficaz. Algumas das empresas que estão usando o GitOps incluem o Google, a Netflix e a Amazon Web Services (STORZ, 2022).

[1] Git é um sistema de controle de versões distribuído, usado principalmente no desenvolvimento de *software*

2.3.1

PRINCÍPIOS DO GITOPS

Existem alguns princípios fundamentais associados ao GitOps. Aqui estão eles:

- Infraestrutura como código: O GitOps, juntamente com ferramentas como o Terraform, promove a ideia de que toda a infraestrutura necessária para executar um aplicativo deve ser especificada e gerenciada como código. Isso significa que todas as configurações de infraestrutura, como ambientes, *clusters*, recursos de nuvem e políticas de segurança, são versionadas no Git. O Terraform, em particular, é uma ferramenta amplamente utilizada para provisionar e gerenciar infraestrutura como código, permitindo que os desenvolvedores descrevam sua infraestrutura desejada de forma declarativa e automatizada. Com o Terraform, é possível criar, modificar e destruir recursos de infraestrutura de forma previsível e controlada, garantindo a consistência e a reprodutibilidade do ambiente de implantação.

- Declaração do estado desejado: O estado desejado do sistema é declarado por meio de arquivos de configuração. Esses arquivos especificam a infraestrutura, os recursos e as dependências necessárias para implantar um aplicativo em um ambiente específico.

- Automação e entrega contínua: O GitOps enfatiza a automação e a entrega contínua para simplificar o processo de implantação de aplicativos. As alterações no código e nas configurações são verificadas no Git e, em seguida, implantadas automaticamente nos ambientes apropriados.

- Reconciliação do estado: O GitOps adota o conceito de reconciliação contínua para garantir que o estado atual do sistema corresponda ao estado declarado no Git. Uma ferramenta de GitOps monitora continuamente o repositório do Git e aplica as alterações necessárias para garantir que o estado atual seja sempre igual ao estado desejado.

- Rastreabilidade e auditoria: Como todas as alterações são versionadas no Git, o GitOps oferece rastreabilidade e auditoria completas. É possível visualizar o histórico de alterações, rastrear quem fez cada alteração e entender o impacto das mudanças no sistema.

Esses são alguns dos princípios do GitOps. Segui-los pode ajudar a promover uma abordagem mais eficiente e confiável para o gerenciamento de infraestrutura e implantação de aplicativos (REDHAT, 2023).

2.3.2

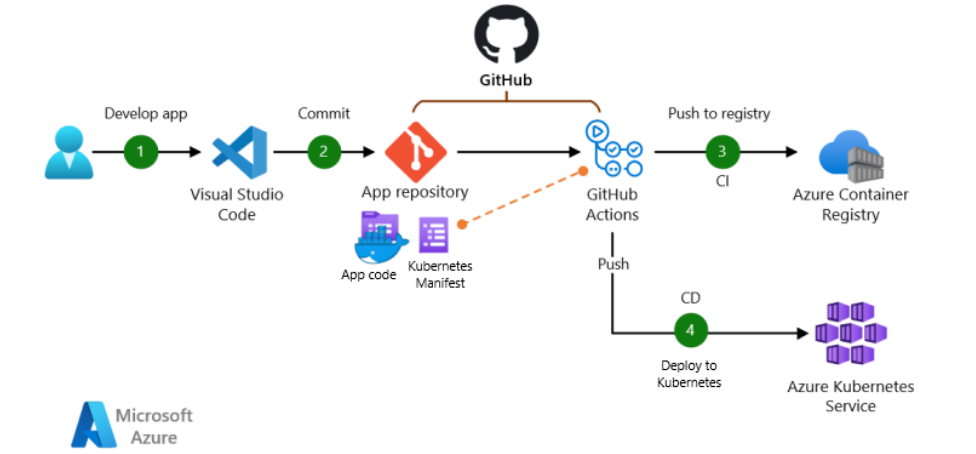
ABORDAGENS GITOPS

O GitOps é a prática de controlar as definições do ambiente e automatizar a execução de mudanças usando o Git. Existem duas abordagens principais do GitOps: Push e Pull. O Push GitOps envolve o uso de um pipeline CI/CD para enviar mudanças para o ambiente, enquanto o Pull GitOps envolve um agente que verifica regularmente o repositório Git e/ou o registro de contêineres em busca de mudanças e aplica a configuração definida no ambiente quando há uma diferença detectada. O Push GitOps é mais acessível, flexível e eficiente, enquanto o Pull GitOps é mais adequado para cargas de trabalho nativas da nuvem e oferece riscos reduzidos de segurança e conformidade (CHIA, 2021).

Segue uma representação visual de ambos os pipelines com ferramentas da Azure:

Push-Based

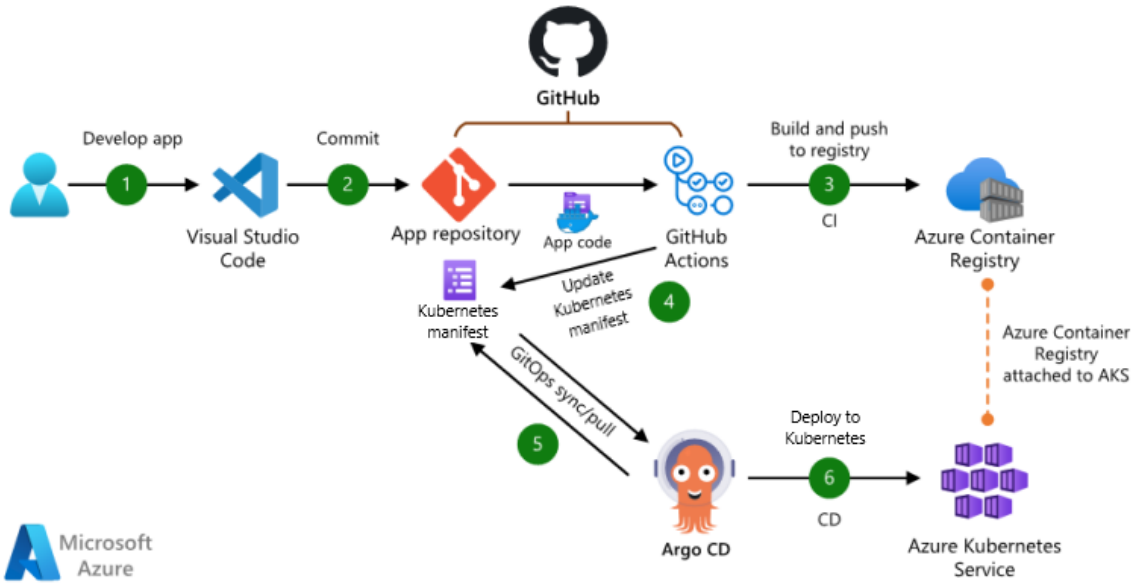
Figura 2.6: Push based pipeline



(a) Fonte: Microsoft Azure

Pull-Based

Figura 2.7: Pull based pipeline



(a) Fonte: Microsoft Azure

Além disso, o GitOps também pode se beneficiar do uso de uma ferramenta específica chamada Fleet. O Fleet é uma solução de GitOps da empresa Rancher Labs que permite gerenciar e sincronizar as configurações de múltiplos *clusters* Kubernetes a partir de um único repositório Git centralizado. Com o Fleet, é possível definir e aplicar configurações consistentes em vários ambientes Kubernetes, facilitando a administração e garantindo a conformidade das implantações (RANCHER, 2023).

O Fleet pode ser implementado como GitOps em um modelo *push ou pull* (CHIA, 2021). O modelo *push* utiliza um *pipeline* CI/CD para enviar alterações para o ambiente, desencadeado por *commits* ou mesclagens de código. É mais simples, flexível, eficiente e mais fácil de padronizar em cargas de trabalho nativas em nuvem e tradicionais. Por outro lado, o modelo pull tem um agente que verifica continuamente o repositório Git e/ou registro de contêineres em busca de alterações, sendo adequado para cargas de trabalho nativas em nuvem e redução de riscos de segurança e conformidade.

2.4
INFRASTRUCTURE AS A CODE - IAC

A infraestrutura de TI tem sido tradicionalmente gerenciada manualmente, exigindo uma série de etapas demoradas e propensas a erros. No entanto, com a crescente complexidade das aplicações e a demanda por agilidade e escalabilidade, surgiu uma abordagem revolucionária para o gerenciamento de infraestrutura: a Infraestrutura como Código (IaC).

Infraestrutura como Código é uma prática que tem se tornado cada vez mais popular entre as empresas que buscam uma gestão mais eficiente de seus recursos tecnológicos. Como o próprio nome sugere, IaC consiste em utilizar código para gerenciar a infraestrutura de TI, incluindo máquinas virtuais, *containers*, aplicativos e bancos de dados. A ideia por trás da IaC é tratar a infraestrutura como qualquer outro componente de *software*, aplicando práticas de desenvolvimento de *software*, como controle de versão, revisão de código e automação de implantação. Com a IaC, é possível criar e modificar infraestruturas de forma ágil, consistente e repetível, garantindo a confiabilidade e a escalabilidade dos sistemas (LUCAS, 2023).

A principal vantagem da IaC é a possibilidade de automatizar todo o processo de implementação, configuração e gerenciamento da infraestrutura, permitindo uma grande economia de tempo e recursos. Além disso, a IaC também ajuda a evitar erros humanos e aumentar a segurança, já que todo o processo é padronizado e pode ser facilmente auditado. Outra das principais vantagens da IaC é a escalabilidade. Com um sistema automatizado de gerenciamento de infraestrutura, é possível expandir rapidamente a capacidade da empresa, sem precisar investir em recursos adicionais. Além disso, a IaC também ajuda a manter um controle mais preciso sobre as versões dos aplicativos e bancos de dados, possibilitando uma recuperação mais rápida em caso de falhas ou desastres (BUCHANAN, 2023).

Existem diversas ferramentas disponíveis no mercado para implementar a IaC, como o Terraform, Ansible, Chef, AWS CloudFormation e SaltStack. Cada uma dessas ferramentas possui suas próprias características e vantagens, e a escolha da melhor opção depende das necessidades específicas da empresa (DELGADO, 2020).

- Terraform: Uma ferramenta de código aberto que permite a criação, alteração e destruição de infraestruturas usando uma linguagem declarativa. O Terraform suporta múltiplos provedores de nuvem, como AWS, Azure e Google Cloud, e permite que os recursos sejam definidos em arquivos de configuração reutilizáveis (HASHICORP, s.d.).

- AWS CloudFormation: Um serviço da Amazon Web Services (AWS) que permite a criação e o gerenciamento de recursos de infraestrutura usando arquivos YAML ou JSON. O CloudFormation oferece recursos para criar pilhas de infraestrutura, que podem incluir instâncias EC2, grupos de segurança, balanceadores de carga e muito mais (AMAZON, s.d.[a]).
- Ansible: Uma ferramenta de automação de TI de código aberto que permite a configuração e o gerenciamento de servidores e infraestrutura de forma programática. O Ansible utiliza uma linguagem simples baseada em YAML para descrever as tarefas de configuração e provisão, facilitando a automação de processos complexos (REDHAT, 2020c).
- Chef: Uma plataforma de automação de TI que permite o gerenciamento e a configuração de infraestruturas complexas. O Chef utiliza uma linguagem de domínio específico (DSL) para descrever a configuração dos sistemas, permitindo que os administradores definam e apliquem políticas de configuração consistentes em diferentes ambientes (DELGADO, 2020).
- SaltStack: O SaltStack é uma plataforma de automação de TI e gerenciamento de configuração. Ele permite automatizar tarefas de gerenciamento de sistemas, configurar aplicativos e provisionar infraestrutura de forma eficiente. Com uma arquitetura cliente-servidor, ele usa a linguagem SaltStack State para definir a configuração dos sistemas (PIRES, 2018).

A Infraestrutura como Código (IaC) revolucionou como a infraestrutura de TI é gerenciada. Ao tratar a infraestrutura como *software*, usando práticas de desenvolvimento de *software*, como controle de versão e automação, a IaC permite uma abordagem ágil, consistente e repetível para a criação e o gerenciamento de ambientes de TI. A adoção da IaC traz benefícios significativos, como a redução de erros, a agilidade na implantação e a capacidade de reverter alterações indesejadas. Com a IaC, as organizações podem avançar em direção a uma infraestrutura mais ágil, flexível e automatizada, impulsionando a inovação e a entrega de valor aos usuários finais.

2.5 ORQUESTRAÇÃO

A orquestração de *containers* é um conjunto de práticas e tecnologias que permitem gerenciar eficientemente a implantação, o dimensionamento e a manutenção de aplicações em *containers*. Os *containers* são unidades leves e isoladas que encapsulam um aplicativo e suas dependências, permitindo que eles sejam executados consistentemente em diferentes ambientes (GUEDES, 2021).

A orquestração de *containers* é essencial para gerenciar ambientes de desenvolvimento e produção por diversos motivos. Primeiro, ela simplifica o processo de implantação de aplicações em *containers*. Com a orquestração, é possível definir a configuração desejada do ambiente em um arquivo declarativo, que especifica quais *containers* devem ser implantados, suas dependências e configurações. Isso facilita a replicação consistente do ambiente em diferentes estágios, desde o desenvolvimento até a produção. Além disso, a orquestração de *containers* permite o dimensionamento automático e dinâmico dos aplicativos. À medida que a carga de trabalho aumenta, a orquestração pode adicionar novas instâncias de *containers* para lidar com a demanda, garantindo que os aplicativos permaneçam disponíveis e com bom desempenho. Da mesma forma, quando a carga diminui, os *containers* podem ser reduzidos automaticamente, economizando recursos e otimizando o uso da infraestrutura (REDHAT, 2019).

Outro benefício importante da orquestração de *containers* é a facilitação da comunicação e do balanceamento de carga entre os *containers* (GUEDES, 2021). Com a orquestração, é possível definir regras de rede e políticas de comunicação para permitir que os *containers* se comuniquem entre si de maneira eficiente e segura. Além disso, a orquestração pode distribuir o tráfego de entrada equilibradamente entre os *containers*, garantindo que nenhum deles seja sobrecarregado e que a carga, seja distribuída uniformemente (REDHAT, 2019).

A orquestração de *containers* também desempenha um papel fundamental na resiliência e recuperação de falhas. Ao usar a orquestração, é possível definir políticas de monitoramento e detecção de falhas, permitindo que os *containers* sejam reiniciados automaticamente ou substituídos em caso de falhas. Isso ajuda a garantir que os aplicativos permaneçam em funcionamento mesmo diante de problemas inesperados, minimizando o tempo de inatividade e os impactos para os usuários finais.

2.5.1

CONTAINERS

Os *containers* são uma tecnologia de virtualização que operam no nível do sistema operacional e que permite a criação e o isolamento de ambientes independentes para a execução de aplicativos. Eles oferecem uma maneira eficiente e portável de empacotar um *software*, juntamente com todas as suas dependências e configurações necessárias, em uma unidade isolada chamada de *container*. Essa abordagem facilita a implantação consistente e confiável de aplicativos em diferentes ambientes, desde desenvolvimento até produção (REDES, 2021).

Um *container* contém todos os componentes necessários para executar um aplicativo, incluindo o código, bibliotecas, frameworks, variáveis de ambiente e arquivos de configuração. Ele é construído a partir de uma imagem de *container*, a qual é uma

representação estática de um sistema de arquivos pré-configurado. As imagens de *container* são criadas a partir de um processo chamado de ”build”, no qual são especificadas as dependências e configurações necessárias. Essas imagens podem ser compartilhadas e reutilizadas, garantindo consistência e facilidade de distribuição do aplicativo (MARCIO, 2019).

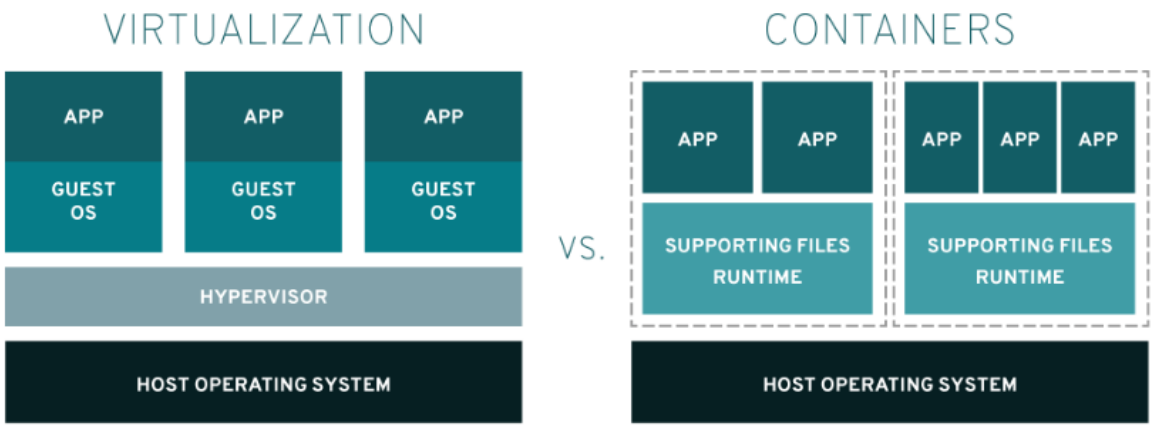
Os *containers* e as Máquinas Virtuais (*VMs*) são tecnologias de virtualização com propósitos distintos e apresentam diferenças importantes. Os *containers* oferecem um nível de isolamento de recursos mais leve em comparação com as *VMs*, já que compartilham o kernel do sistema operacional hospedeiro. Isso resulta em inicialização mais rápida e menor consumo de recursos. Por outro lado, as *VMs* virtualizam todo o sistema operacional, incluindo o kernel, proporcionando maior isolamento entre os aplicativos (PORTNOY, 2016).

Os *containers* possuem menos sobrecarga de recursos em relação às *VMs*. Enquanto cada *VM* requer um sistema operacional completo, vários *containers* podem ser executados em um único sistema operacional hospedeiro, aproveitando melhor os recursos disponíveis. Essa eficiência de recursos faz dos *containers* uma opção mais escalável e permite uma utilização mais eficiente da infraestrutura.

A portabilidade é uma característica-chave dos *containers*. Eles encapsulam o aplicativo e suas dependências em uma unidade isolada, possibilitando a execução em diferentes ambientes, desde laptops de desenvolvimento até data centers em nuvem. As *VMs*, por sua vez, são mais dependentes da infraestrutura subjacente, uma vez que requerem um hipervisor compatível para execução.

Em relação à segurança, as *VMs* oferecem um maior nível de isolamento entre os aplicativos, já que cada *VM* possui seu próprio sistema operacional e kernel. Isso é benéfico em ambientes que exigem separação rígida entre diferentes serviços ou quando há requisitos de segurança mais rigorosos. No entanto, com as configurações adequadas, os *containers* também podem fornecer um bom nível de isolamento e segurança (REDHAT, 2020a).

Figura 2.8: Diferença entre *VM* e *containers*



(a) Fonte: https://www.redhat.com/rhdc/managed-files/virtualization-vs-containers$_{transparent.png}$

Para gerenciar e trabalhar com *containers*, existem várias ferramentas disponíveis, cada uma com suas características específicas. Alguns exemplos populares incluem:

— Docker e Kubernetes que serão abordados com mais detalhes posteriormente;

— Podman: O Podman é uma ferramenta alternativa ao Docker, que permite a execução de *containers* sem a necessidade de um daemon em execução em segundo plano. Ele oferece uma experiência similar ao Docker, permitindo a criação, execução e gerenciamento de *containers* e imagens. (PODMAN, 2023)

— CRI-O: O CRI-O é uma implementação do Kubernetes Container Runtime Interface (CRI) que fornece uma interface compatível com o Kubernetes para execução de *containers*. Ele é projetado para ser leve e otimizado para ambientes Kubernetes, oferecendo um tempo de inicialização mais rápido e menor consumo de recursos em comparação com outras ferramentas. (FOUNDATION, s.d.)

Essas são apenas algumas das muitas ferramentas disponíveis para trabalhar com *containers*. Cada uma possui suas próprias características e casos de uso específicos.

2.5.2

DOCKER

O Docker é uma plataforma de código aberto que revolucionou a forma como as aplicações são desenvolvidas, empacotadas e implantadas. Ele é amplamente utilizado na indústria de *software* devido à sua abordagem eficiente e portátil de empacotar e

executar aplicativos em ambientes isolados chamados de *containers* (DANIELA, 2023).

Em sua essência, o Docker permite que os desenvolvedores criem, distribuam e executem aplicativos dentro de *containers*. Essa abordagem de *build* garante que o aplicativo seja executado de maneira consistente, independentemente do ambiente em que está sendo implantado. O uso do Docker traz diversos benefícios para o desenvolvimento e a implantação de aplicativos. Ele simplifica o processo de *build* e distribuição, melhora a consistência entre diferentes ambientes e permite uma implantação ágil e escalável. Com sua popularidade crescente, o Docker se tornou uma ferramenta essencial no ecossistema de desenvolvimento de *software* moderno (DOCKER, 2022).

Uma das principais vantagens do Docker é a portabilidade. Os *containers* Docker podem ser executados em qualquer sistema operacional que tenha suporte ao Docker, seja um ambiente de desenvolvimento local, servidores em nuvem ou *data centers*. Isso significa que os desenvolvedores podem criar um ambiente consistente em suas máquinas locais e, em seguida, implantar os *containers* em qualquer outro ambiente sem se preocupar com problemas de compatibilidade. Outro benefício do Docker é a eficiência de recursos. Ao contrário das tradicionais máquinas virtuais, o Docker compartilha o kernel do sistema operacional hospedeiro entre os *containers*, resultando em uma inicialização mais rápida e um consumo de recursos mais baixo. Múltiplos *containers* podem ser executados simultaneamente em um único servidor, maximizando a utilização dos recursos disponíveis (ABSAM, 2023).

O Docker também possui uma vasta gama de ferramentas e recursos que facilitam a construção e o gerenciamento de *containers*. O Docker Hub, por exemplo, é um registro de imagens de *containers*, onde os desenvolvedores podem compartilhar e baixar imagens prontas para uso. O Docker Compose permite definir e gerenciar aplicativos compostos por vários containers interconectados. Além disso, o Docker Swarm e o Kubernetes fornecem recursos de orquestração para facilitar o gerenciamento e a escalabilidade de clusters de *containers* (DOCKER, 2023).

Em resumo, o Docker é uma plataforma de virtualização leve e portátil que permite empacotar, distribuir e executar aplicativos em *containers* isolados. Com sua abordagem eficiente e recursos poderosos, ele simplifica a implantação de aplicativos, oferece consistência entre diferentes ambientes e impulsiona a agilidade no desenvolvimento de *software*.

2.5.3

KUBERNETES - K8S

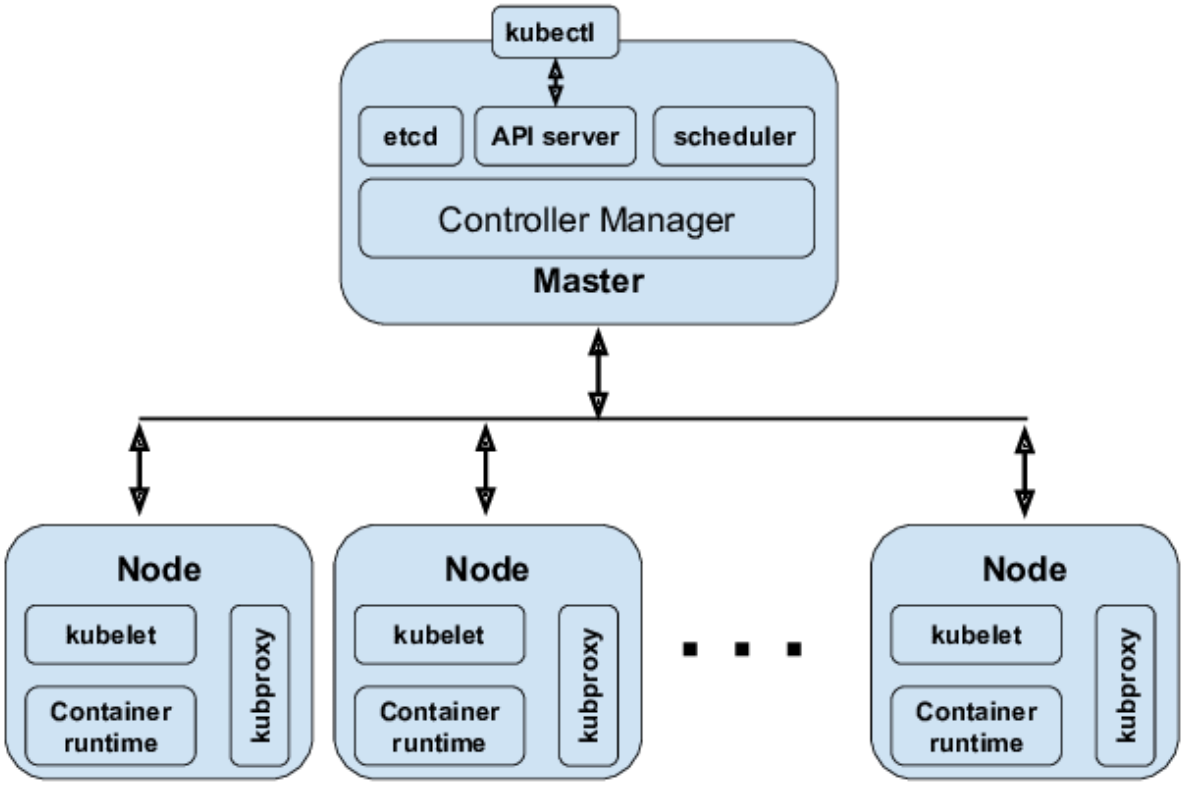
O Kubernetes é uma plataforma de orquestração de *containers* de código aberto que revolucionou como as aplicações são implantadas, dimensionadas e gerenciadas em ambientes distribuídos. Ele foi desenvolvido originalmente pelo Google e agora é mantido pela *cloud* Native Computing Foundation (CNCF), tornando-se uma das ferramentas mais populares e amplamente adotadas no mundo da tecnologia (GONCALVES, 2021).

O Kubernetes permite que você gerencie e automatize o ciclo de vida de seus aplicativos em *containers*. Com o Kubernetes, você pode implantar aplicativos em um cluster de máquinas, que podem variar de um único servidor à centena ou até milhares de máquinas em nuvem. Ele oferece recursos avançados de gerenciamento, dimensionamento, monitoramento e recuperação de falhas para garantir a alta disponibilidade e escalabilidade dos aplicativos. Uma das principais características do Kubernetes é a abstração de infraestrutura. Ele fornece uma camada de abstração que permite aos desenvolvedores e operadores tratar os recursos de computação, armazenamento e rede de maneira uniforme, independentemente da infraestrutura subjacente. Isso permite que você implante seus aplicativos em qualquer ambiente compatível com o Kubernetes, desde data centers locais até provedores de nuvem pública (REDHAT, 2020b).

O Kubernetes opera com base em um modelo declarativo, o que significa que você descreve o estado desejado do seu aplicativo em um arquivo de configuração YAML ou JSON. Em seguida, o Kubernetes se encarrega de garantir que o estado real do aplicativo corresponda ao estado declarado. Ele gerencia automaticamente a alocação de recursos, o escalonamento de *containers*, o balanceamento de carga, a recuperação de falhas e a implantação de novas versões de aplicativos (KUBERNETS, 2022).

Além disso, o Kubernetes possui recursos poderosos para o dimensionamento automático com base na demanda. Ele pode aumentar ou diminuir o número de instâncias de *containers* para garantir que seus aplicativos tenham recursos suficientes disponíveis, mesmo durante picos de tráfego. Isso permite otimizar o uso dos recursos do *cluster*, garantindo uma melhor utilização e redução de custos (SHAUL, 2022).

Figura 2.9: Componentes kubernetes



(a) Fonte: researchgate.net/

O ecossistema do Kubernetes é rico em ferramentas e serviços complementares. Por exemplo, o Helm é um gerenciador de pacotes que facilita o compartilhamento e a implantação de aplicativos em um cluster Kubernetes. O Istio fornece recursos avançados de gerenciamento de tráfego e segurança para aplicações em microserviços. Além disso, existem várias soluções de monitoramento, logística e CI/CD (Integração Contínua/Entrega Contínua) integradas ao Kubernetes. Abaixo temos mais alguns exemplos de ferramentas e serviços do Kubernetes (REDHAT, 2020b).

- kubectl: É a ferramenta de linha de comando oficial do Kubernetes. Ela permite interagir com os clusters Kubernetes, gerenciando pods, serviços, deployments e outros recursos do Kubernetes.
- Minikube: É uma ferramenta que permite executar um cluster Kubernetes em uma máquina local para fins de desenvolvimento e testes. Possibilita explorar e aprender sobre o Kubernetes sem precisar de um ambiente de produção.
- kubeadm: É uma ferramenta para configurar e inicializar clusters Kubernetes. Ela automatiza várias etapas envolvidas na configuração de um cluster, como a instalação de componentes principais e a geração de arquivos de configuração.

- Helm: É um gerenciador de pacotes para o Kubernetes. Ele permite criar, compartilhar e implantar aplicativos em um cluster Kubernetes usando “charts”, os quais são pacotes que contêm todos os recursos necessários para executar um aplicativo (HELM, 2023).
- Prometheus: É um sistema de monitoramento e alerta de código aberto que é frequentemente usado em conjunto com o Kubernetes. Ele coleta métricas do cluster Kubernetes e dos aplicativos nele executados, permitindo monitorar o desempenho, a disponibilidade e a saúde do sistema (BURILLO, 2021).
- Grafana: É uma plataforma de visualização de dados que pode ser integrada ao Prometheus e a outras fontes de dados. Ela fornece painéis interativos e personalizáveis para visualizar métricas e estatísticas coletadas pelo Prometheus (LABS, 2022).

A implantação do Kubernetes(k8s) pode ser dada de duas formas gerenciada e não gerenciada. O Kubernetes não gerenciado é uma opção de implantação de cluster Kubernetes que não é gerenciada por um provedor de nuvem ou por uma plataforma de gerenciamento de contêineres. Isso significa que, em vez de delegar tarefas de gerenciamento para um provedor terceirizado, a empresa é responsável por configurar, implantar e gerenciar seu próprio cluster Kubernetes.

Embora essa opção possa ser mais trabalhosa do que usar um provedor gerenciado, ela oferece maior controle e flexibilidade sobre a implantação. Além disso, pode ser uma opção mais econômica para empresas que não precisam de todos os recursos oferecidos pelos provedores gerenciados. (FENSTERER, 2022) No entanto, é importante lembrar que a implantação de um cluster Kubernetes não gerenciado requer uma compreensão profunda da plataforma e de suas ferramentas. As empresas precisam investir tempo e recursos na configuração e manutenção do cluster, além de garantir que seus funcionários tenham as habilidades necessárias para gerenciá-lo (CAREY, 2021).

Além disso, a implantação de um cluster Kubernetes não gerenciado requer uma atenção cuidadosa à segurança, uma vez que a responsabilidade pela proteção dos dados e dos recursos recai inteiramente sobre a empresa. Isso inclui a implementação de políticas de acesso, monitoramento de segurança e aplicação de patches e atualizações regularmente. Outra consideração importante é a escalabilidade. Enquanto provedores de nuvem oferecem escalabilidade automática e gerenciamento de recursos, em um ambiente não gerenciado, a empresa precisa planejar e provisionar os recursos necessários para suportar o crescimento da carga de trabalho. Isso requer uma análise cuidadosa dos requisitos e um monitoramento constante do desempenho

do *cluster* (WATTS, 2022).

Apesar dos desafios, muitas empresas optam por um *cluster* Kubernetes não gerenciado devido à sua flexibilidade e controle. Elas podem personalizar a configuração do *cluster* de acordo com suas necessidades específicas e adaptá-lo para diferentes casos de uso. Além disso, um cluster não gerenciado pode ser uma opção preferida para organizações com requisitos de conformidade estritos, permitindo que elas tenham controle total sobre os aspectos de segurança e conformidade do ambiente (CAREY, 2021).

O Kubernetes gerenciado é uma opção de implantação de cluster Kubernetes oferecida por provedores de nuvem e plataformas de gerenciamento de contêineres. Ao optar por um Kubernetes gerenciado, as empresas podem contar com a infraestrutura fornecida pelo provedor para configurar, implantar e gerenciar seus *clusters* Kubernetes, enquanto delegam tarefas de gerenciamento operacional para o provedor (FENSTERER, 2022).

Uma das principais vantagens do Kubernetes gerenciado é a simplicidade e conveniência que ele proporciona. Os provedores de nuvem cuidam da maioria das tarefas operacionais, como provisionamento de recursos, escalabilidade automática, monitoramento e aplicação de patches e atualizações do Kubernetes e redução de custos de infraestrutura. Isso libera a equipe interna para se concentrar em outras tarefas importantes, como desenvolvimento de aplicativos e inovação.(CAREY, 2021).

Existem muitos provedores de Kubernetes gerenciados no mercado, como o Amazon Elastic Kubernetes Service (EKS), o Azure Kubernetes Service (AKS) e o Google Kubernetes Engine (GKE) (GONCALVES, 2021). Esses provedores oferecem serviços gerenciados de Kubernetes, o que significa que eles lidam com as tarefas de gerenciamento de infraestrutura, como provisionamento de nós e atualizações de *software* (KUBERNETS, s.d.).

2.5.4

RANCHER

O Rancher é uma plataforma de gerenciamento de contêineres que oferece uma ampla gama de recursos e funcionalidades para implantar, gerenciar e dimensionar aplicativos em qualquer infraestrutura. Projetado especialmente para simplificar a implantação do Kubernetes, o Rancher facilita o trabalho das equipes de TI ao permitir que elas

implantem e gerenciem *clusters* Kubernetes de qualquer tamanho ou complexidade (CONTAINERIZE, s.d.).

O Rancher apresenta uma das principais vantagens de ser acessível por meio de uma interface gráfica intuitiva, proporcionando uma experiência de gerenciamento simplificada para os administradores de sistema. Esta interface permite que seja possível visualizar e controlar todos os *clusters* Kubernetes em execução, acompanhar o desempenho dos aplicativos, gerenciar os recursos de *hardware* e ajustar as configurações conforme necessário. Além disso, o Rancher oferece recursos avançados de monitoramento, permitindo que os administradores acompanhem o status e o desempenho dos aplicativos e dos nós de cluster. Isso inclui métricas detalhadas sobre o uso de CPU, memória e armazenamento, bem como informações sobre a saúde dos pods e *containers* em execução. Com esses insights, as equipes de TI podem identificar e resolver problemas rapidamente, garantindo uma operação eficiente e confiável (RODRIGUES, 2022).

O Rancher fornece um sistema sólido de gerenciamento de usuários e equipes. Com ele, você pode configurar diferentes níveis de acesso e permissões para os usuários, permitindo que as equipes colaborem e trabalhem com segurança. Isso é especialmente útil em organizações com várias equipes de desenvolvimento e operações, onde é fundamental controlar o acesso a *clusters* Kubernetes e garantir a conformidade com as políticas internas (SOUZA, 2023).

Com o Rancher, as empresas podem implantar aplicativos com rapidez e eficiência. A plataforma simplifica todo o processo de implantação, desde a criação e configuração de *clusters* Kubernetes até a implantação de aplicativos. Isso economiza tempo e recursos valiosos, liberando as equipes de TI para se concentrar em outras tarefas críticas.

Além de trazer facilidade na implantação e gerenciamento de *clusters* kubernetes, o Rancher provê também funcionalidades de CI/CD que podem ser utilizadas no deployment e gerenciamento de aplicações, bem como da infraestrutura, um exemplo é o Fleet.

O Fleet é um recurso do Rancher que permite gerenciar múltiplos *clusters* Kubernetes em diferentes ambientes. Com o Fleet, é possível definir e aplicar políticas de configuração em larga escala, além de implantar aplicativos e atualizações consistentemente em todos os *clusters*. O Fleet também oferece recursos de monitoramento e diagnóstico integrados, permitindo que os administradores visualizem o status dos

clusters e identifiquem problemas em tempo real. Com sua abordagem centrada em GitOps, o Fleet torna o gerenciamento de infraestrutura em larga escala mais fácil e eficiente do que nunca (RANCHER, 2023).

No conjunto de práticas do GitOps, as configurações e as definições de implantação dos aplicativos são armazenadas em um repositório Git. O Fleet monitora continuamente esse repositório e garante que as alterações sejam aplicadas consistentemente em todos os *clusters*. Isso significa que os administradores podem implantar novos aplicativos, atualizações e correções de maneira confiável e controlada, em escala. O Fleet também oferece recursos integrados de monitoramento e diagnóstico. Os administradores podem visualizar o status de todos os *clusters* e seus componentes, bem como monitorar métricas e logs em tempo real. Isso permite identificar problemas e tomar medidas corretivas rapidamente, garantindo que os *clusters* estejam sempre em um estado saudável e funcionando de maneira eficiente (RANCHER, 2023).

3

METODOLOGIA

A metodologia utilizada no desenvolvimento do trabalho foi o PDCA que consiste em um método de gestão e melhoria contínua composto por quatro etapas: planejar (Plan), executar (Do), verificar (Check) e agir (Act). O ciclo começa com o planejamento de metas e ações, seguido pela execução dessas ações. Depois, é feita a verificação dos resultados e comparação com as metas estabelecidas. Por fim, são tomadas medidas corretivas ou preventivas para melhorar o desempenho e iniciar um novo ciclo de melhoria.

Começando pela fase do planejar, foram definidos os objetivos desse trabalho que consistem na criação de uma infraestrutura na nuvem (OCI) totalmente automatizada para hospedar pipelines de CI/CD e o sistema WEB que percorrerá esses pipelines, essa infraestrutura será construída como um código (IaC) e baseada em *cluster* Kubernetes. Para a criação será utilizado conceitos do GitOps, como o versionamento do código de infraestrutura e o provisionamento automático desse código. O objetivo mais abstrato do trabalho é conseguir trazer as práticas DevOps para o dia a dia de desenvolvedores que não tem familiaridade com infraestrutura através do IaC e GitOps.

Para ter uma visualização melhor desses objetivos temos a tabela 3.1:

Tabela 3.1: Objetivos, indagações e possíveis soluções

Objetivo	Desafios	Possível Solução
Criar a infraestrutura para hospedar um cluster k8s em uma nuvem publica utilizando IaC	- Existe uma conta na nuvem?	Ter acesso a uma conta de um provedor da nuvem onde possa ser feito a criação dessa infraestrutura.
	- Essa nuvem possibilita que usemos IaC?	Acesso à documentação da nuvem, para identificar um plugin para o Terraform, assim possibilitando a utilização do IaC.
	- Existe algum problema implementar a infraestrutura nesse provedor?	Lista de possíveis problemas ao utilizar esse provedor para criação da infraestrutura.
Introduzir desenvolvedores a utilização de práticas DevOps no deployment dos seus sistemas web	- As práticas estão bem claras para o entendimento de pessoas não DevOps?	Disponibilizar uma documentação clara e sucinta sobre o DevOps e suas práticas essenciais.
Mostrar a importância do passo de implantação no desenvolvimento de softwares web	- Qual o tamanho dessa etapa no desenvolvimento do sistema?	Explicitar a diferença de recursos e tempo demandado quando se seguem as recomendações ideias para a etapa de implantação e que isso não é apenas de importância para o time de operações.
Aplicar conceitos de GitOps para versionamento, controle e provisionamento da infraestrutura criada	- Quais conceitos adotar?	Todos os conceitos do GitOps são fundamentais para a execução do provisionamento automático da infraestrutura, então explicitar isso no documento.
	- Qual parte versionar?	É essencial que todos os arquivos que compõe a IaC estejam versionados em um repositório, explicar como fazer da forma correta no documento.

A fase de execução contou com um período de 6 meses para capacitação nos assuntos e realização de exemplos. Para construção do objetivo principal precisamos de alguns requisitos fundamentais: Primeiramente para hospedar toda infraestrutura é necessária uma conta na nuvem, para isso foi utilizada a Oracle Cloud Infrastructure (OCI), para realizar a criação da infraestrutura com o Terraform e Ansible é necessário: uma máquina de host com um sistema operacional que suporte o Ansible (máquinas com windows não suportam o Ansible) e as ferramentas devem estar previamente instaladas nesse host, para realizar o versionamento foram utilizadas as ferramentas: Git e o GitHub sendo necessário a criação de uma conta e um repositório no GitHub para hospedar todos os códigos criados. E por fim, mas não menos importante é preciso ter uma conexão com a internet para conectar os plugins do Terraform com a nuvem, para o acesso ao Rancher e para o versionamento dos códigos, todas essas etapas necessitam de acesso à Internet.

Já as etapas de verificação e ação podem ser unificadas, logo após a execução do código criado, foi realizado a verificação dos mesmos e feita a validação se toda a

infraestrutura corresponde com a necessidade do desafio, por exemplo, foi verificado se o shape (configuração da máquina) das VMs correspondiam com a demanda e se não extrapolavam ou não era suficiente, em seguida foi feito a ação de formatar os shapes para a demanda necessária, um caso que ocorreu foi o de uma VM que era responsável pelo papel de control-plane no K8s estava com o shape standard com um número de CPUs insuficiente para executar o K8s, então na etapa de verificação foi identificado esse erro e logo em seguida foi feita a ação para alterar esse shape para um que correspondesse a necessidade.

4

IMPLEMENTAÇÃO

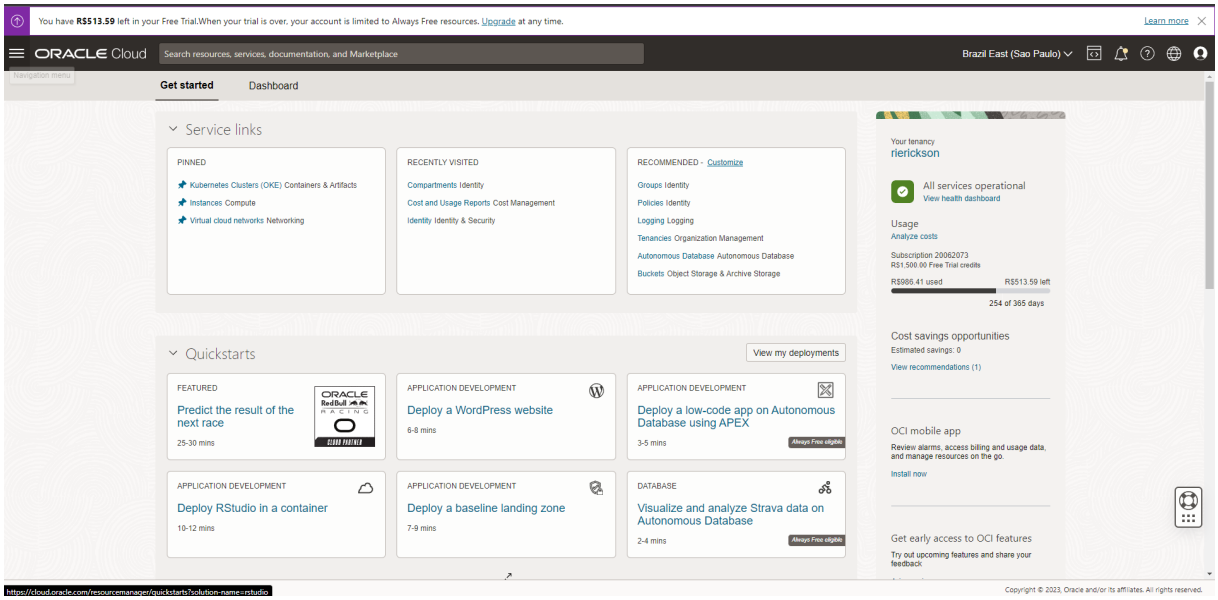
O objetivo deste capítulo é demonstrar como a infraestrutura na Oracle Cloud foi implementada de forma automática com o Kubernetes e Ansible para alojar *cluster*s kubernetes que hospedem aplicações WEB e *pipelines* CI/CD para essas aplicações e após a criação da infraestrutura, mostrar a criação dos *cluster*s k8s e do seu gerenciamento com a ferramenta Rancher.

4.1

PROVEDOR DE NUVEM

Para hospedar toda a infraestrutura que será criada se faz necessário uma conta em um provedor de nuvem. A OCI é uma ótima opção por dois principais motivos: A disponibilização de recursos gratuitos que satisfazem a demanda para ser possível a criação de *cluster* k8s, porém precisaríamos efetuar algumas alterações como a conversão de arquitetura de aplicações de x86 (os quais são a arquitetura dos processadores padrões da AMD e Intel) para ARM (sendo a arquitetura do processador Ampere, que é o Always Free da OCI). Para realizar a implementação foram utilizados recursos pagos e isso foi possível por que a OCI disponibiliza uma conta de testes para estudantes com 1.500 reais de créditos, e esse foi o segundo motivo pelo qual a OCI foi escolhida para o desenvolvimento desse trabalho. Após a criação da conta na Oracle, é possível acessar o dashboard da sua cloud.

Figura 4.1: Dashboard da OCI



(a) Fonte: Próprio Autor (2023)

Com uma conta já funcional, é preciso arquitetar a infraestrutura para satisfazer os requisitos do projeto.

4.2 ARQUITETANDO A INFRAESTRUTURA

Antes de iniciar a codificação da infraestrutura em si, é preciso pensar em algumas variáveis que podem ser fundamentais no desenvolvimento do projeto como:

- Será utilizado kubernetes gerenciado ou não gerenciado? Essa pergunta é de extrema importância para a continuação do desenvolvimento, porque ela altera todo o seu planejamento e os recursos que serão necessários.
- Quais shapes utilizar nos nodes? Para os *clusterss* funcionarem corretamente é importante que os requisitos utilizados sejam o suficiente e adequados para as cargas de trabalhos executadas e não superdimensionados, evitando assim o consumo rápido dos créditos ou pagamento de altos valores quando se utiliza contas pagas. No Always Free (modo gratuito da OCI) não é possível usar shapes grandes e caros porque a cloud não os disponibiliza, vide que são oferecidos apenas recursos gratuitos e com limitações.
- Qual virtual cloud network (VCN) utilizar? As criadas por padrão nos *clusterss* k8s ou criar sua própria VCN? Essa escolha depende muito da resposta a primeira

questão, se utilizarmos o kubernetes gerenciado a VCN já será criada por padrão, já se escolher o k8s não gerenciado é necessário criar a própria VCN.

4.2.1

K8S GERENCIADO VS K8S NÃO GERENCIADO

Para o desenvolvimento do projeto, foi escolhido o k8s gerenciado, ao escolher o Kubernetes gerenciado em vez do Kubernetes não gerenciado, opta-se por uma solução que traz abstrações valiosas que facilitam a implementação. Isso significa que a equipe de operações e/ou desenvolvimento não precisa se preocupar com a configuração e manutenção do *cluster* do Kubernetes, balanceamento de carga, escalabilidade automática ou atualizações de versões. Essas tarefas são tratadas pelo provedor, permitindo que você se concentre no desenvolvimento e na implantação de aplicativos.

O k8s gerenciado oferece alta disponibilidade, uma vez que os *cluster*s são distribuídos em várias zonas de disponibilidade ou regiões, garantindo a resiliência e a continuidade dos serviços. Outro benefício é a possibilidade de integração com outras ferramentas e serviços nativos da nuvem, como armazenamento, banco de dados e monitoramento, proporcionando uma infraestrutura robusta e escalável. Por fim, a segurança também é aprimorada com um k8s gerenciado, uma vez que as atualizações de segurança e *patches* são aplicados automaticamente pela plataforma, mantendo os *cluster*s protegidos contra ameaças emergentes. E esses detalhes que são abstraídos contam muito, principalmente para equipes reduzidas onde cada um dos integrantes desempenham mais de uma função, o ganho de tempo e de esforço com o k8s gerenciado pode ser crucial em alguns projetos.

Por esses motivos e ganho de velocidade de implementação e diminuição de complexidade é preferível utilizar o k8s gerenciado. O OKE (Oracle Container Engine for Kubernetes) é um serviço de orquestração de contêineres fornecido pela Oracle. Ele é baseado na tecnologia Kubernetes e oferece aos usuários a capacidade de implantar, gerenciar e dimensionar aplicativos em contêineres de forma eficiente e confiável (ORACLE, 2022).

4.2.2

RECURSO DE HARDWARE NECESSÁRIO

Na utilização de um *clusters* não gerenciado é necessário instanciar pelo menos 3 VMs — um control-plane e pelo menos dois nodes. Os requisitos para o control-plane são:

uma máquina com sistema operacional Linux compatível (o projeto Kubernetes provê instruções para distribuições Linux baseadas em Debian e Red Hat, bem como para distribuições sem um gerenciador de pacotes, 2 GB ou mais de RAM por máquina, menos que isso deixará pouca memória para as aplicações, 2 CPUs ou mais; conexão de rede entre todas as máquinas no *cluster* — seja essa pública ou privada; portas específicas abertas em máquinas que estão em seu *cluster*, seja nodes ou control-plane.) (KUBERNETES, 2023).

Já utilizando o k8s gerenciado a utilização do shape mais básico da classe *standard* o 2.1, que possui 1 Oracle CPU (OCPU) e 2 MAX VNIC (Virtual Cloud interface Network), esse plano é suficiente para instanciar as maquinas do pool do OKE.

4.2.3

VCN PADRÃO OU VCN PRÓPRIA

No momento da criação do *cluster* a OCI possibilita que o usuário escolha criar sua própria Virtual Cloud Network (VCN) ou utilizar uma VCN criada automaticamente pela própria OCI. A vantagem de criar sua própria network é que você possui controle sobre a security list (a security list é responsável pelo controle de acesso a VCN por meio de regras para IP e portas), range de IP e algumas outras configurações importantes para o funcionamento da sua network. Abaixo temos a implementação de um VCN como um código, os detalhes serão abordados em outro tópico, de momento é apenas um exemplo para mostrar como é a definição de sua própria VCN com o Terraform.

Figura 4.2: Implementação VCN como código

```
You, 2 months ago | 1 author (You)
resource "oci_core_subnet" "subnet-nodepool" {
  cidr_block        = cidrsubnet(var.vcn_cidr, 8, 0)
  compartment_id    = var.comapartment_ocid
  display_name      = "subnetnodepool"
  vcn_id            = oci_core_virtual_network.vcn.id
  route_table_id    = oci_core_route_table.rt.id
  dns_label         = "subnetnodepool"
  security_list_ids = [oci_core_security_list.sl.id]
}
```

(a) Fonte: Próprio Autor (2023)

Apesar da criação padrão de uma VCN nos garantir ganho de tempo, não aparenta

ser a melhor escolha já que o controle de acesso via Internet dos seus nodes são um fator de extrema importância para manter a segurança da infraestrutura, sendo assim preferível utilizar uma VCN criada pelo usuário, porém, se o objetivo é ganhar tempo e diminuir complexidade recomenda-se utilizar uma VCN criada pela própria OCI.

Figura 4.3: Configurando a network da pool do k8s

(a) Fonte: Próprio Autor (2023)

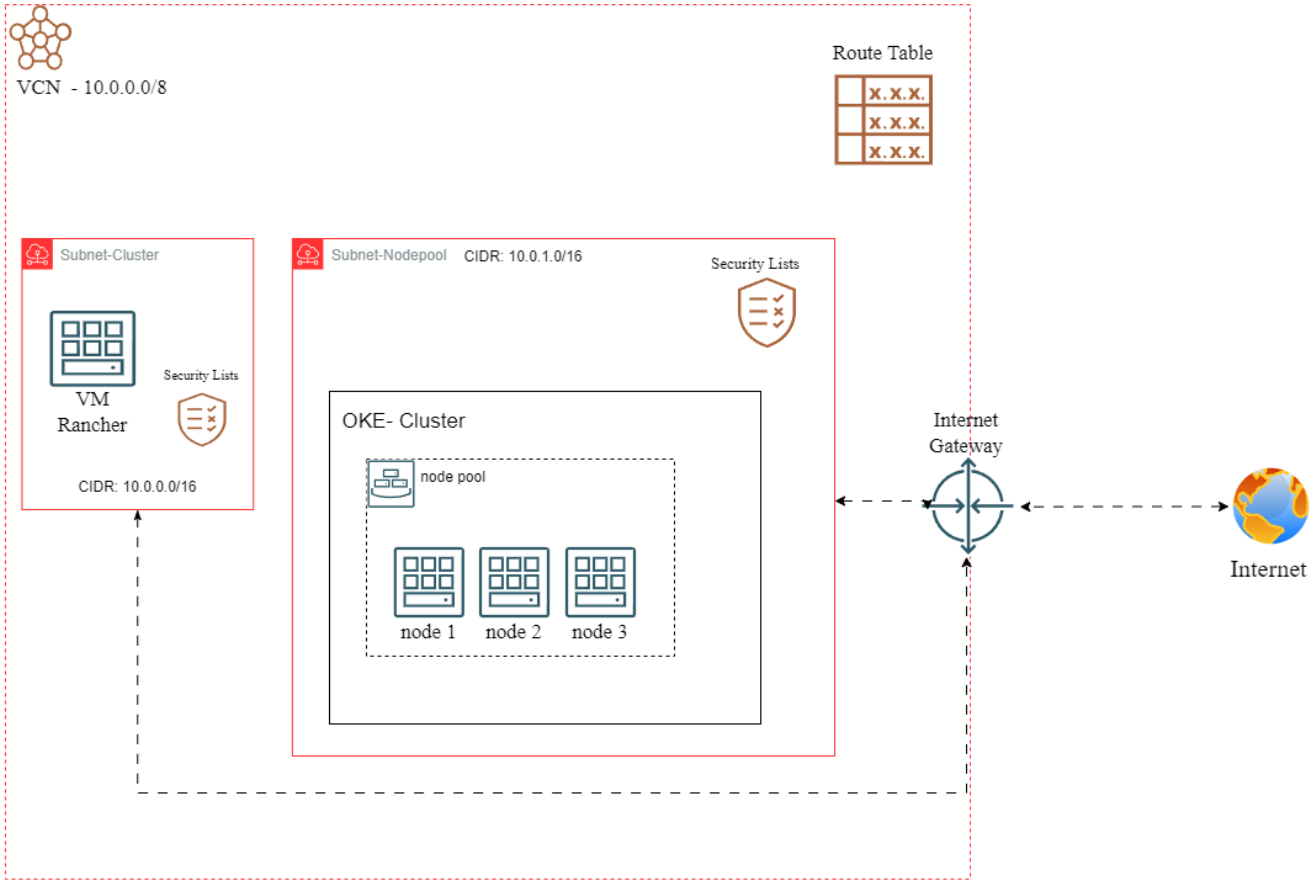
Como mostrado na imagem 4.3, na configuração dos nodes do k8s definimos como subnet a rede criada manualmente.

4.2.4

REPRESENTAÇÃO DA INFRAESTRUTURA

Com essas perguntas respondidas, é ideal que se construa um diagrama para ficar mais fácil a visualização e a compreensão de toda infraestrutura que será construída como um código. A seguir segue a representação da infraestrutura arquitetada:

Figura 4.4: Representação da infraestrutura



(a) Fonte: Próprio Autor (2023)

4.3

CODIFICANDO A INFRAESTRUTURA

O primeiro passo para começar a codificar a infraestrutura que foi previamente arquitetada é a decisão da ferramenta que será responsável por essa automação, essa é uma das decisões mais importantes, escolher a ferramenta certa para automatizar a implantação e o gerenciamento de recursos de infraestrutura.

O Terraform será utilizado por algumas vantagens em relação as demais ferramentas existentes no mercado: Popularidade no mercado — o Terraform é a ferramenta mais utilizada atualmente isso garante que ela é atual e possui um grande fórum para esclarecimento de possíveis dúvidas; Facilidade de compreensão — o Terraform destaca-se por sua sintaxe simples e legível, facilitando a compreensão e o aprendizado da ferramenta. Sua abordagem declarativa permite descrever a infraestrutura desejada usando uma linguagem comum; Segurança — a segurança é uma consideração essencial ao lidar com infraestrutura de TI. O Terraform fornece recursos robustos

para garantir a segurança dos recursos provisionados; Rapidez — a automação é um dos principais benefícios do IaC, e o Terraform se destaca pela rapidez em fornecer infraestrutura consistente e replicável. Com a possibilidade de escrever e executar scripts, é possível provisionar e configurar recursos de maneira eficiente, economizando tempo e esforço (KUBERNETS, s.d.).

Definido que a será utilizado o Terraform, para começar a codificação é preciso escolher um provider. Os providers do Terraform são componentes essenciais que permitem ao Terraform interagir com diferentes serviços e plataformas de infraestrutura, como provedores em nuvem, serviços de rede e provedores de armazenamento. Eles atuam como tradutores entre as definições de recursos e configurações do Terraform e as chamadas de API específicas do provedor.)

Como estamos utilizando a OCI, ela possui seu provider para o Terraform, assim que fica configurado o provider:

Figura 4.5: Provider OCI no Terraform

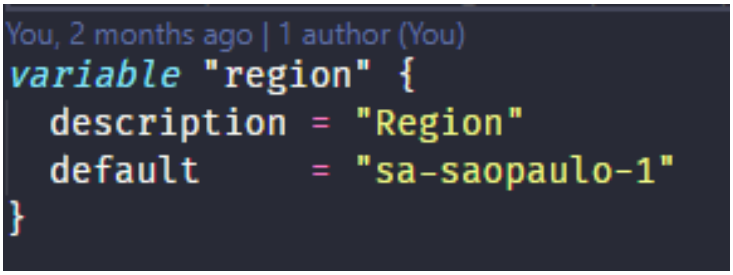
```
provider "oci" {
  tenancy_ocid     = var.tenancy_ocid
  user_ocid        = var.user_ocid
  fingerprint      = var.fingerprint
  private_key_path = var.private_key_path
  region           = var.region
}
```

(a) Fonte: Próprio Autor (2023)

Para configurar esse provider é necessário cinco variáveis da sua cloud, tenancy-ocid é o id da tenancy da conta OCI, user-ocid é o id do usuário, fingerprint é um conjunto de caracteres gerado após a confirmação de uma chave de acesso que é a private key da conta, o private key path é o caminho na máquina host onde está localizada a key e por fim a region é onde sua conta está hospedada.

Todos essas variáveis podem ser armazenada em um arquivo chamado variables.tf que pode ser estruturado dessa forma:

Figura 4.6: Variáveis no Terraform



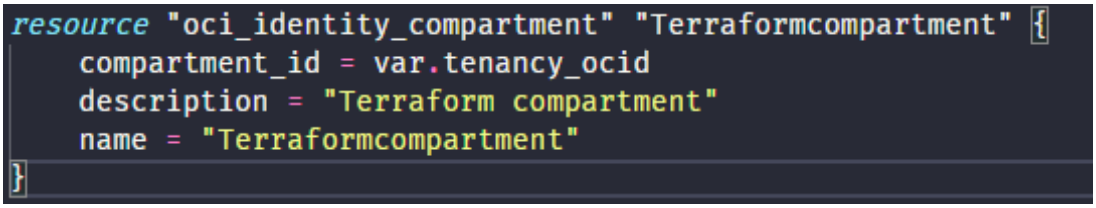
(a) Fonte: Próprio Autor (2023)

Toda variável que precise ser armazenada pode ser declara nesse formato, onde description é a descrição para ficar mais legível e default é o valor da variável que será armazenado.

4.4
CRIANDO A VCN

Dando início a criação da infraestrutura planejada, segue abaixo o código para criação de um *compartment (compartimento*:

Figura 4.7: Compartment OCI no terraform



(a) Fonte: Próprio Autor (2023)

Um compartment é uma estrutura organizacional que isola e organiza recursos, compartments ajudam a gerenciar recursos de forma mais eficiente, simplificando a navegação, protegendo o acesso aos recursos e permitindo uma melhor organização conforme as necessidades de negócios.

Como arquitetado anteriormente a infraestrutura possui alguns recursos que dependem de outros, então para seguir uma ordem lógica, criam-se primeiramente os recursos dependências como o: Compartment, a VCN, as subnets... Segue o código de criação da VCN:

Figura 4.8: VCN OCI no Terraform

```
...
resource "oci_core_virtual_network" "vcn" {
  cidr_block     = var.vcn_cidr
  compartment_id = var.comapartment_ocid
  display_name   = "vcn-terraform"
  dns_label      = var.vcn_dns_label
}
```

(a) Fonte: Próprio Autor (2023)

Uma VCN OCI é uma rede virtual personalizável e isolada que permite a criação de uma infraestrutura de rede segura e escalável na Oracle Cloud. Ela fornece controle, conectividade e flexibilidade para os recursos e serviços implantados na nuvem da Oracle.

Mas como o diagrama mostra, a VCN precisa de alguns recursos para funcionar corretamente, abaixo temos a implementação do Internet Gateway:

Figura 4.9: Internet Gateway no Terraform

```
...
resource "oci_core_internet_gateway" "igw" {
  compartment_id = var.comapartment_ocid
  display_name   = "igw-terraform"
  vcn_id         = oci_core_virtual_network.vcn.id
}
```

(a) Fonte: Próprio Autor (2023)

O Internet Gateway OCI é um serviço que fornece conectividade entre uma VCN e a Internet. Ele permite a comunicação de recursos dentro da VCN com serviços externos na Internet, fornecendo endereços IP públicos e recursos de roteamento.

O próximo recurso a ser implementado é o Router Table:

Figura 4.10: Router Table no Terraform

```
...
resource "oci_core_route_table" "rt" {
  compartment_id = var.comapartment_ocid
  vcn_id         = oci_core_virtual_network.vcn.id
  display_name   = "rtTerraform"

  ...
  route_rules {
    cidr_block        = "0.0.0.0/0"
    network_entity_id = oci_core_internet_gateway.igw.id
  }
}
```

(a) Fonte: Próprio Autor (2023)

Uma Route Table é um componente que define as regras de roteamento em uma VCN. Ela determina como o tráfego é encaminhado entre as sub-redes e os recursos dentro da VCN. A Route Table permite controlar a conectividade, definir rotas padrão e direcionar o tráfego para serviços externos.

Agora serão criadas duas subnet como especificado no diagrama, uma subnet se destina a nodepool do OKE e outra ficará para o *cluster* e para a VM que ficará encarregada de hospedar o Rancher, ela será mais uma camada de gerenciamento. A Primeira implementação é da subnet do *cluster*:

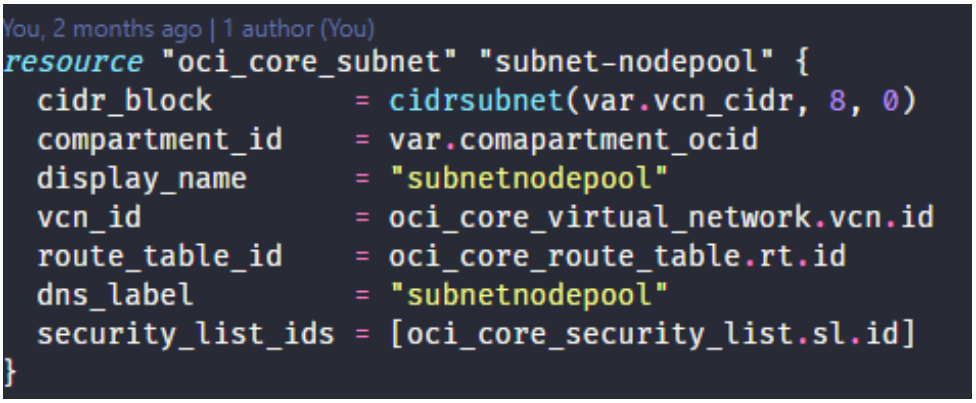
Figura 4.11: Cluster Subnet no Terraform

```
...
resource "oci_core_subnet" "subnet-cluster" {
  cidr_block        = cidrsubnet(var.vcn_cidr, 8, 1)
  compartment_id    = var.comapartment_ocid
  display_name      = "subnetcluster"
  vcn_id            = oci_core_virtual_network.vcn.id
  route_table_id    = oci_core_route_table.rt.id
  dns_label         = "subnetcluster"
  security_list_ids = [oci_core_security_list.sl.id]

}
```

(a) Fonte: Próprio Autor (2023)

A segunda é a implementação da sub-rede de nodepool:

Figura 4.12: Nodepool Subnet no Terraform



(a) Fonte: Próprio Autor (2023)

Uma subnet é uma subdivisão de uma Virtual Cloud Network que permite a segmentação e organização dos recursos de rede. Uma subnet é uma rede delimitada por um intervalo específico de endereços IP e pode ser associada a uma ou mais Availability Domains (os ”Availability Domains- ADs - são data centers independentes e isolados fisicamente uns dos outros em uma região).

Para finalizar a criação da VCN o último recurso a ser codificado é a security list:

Figura 4.13: Security List no Terraform

(a) Fonte: Próprio Autor (2023)

Security List é um componente de segurança usado para controlar o tráfego de rede em uma VCN. Ela consiste em regras de segurança que determinam quais tipos de tráfego são permitidos ou bloqueados para recursos específicos. As Security Lists fornecem controle granular sobre a conectividade da rede e podem ser associadas a diferentes sub-redes dentro da VCN.

4.5
CRIANDO A VM

Após implementar todos os códigos necessários para criação da VCN, o próximo recurso a ser criado será a VM que ficará responsável por realizar o controle do *cluster* OKE. Uma VM é uma emulação de um computador físico que permite que você execute vários sistemas operacionais e aplicativos nele. Em vez de depender de um único servidor físico dedicado, você pode criar e gerenciar várias VMs em um ambiente virtualizado.

Figura 4.14: Virtual Machine no Terraform

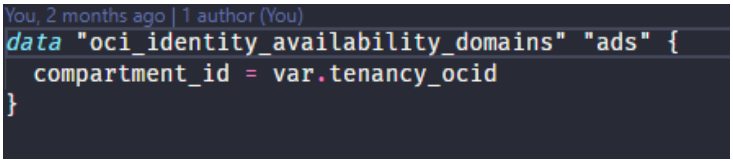
```
...
resource "oci_core_instance" "Controller" {
  availability_domain = data.oci_identity_availability_domains.ads.availability_domains[0].name
  compartment_id      = var.comapartment_ocid
  shape               = "VM.Standard2.1"
  ...
  source_details {
    source_id   = data.oci_core_images.images_controller.images[0].id
    source_type = "image"
  }

  # Optional
  display_name = "Controller"      You, 2 months ago • update pipeline …
  ...
  create_vnic_details {
    assign_public_ip = true
    subnet_id        = oci_core_subnet.subnet-cluster.id
  }

  ...
  metadata = {
    ssh_authorized_keys = file(var.ssh_public_key_file)
  }
  preserve_boot_volume = false

}
```

(a) Fonte: Próprio Autor (2023)

A imagem acima demonstra a criação da VM responsável por hospedar o Rancher que será mais uma camada de gerenciamento na nossa infraestrutura, as primeiras linhas são definições de onde essa VM ficará hospedada, tanto em qual AD e também

em qual compartment, como já existe a implementação do compartment só é necessário referenciar. Já o AD é preciso ser instanciado dessa forma:

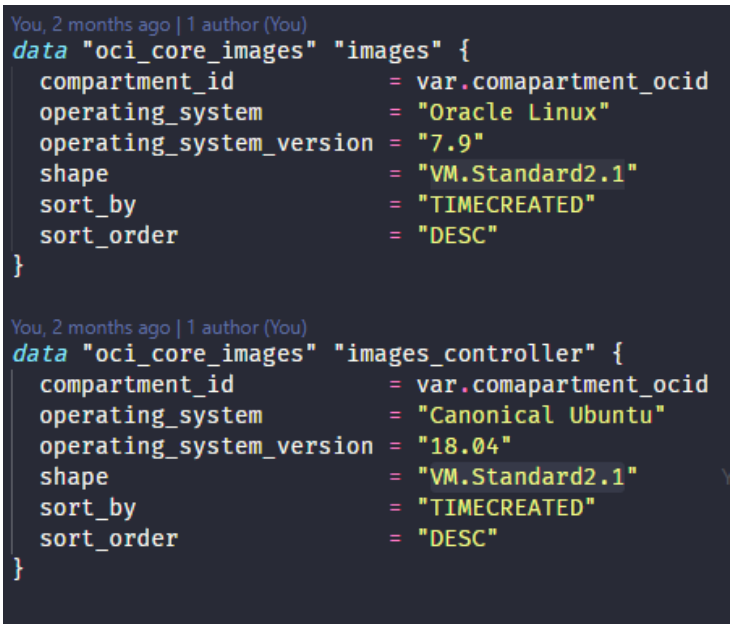
Figura 4.15: Availability Domains no Terraform



(a) Fonte: Próprio Autor (2023)

Voltando a figura 4.14, depois da definição do local dessa VM temos que o shape definido é o *standard*2 como já explicado, ele satisfaz a necessidade das ferramentas que serão utilizadas, possui 1 OCPU e 4GB de RAM. A próxima definição é de qual imagem de sistema operacional será utilizada, para isso deve-se também implementar isso como um código dessa forma:

Figura 4.16: Data Source no Terraform



(a) Fonte: Próprio Autor (2023)

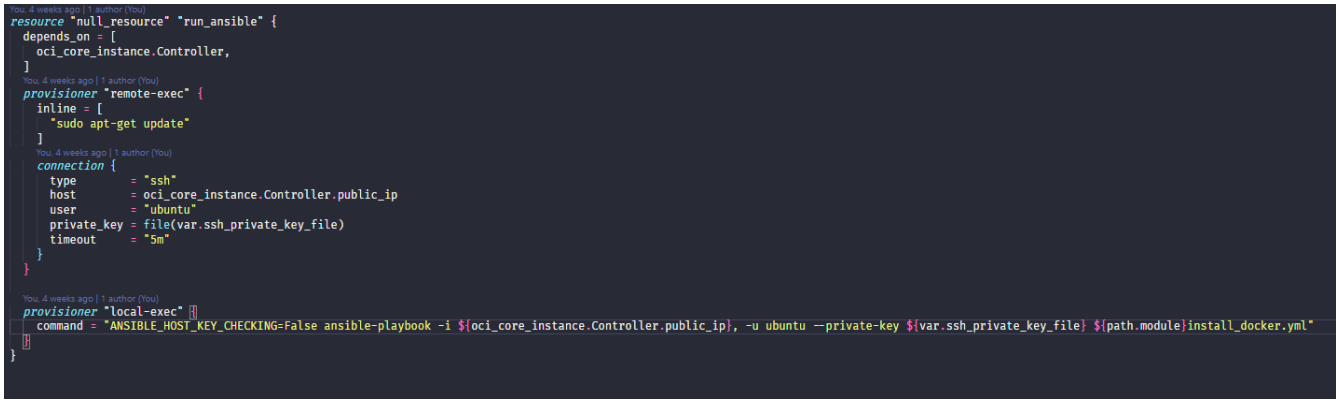
Nesse código, temos a definição de dois tipos de imagens, a primeira são as imagens para a nodepool do *cluster* OKE, ela está definida com o sistema operacional(S.O.) Oracle Linux na versão 7.9, cada node receberá uma imagem com esses requisitos. Já a segunda é a imagem que será usada na VM de gerenciamento do Rancher na infraestrutura, ela define o S.O. como o Ubuntu na versão 18.04.

Seguindo com a implementação da VM, temos a configuração da sub-rede que será adotada na VM, como já foi definido a subnet, só é necessário referenciar o ID dessa

sub-rede. Por fim, temos a definição de qual será a chave ssh da máquina virtual onde será instanciada, e é possível utilizar uma já existente (caso utilizado na imagem) ou ao criar a VM ela gera novas chaves e a última configuração é a definição de se a VM for excluída o seu disco será preservado ou não.

Para finalizar a criação da VM, é preciso instalar o Docker e o Rancher de forma automática para isso utiliza-se o Ansible juntamente com o Terraform, um novo recurso deve ser criado para executar o playbook do Ansible na VM. Abaixo segue o código do recurso no Terraform:

Figura 4.17: Run Ansible Resource Terraform



(a) Fonte: Próprio Autor (2023)

Esse recurso executa o Playbook do Ansible abaixo na máquina virtual via SSH:

Figura 4.18: Ansible playbook - instalar Docker e Rancher

```
- host: all
  become: yes

  task:
    #install docker
    - name: install docker
      apt:
        name: docker.io
        state: present

    - name: run a docker container rancher
      #run a command in bash
      shell: docker run -d --restart=unless-stopped -p 8080:80 -p 8443:443 rancher/rancher
      docker logs  container-id  2>&1 | grep "Bootstrap Password:"

    - name: wait for rancher to be up
      wait_for:
        host: localhost
        port: 8080
        delay: 10
        timeout: 300

    - name: get a pswd from rancher
      shell: docker exec rancher cat /var/lib/rancher/k3s/server/node-token
      register: node_token
```

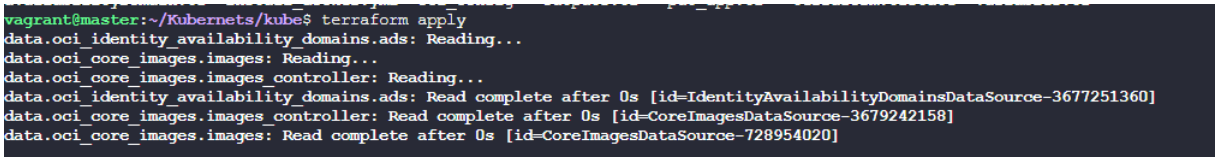
(a) Fonte: Próprio Autor (2023)

Ao executar esses playbooks do Ansible, todo o processo de instalação do Rancher e do Docker é automatizado, garantindo que todas as etapas sejam realizadas corretamente e de forma consistente em todos os servidores ou máquinas virtuais em que você deseja implantar essas aplicações.

Os principais motivos de se utilizar o Rancher é a sua facilidade de uso e interface intuitiva. Com uma interface gráfica amigável e recursos de gerenciamento simplificados, o Rancher torna mais acessível à implantação, configuração e monitoramento de *clusters*s Kubernetes, mesmo para equipes sem experiência prévia na tecnologia, a interface gráfica disponibilizada para o usuário permite que esse processo de gerenciamento seja intuitivo.

Com a estrutura codificada, é necessário executar o comando “terraform apply” para que o código seja transformado em uma infraestrutura na nuvem:

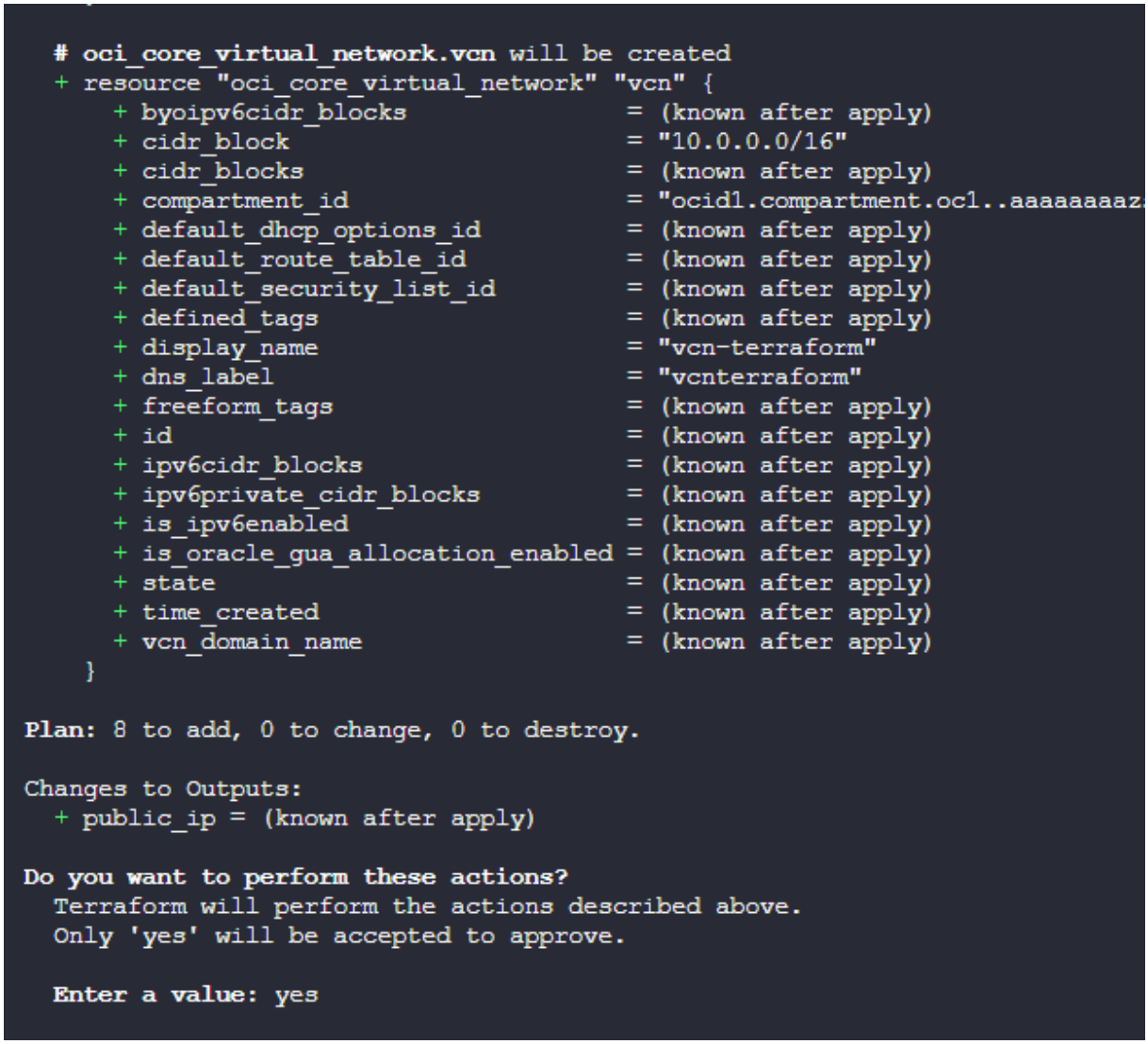
Figura 4.19: Terraform apply



(a) Fonte: Próprio Autor (2023)

Após a execução desse comando, são exibidos todos os recursos que serão criados na nuvem, nessa etapa a análise do que será criado é crucial, se tudo estiver correto basta confirmar e todos os recursos listados serão implementados:

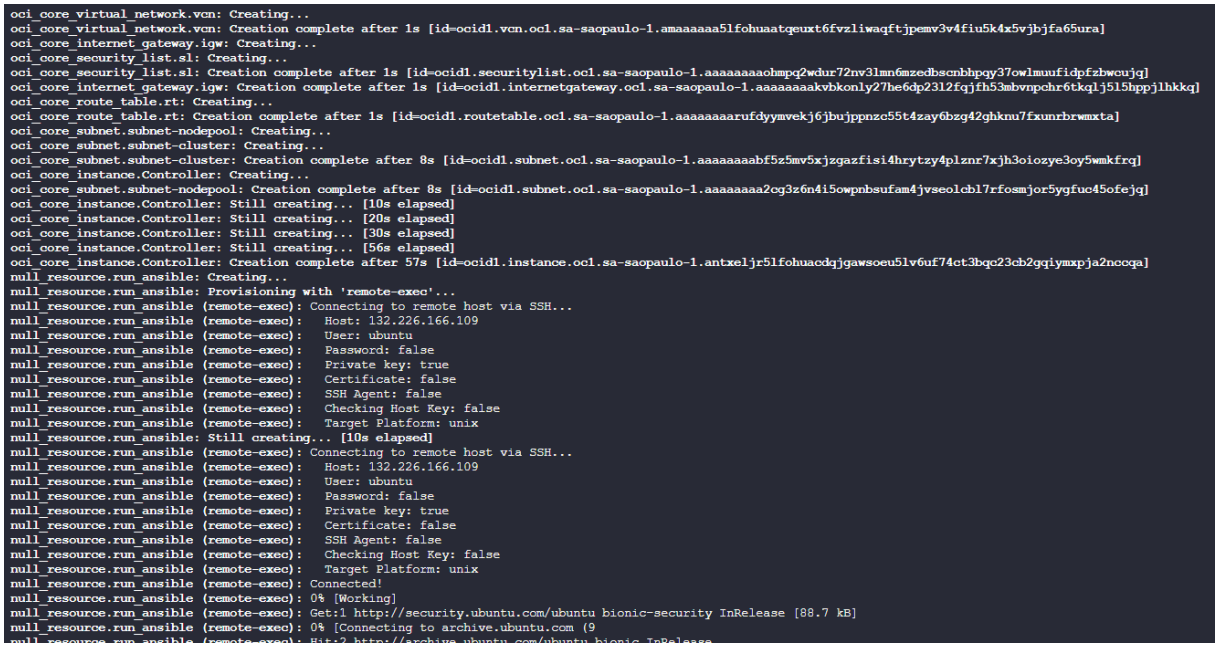
Figura 4.20: Confirmar o Terraform apply



(a) Fonte: Próprio Autor (2023)

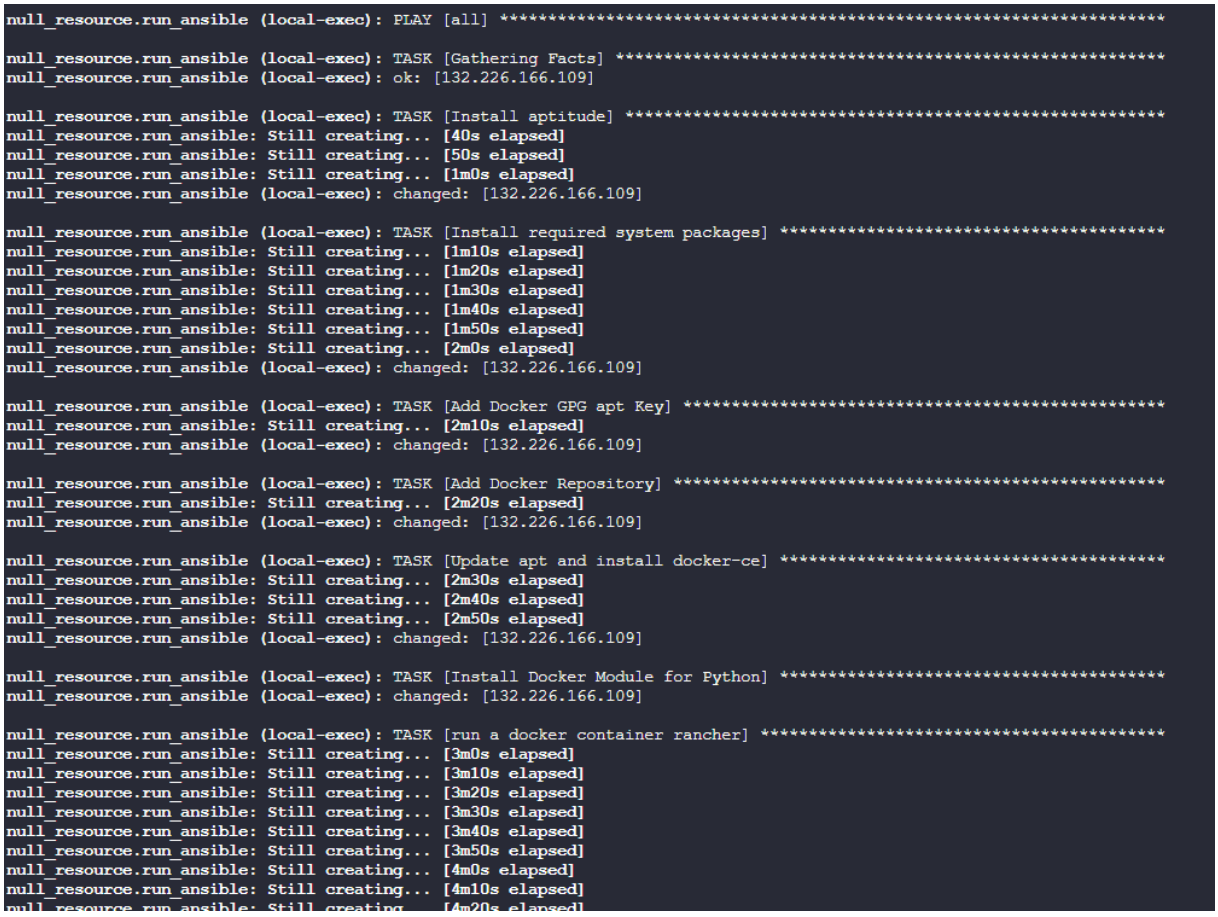
Abaixo, segue como o Terraform se comporta enquanto cria os recursos e, logo em seguida, quando está executando o ansible:

Figura 4.21: Criando recursos Terraform



(a) Fonte: Próprio Autor (2023)

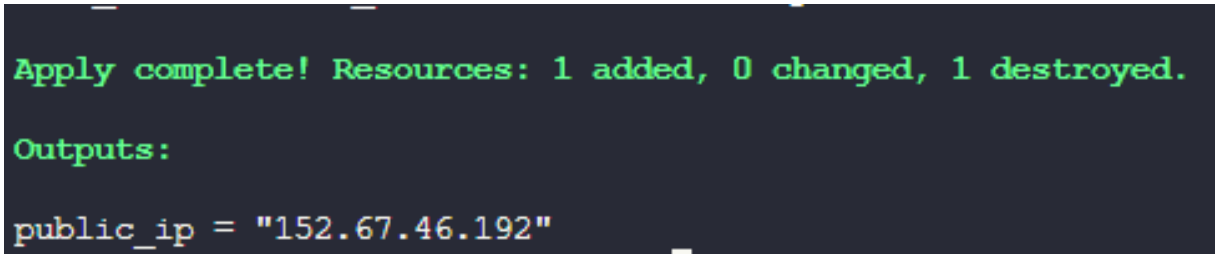
Figura 4.22: Rodando Ansible playbook



(a) Fonte: Próprio Autor (2023)

Se tudo estiver correto o final será mostrado uma mensagem de confirmação e um IP publico que será importante para poder acessar o rancher:

Figura 4.23: Sucesso ao implantar os recursos



(a) Fonte: Próprio Autor (2023)

4.6 IMPLEMENTANDO GITOPS NO REPOSITÓRIO

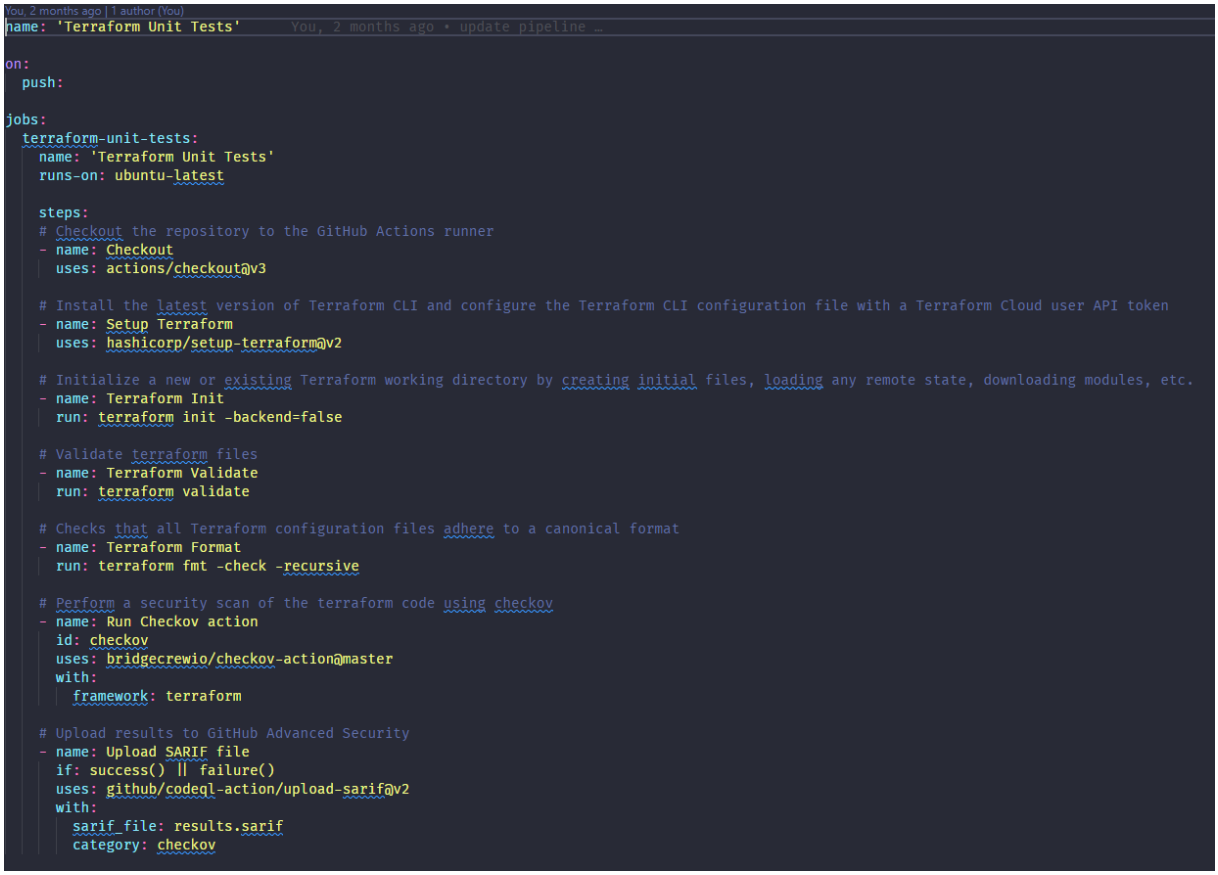
A implementação do *pipeline* de CI/CD no repositório de infraestrutura, seguindo os conceitos do GitOps, é uma abordagem altamente eficiente e automatizada para gerenciar a infraestrutura como código. Com o GitOps, todo o processo de implantação e gerenciamento da infraestrutura é controlado por meio de um repositório Git centralizado. O *pipeline* de CI/CD desse repositório permite que as mudanças na infraestrutura sejam rastreadas, revisadas e testadas de forma consistente e confiável. Foi utilizada a funcionalidade de actions do Github para toda implantação da esteira CI/CD e o codigo encontra-se e no repositório de Kaique (RIERICKSON, 2023).

Ao adotar o GitOps, as alterações na infraestrutura são realizadas por meio de push (adição de modificações) no repositório. O *pipeline* de CI/CD é responsável por compilar, validar e implantar essas alterações automaticamente em um ambiente de teste ou produção. Ele garante que as alterações propostas sigam as melhores práticas, estejam conforme as políticas definidas e não causem impactos indesejados no ambiente.

Dentro dessas boas práticas, podem ser implementados testes unitários no código criado para definição da infraestrutura visando aumentar a qualidade e confiabilidade do ambiente. Os testes unitários são uma parte essencial do desenvolvimento de software, incluindo a criação e definição da infraestrutura. Eles ajudam a identificar possíveis erros e garantir que o código funcione corretamente.

Segue abaixo a configuração de testes unitários automatizados que dizem respeito a parte de CI do repositório:

Figura 4.24: Testes unitários do repositório

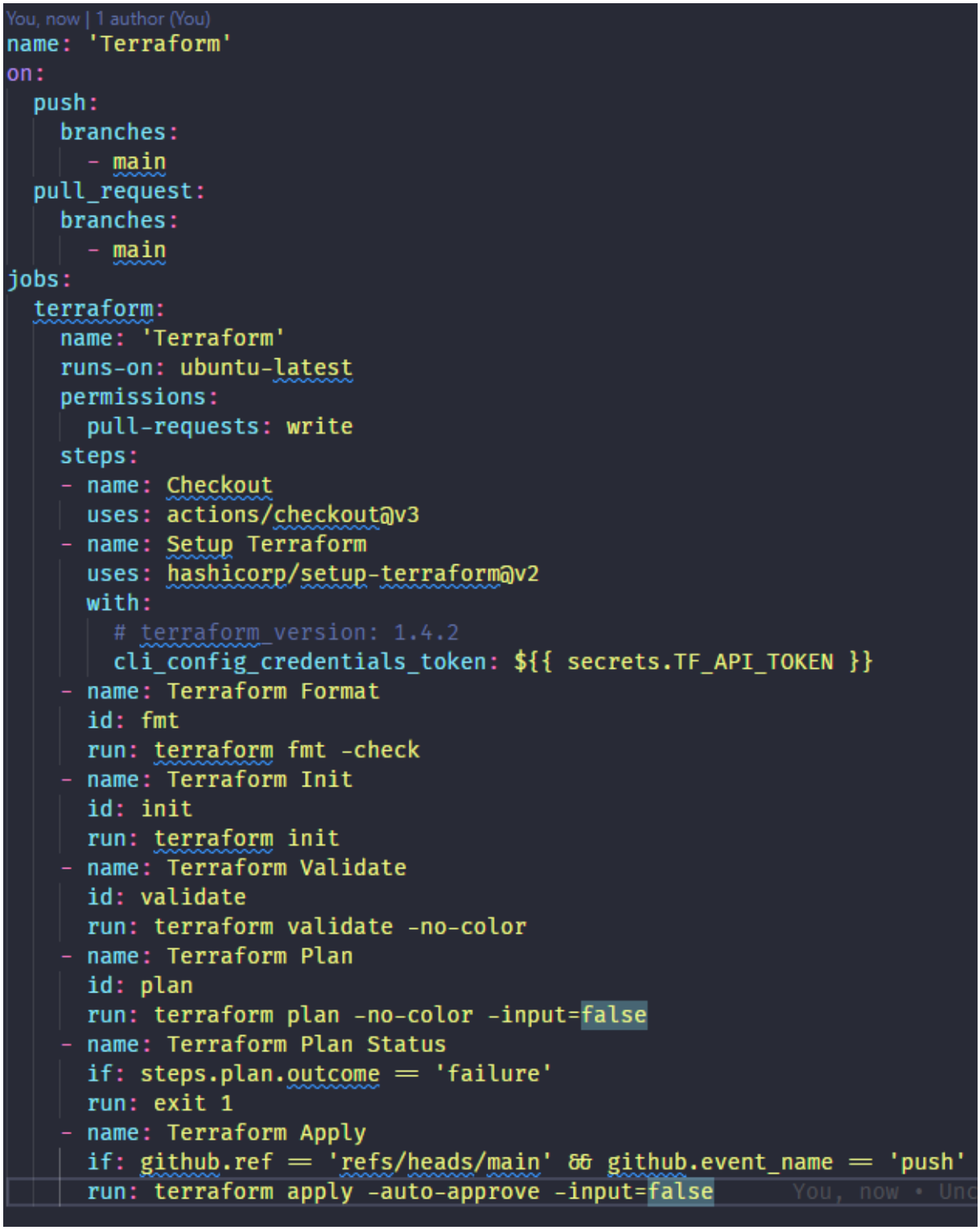


```yaml
You, 2 months ago | 1 author (You)
name: 'Terraform Unit Tests'      You, 2 months ago • update pipeline …

on:
  push:

jobs:
  terraform-unit-tests:
    name: 'Terraform Unit Tests'
    runs-on: ubuntu-latest

    steps:
    # Checkout the repository to the GitHub Actions runner
    - name: Checkout
      uses: actions/checkout@v3

    # Install the latest version of Terraform CLI and configure the Terraform CLI configuration file with a Terraform Cloud user API token
    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v2

    # Initialize a new or existing Terraform working directory by creating initial files, loading any remote state, downloading modules, etc.
    - name: Terraform Init
      run: terraform init -backend=false

    # Validate terraform files
    - name: Terraform Validate
      run: terraform validate

    # Checks that all Terraform configuration files adhere to a canonical format
    - name: Terraform Format
      run: terraform fmt -check -recursive

    # Perform a security scan of the terraform code using checkov
    - name: Run Checkov action
      id: checkov
      uses: bridgecrewio/checkov-action@master
      with:
        framework: terraform

    # Upload results to GitHub Advanced Security
    - name: Upload SARIF file
      if: success() || failure()
      uses: github/codeql-action/upload-sarif@v2
      with:
        sarif_file: results.sarif
        category: checkov
```

(a) Fonte: Próprio Autor (2023)

Depois que os testes são executados, o fluxo entra na parte de CD com a implementação da infraestrutura através desse yml abaixo:

Figura 4.25: Deploy do código no repositório



```
You, now | 1 author (You)
name: 'Terraform'
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
jobs:
  terraform:
    name: 'Terraform'
    runs-on: ubuntu-latest
    permissions:
      pull-requests: write
    steps:
    - name: Checkout
      uses: actions/checkout@v3
    - name: Setup Terraform
      uses: hashicorp/setup-terraform@v2
      with:
        # terraform_version: 1.4.2
        cli_config_credentials_token: ${{ secrets.TF_API_TOKEN }}
    - name: Terraform Format
      id: fmt
      run: terraform fmt -check
    - name: Terraform Init
      id: init
      run: terraform init
    - name: Terraform Validate
      id: validate
      run: terraform validate -no-color
    - name: Terraform Plan
      id: plan
      run: terraform plan -no-color -input=false
    - name: Terraform Plan Status
      if: steps.plan.outcome == 'failure'
      run: exit 1
    - name: Terraform Apply
      if: github.ref == 'refs/heads/main' && github.event_name == 'push'
      run: terraform apply -auto-approve -input=false      You, now • Unc
```

(a) Fonte: Próprio Autor (2023)

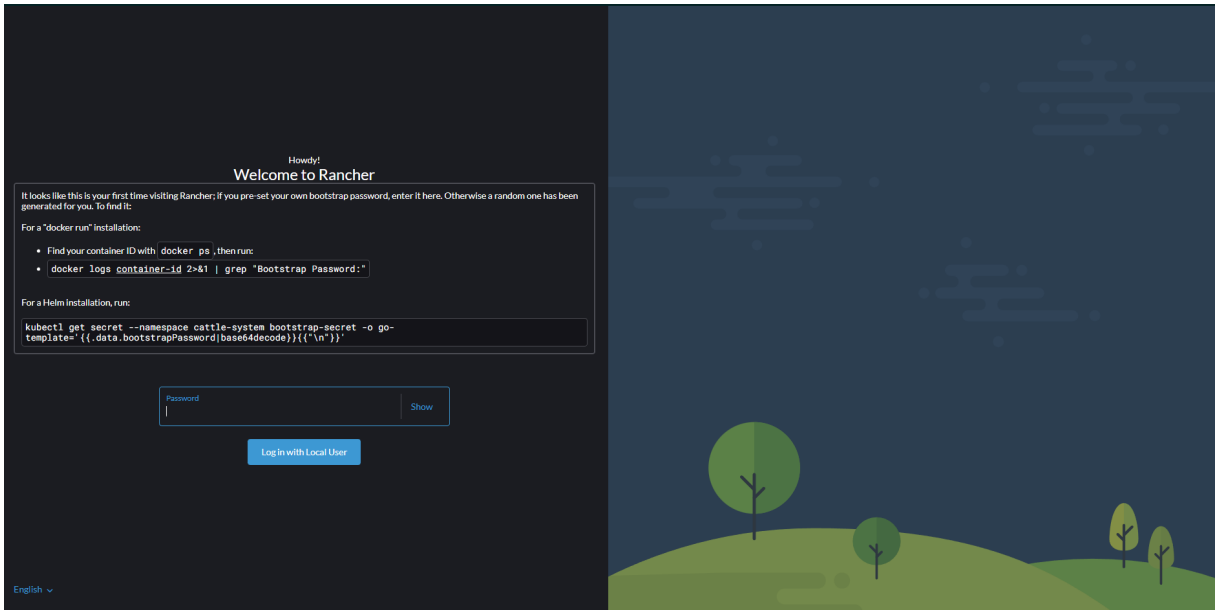
Com a implementação do *pipeline* de CI/CD no repositório de infraestrutura, seguindo os conceitos do GitOps, a organização pode obter maior eficiência operacional, maior rastreabilidade e uma infraestrutura mais confiável. Essa abordagem permite

uma gestão de mudanças simplificada e automatizada, reduzindo os riscos de falhas e melhorando a velocidade de entrega das alterações na infraestrutura.

4.7
IMPLEMENTAÇÃO E GERENCIAMENTO DO CLUSTER

Depois da criação da infraestrutura, é disponibilizado um IP, com esse IP é possível acessar o Rancher de qualquer lugar. Ao acessar o IP a tela que aparecerá é essa:

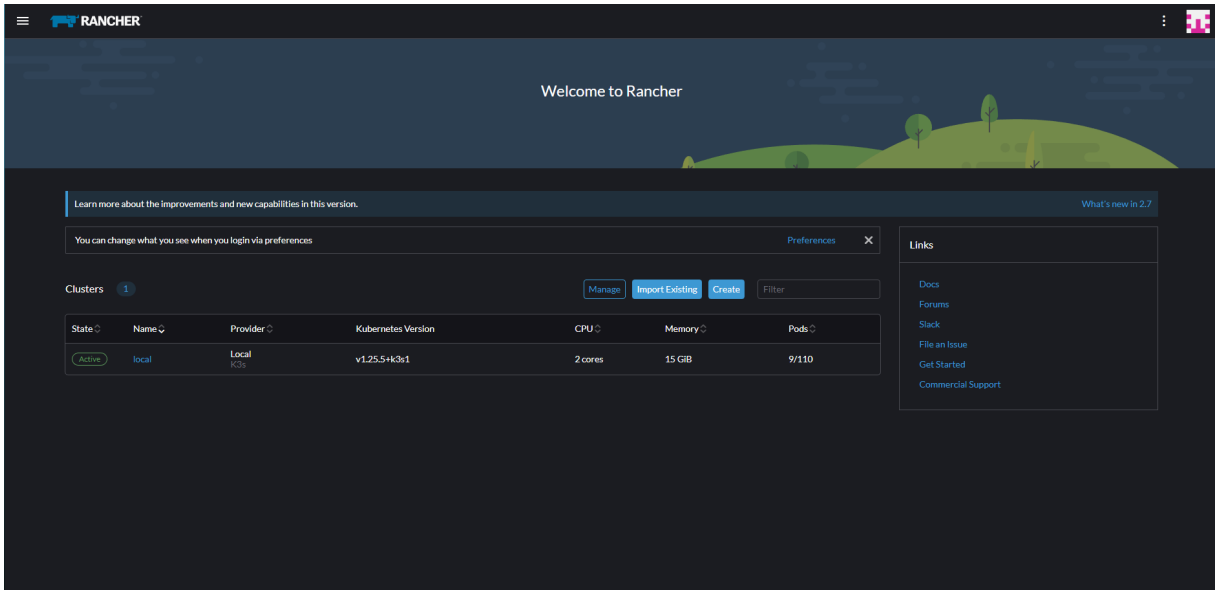
Figura 4.26: Primeira tela Rancher



(a) Fonte: Próprio Autor (2023)

O próprio Rancher disponibiliza um passo a passo para a senha ser adquirida pelo usuário. Para funcionar corretamente esse comando dever ser executado na máquina do Rancher e para poder acessar a máquina criada é necessário executar esse comando “ssh ubuntu@[id da máquina]”. Em seguida pode ser executado esse passo que o Rancher mostra.

Figura 4.27: Home Page Rancher



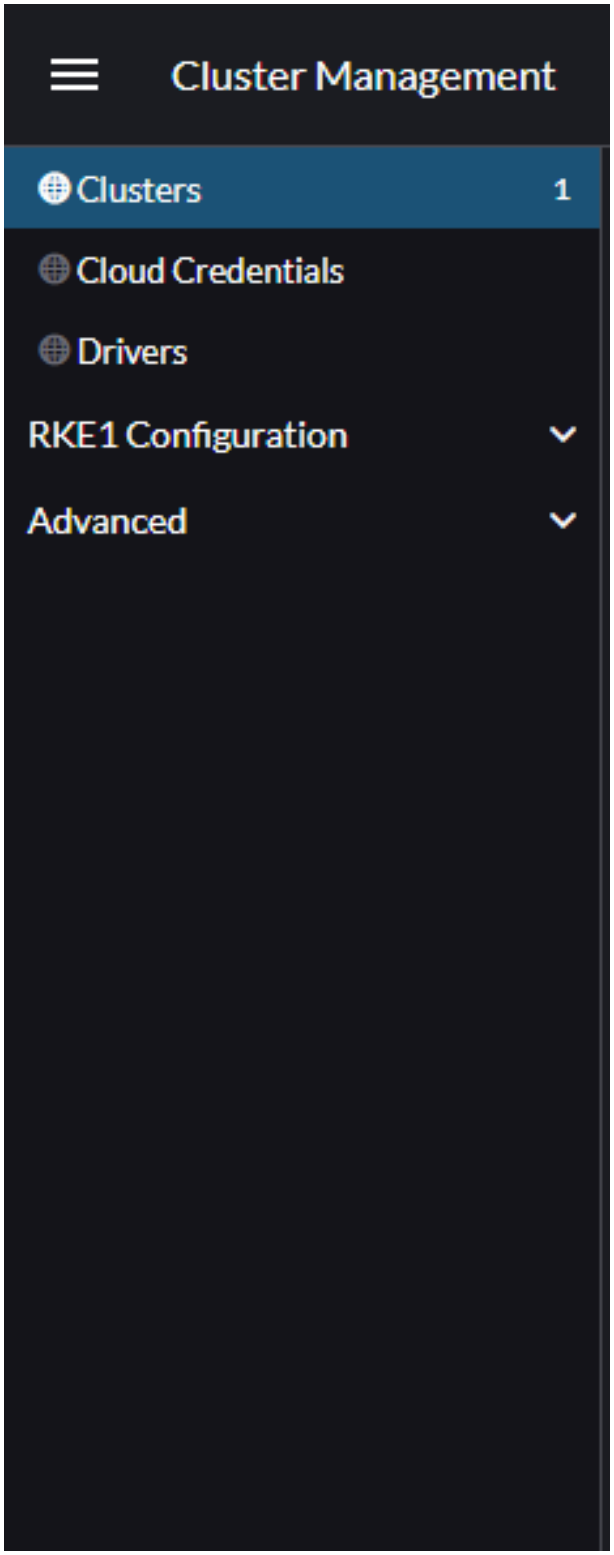
(a) Fonte: Próprio Autor (2023)

Ao inserir a senha corretamente, a página principal do Rancher será exibida.

A ideia inicial do projeto seria realizar a criação do *cluster* OKE através da infraestrutura como um código e utilizar o Rancher para gerenciar esse *cluster*. Porém, temos um problema em utilizar a OCI, como ela é uma cloud recente algumas funcionalidades ainda não estão disponíveis para ela, uma dessas é a importação de *cluster* já existentes para o Rancher de forma automática, por isso se faz necessário a criação manual do *cluster* OKE pela interface do Rancher, facilitando todo o processo para essa criação, a seguir são definidos os conjuntos de passos para criar o *cluster* na OCI:

Primeiro precisa-se habilitar os drivers que são correspondentes a Oracle Cloud, como é uma nuvem mais recente esses drivers vem por padrão desabilitado, porém, é bem simples de ativar.

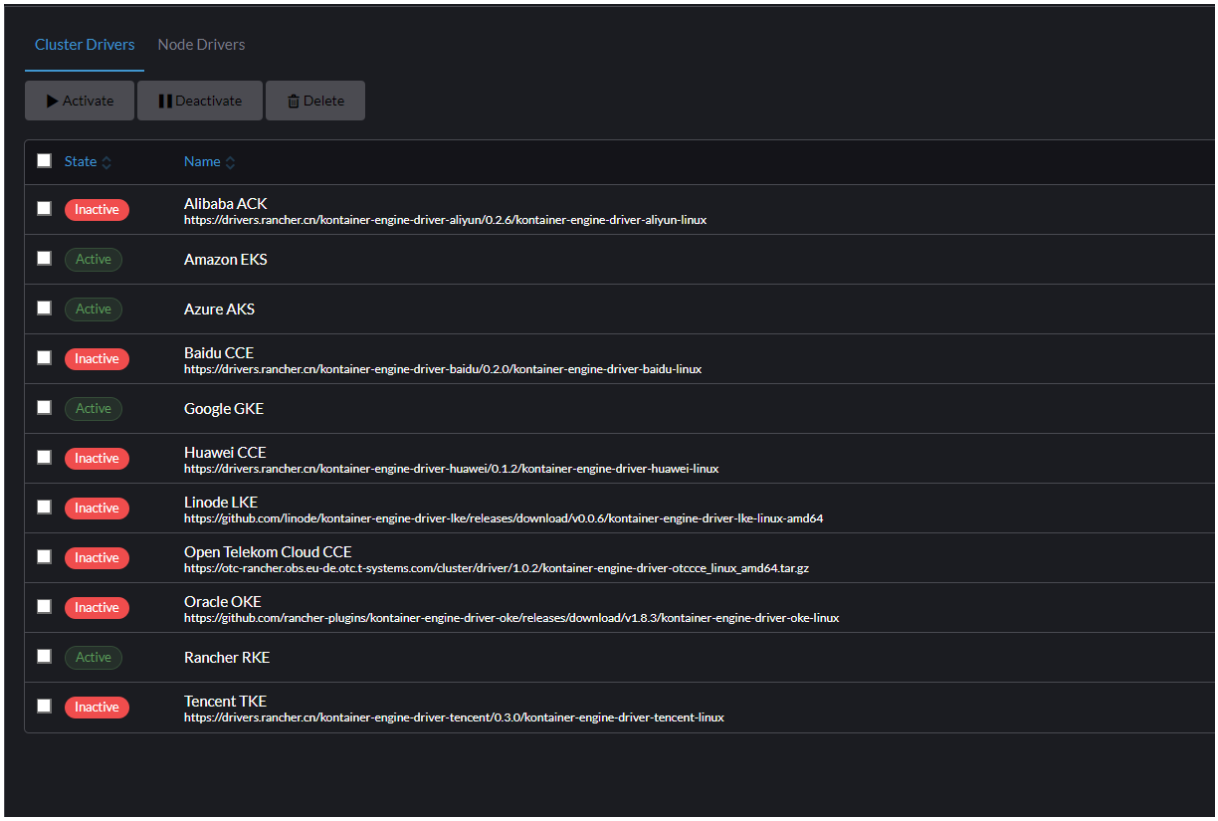
Figura 4.28: Selecionar Driver Rancher



(a) Fonte: Próprio Autor (2023)

Para ativar é preciso acessar as pagina de driver, clicando no nome Drivers:

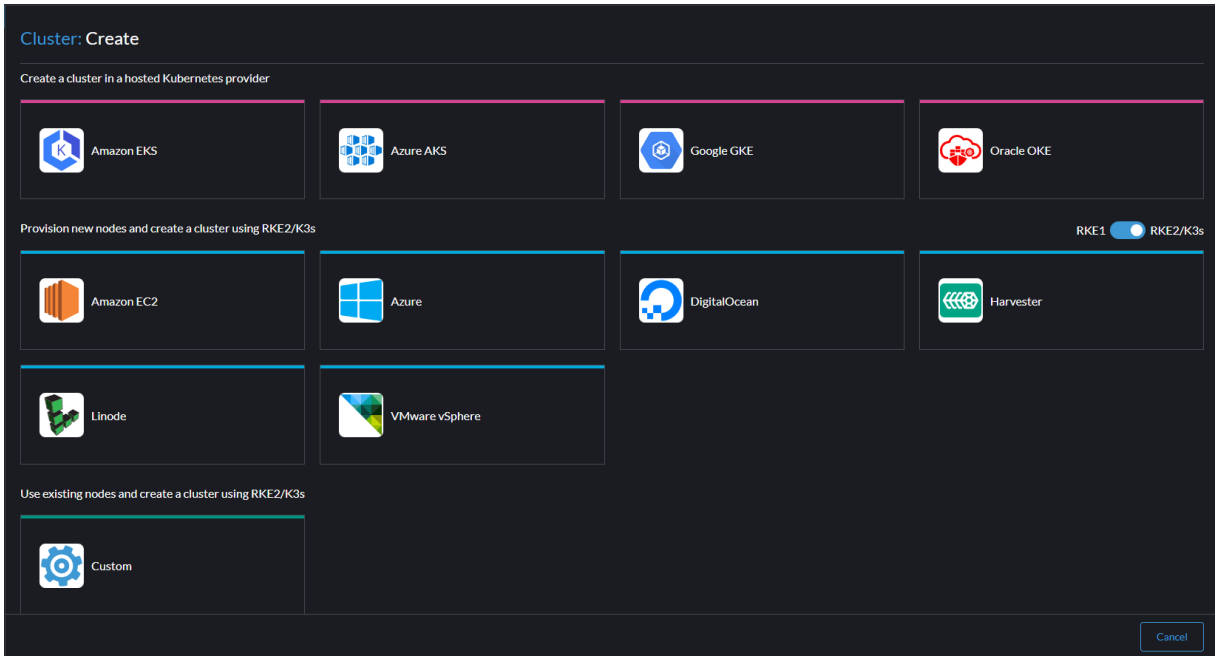
Figura 4.29: Drivers Menu Rancher



(a) Fonte: Próprio Autor (2023)

Após localizar os drivers que correspondem à especificação OCI, o processo de ativação desses drivers é bastante simples no Rancher. Basta clicar nos drivers desejados e, em seguida, no botão de ativar para habilitá-los. Essa mesma operação pode ser repetida na aba de node para ativar os drivers nos nós do *cluster*.

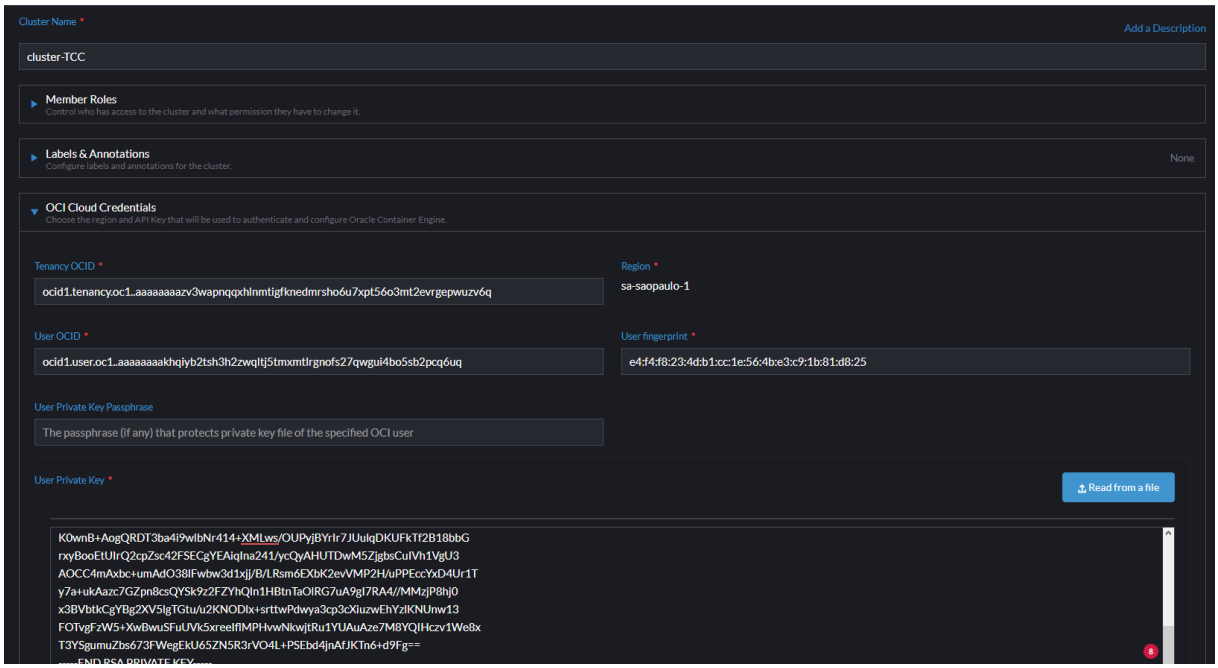
Figura 4.30: Menu de criação Rancher



(a) Fonte: Próprio Autor (2023)

Aqui é onde escolhemos qual o provedor que irá hospedar o *cluster*, no caso como o projeto esta sendo desenvolvido na OCI, precisa que a opção do OKE seja selecionado.

Figura 4.31: Configurando Acesso da OCI no Rancher

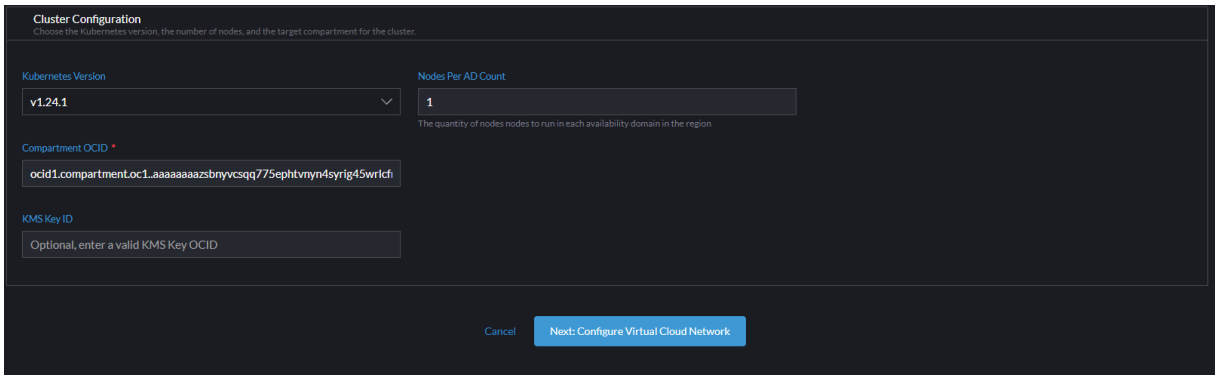


(a) Fonte: Próprio Autor (2023)

Nessa etapa você deve preencher os campos com as informações pedidas de acordo com sua cloud, cada conta terá sua própria Tenancy, próprio fingerprint e afins.

Um adendo: a private key gerada pela a OCI vem um formato no qual o Rancher não aceita, mas para contornar esse problema é bem simples basta executar esse comando “openssl rsa -in [nome-chave].pem -out [nova-chave].pem” e após ser gerada essa nova chave, basta colar ela no seu campo.

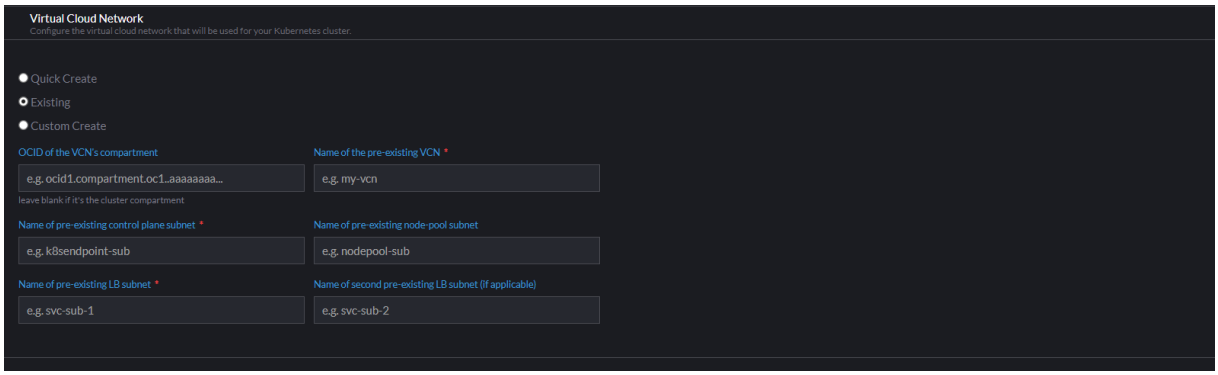
Figura 4.32: Configuração do cluster no Rancher



(a) Fonte: Próprio Autor (2023)

O próximo passo é configurar os requisitos específicos do *cluster* no Rancher. Isso inclui definir a versão do Kubernetes que será implantada, especificar em qual compartment (compartimento) o *cluster* estará localizado e determinar em quantas Availability Domains (ADs) o *cluster* será hospedado.

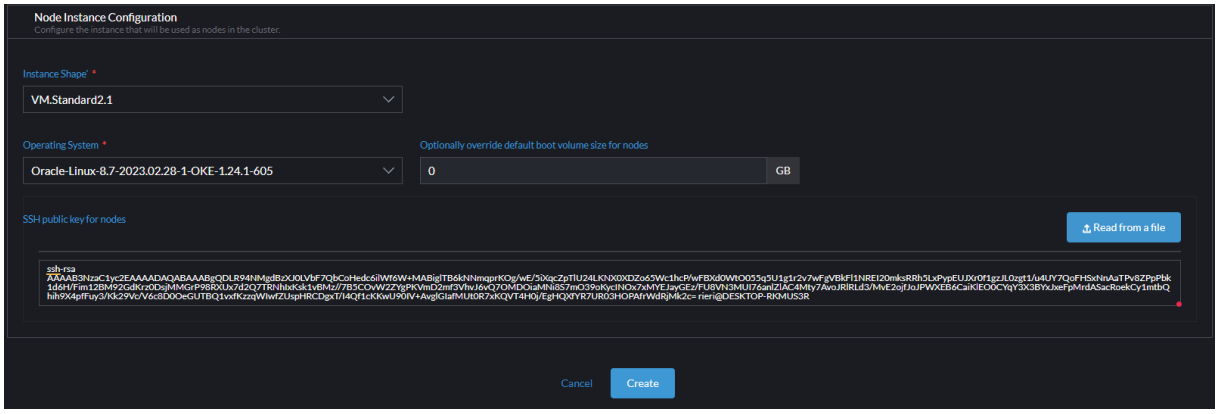
Figura 4.33: Configuração da VCN no Rancher



(a) Fonte: Próprio Autor (2023)

Após a criação da Virtual Cloud Network (VCN), o processo de configuração no Rancher é simplificado. Você pode selecionar a opção de utilizar uma VCN já existente, evitando a necessidade de criar uma nova do zero. Isso proporciona conveniência e agilidade, pois você pode aproveitar uma VCN previamente configurada e adaptada às necessidades da organização.

Figura 4.34: Configuração dos nodes no Rancher



(a) Fonte: Próprio Autor (2023)

Por fim, é preciso definir os requisitos das máquinas que estarão dentro da node-pool, é preciso se ater as descrições que o Rancher mostra, algumas máquinas pode não ser compatíveis com alguma versão do OKE.

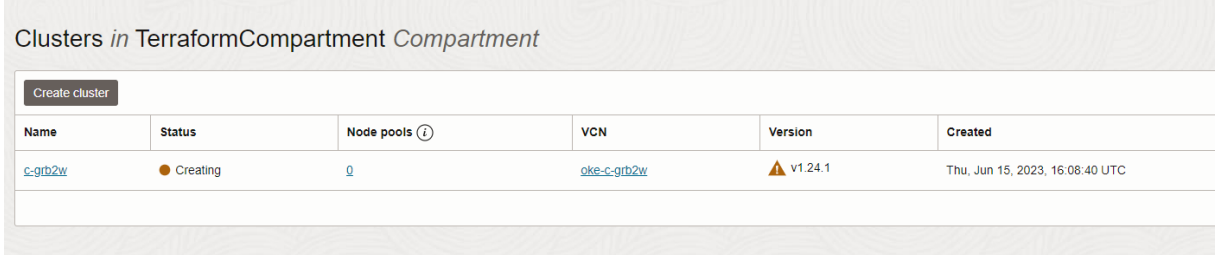
Se todas as configurações estiverem corretas, depois de um tempo irá aparecer o *cluster* criado na sua dashboard:

Figura 4.35: Dashboard do Rancher (pós-criação do cluster)

(a) Fonte: Próprio Autor (2023)

E na conta OCI, deve aparecer o *cluster* sendo provisionado:

Figura 4.36: Dashboard de cluster no OCI

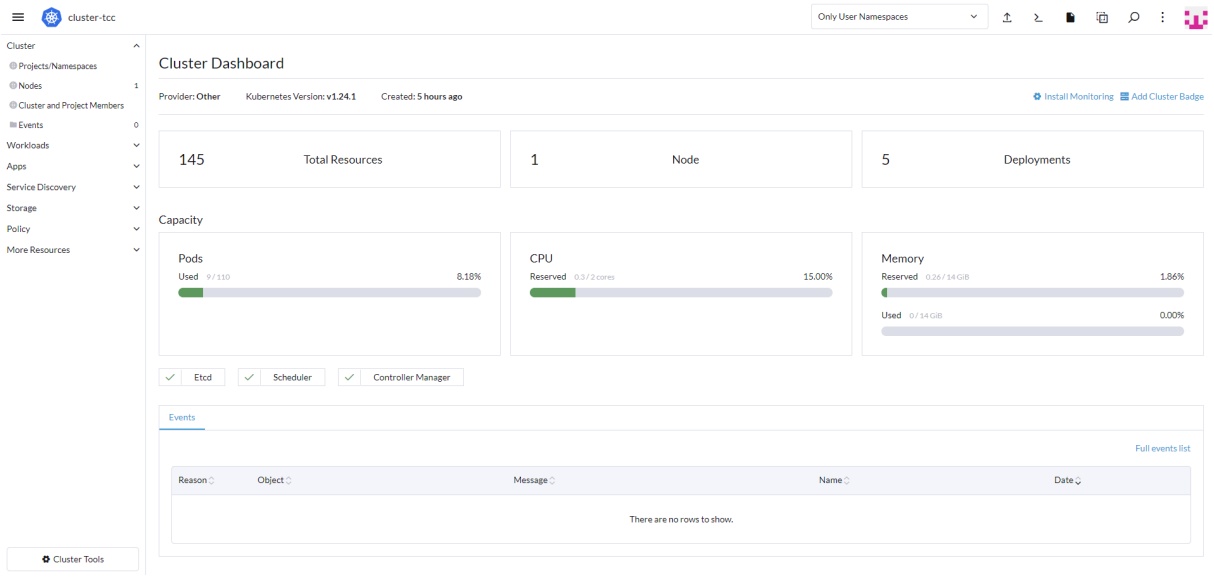


(a) Fonte: Próprio Autor (2023)

Em termos de monitoramento, o Rancher permite visualizar e acompanhar o desempenho dos seus *clusterss* em tempo real. Você pode monitorar métricas essenci-

ais, como utilização de CPU, memória e rede, para garantir que os recursos estejam sendo utilizados eficientemente. Além disso, o Rancher fornece alertas configuráveis que notificam quando algum limite predefinido é ultrapassado, permitindo uma resposta rápida a problemas de desempenho ou disponibilidade.

Figura 4.37: Dashboard do cluster no Rancher



(a) Fonte: Próprio Autor (2023)

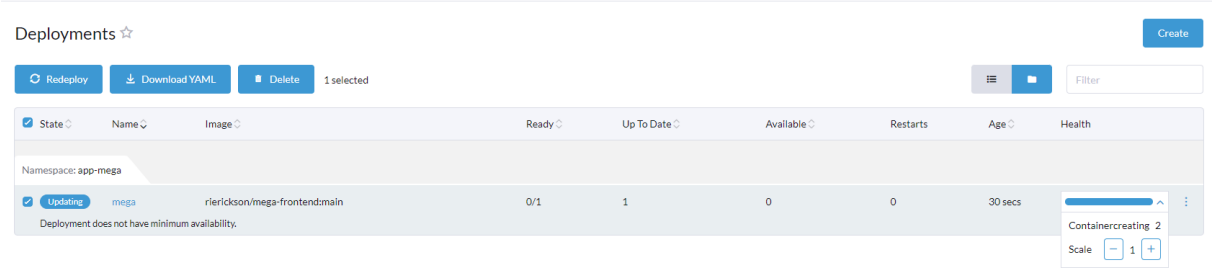
Figura 4.38: Logs do cluster no Rancher

Reason	Object	Message	Name	Date
Pulled	Pod mega-7b686bb86c-k46z8	Successfully pulled image "rierickson/mega-frontend:main" in 1.495084363s	mega-7b686bb86c-k46z8.1768f2cbb644390d	Thu, Jun 15 2023 6:36:57 pm
Created	Pod mega-7b686bb86c-k46z8	Created container container-mega	mega-7b686bb86c-k46z8.1768f2cbc693f1cb	Thu, Jun 15 2023 6:36:57 pm
Started	Pod mega-7b686bb86c-k46z8	Started container container-mega	mega-7b686bb86c-k46z8.1768f2cbc998fe02	Thu, Jun 15 2023 6:36:57 pm
Killing	Pod mega-844f44c5-ptgjd	Stopping container container-mega	mega-844f44c5-ptgjd.1768f2cbd8ddee02	Thu, Jun 15 2023 6:36:57 pm
SuccessfulDelete	ReplicaSet mega-844f44c5	Deleted pod: mega-844f44c5-ptgjd	mega-844f44c5.1768f2cbd8e0426b	Thu, Jun 15 2023 6:36:57 pm
ScalingReplicaSet	Deployment mega	Scaled down replica set mega-844f44c5 to 0	mega.1768f2cbd7c775ff	Thu, Jun 15 2023 6:36:57 pm
Pulling	Pod mega-7b686bb86c-k46z8	Pulling image "rierickson/mega-frontend:main"	mega-7b686bb86c-k46z8.1768f2cb5d266553	Thu, Jun 15 2023 6:36:55 pm
Scheduled	Pod mega-7b686bb86c-k46z8	Successfully assigned app-mega/mega-7b686bb86c-k46z8 to 10.0.10.28	mega-7b686bb86c-k46z8.1768f2cb34f52880	Thu, Jun 15 2023 6:36:55 pm
SuccessfulCreate	ReplicaSet mega-7b686bb86c	Created pod: mega-7b686bb86c-k46z8	mega-7b686bb86c.1768f2cb345da183	Thu, Jun 15 2023 6:36:55 pm
ScalingReplicaSet	Deployment mega	Scaled up replica set mega-7b686bb86c to 1	mega.1768f2cb2d11472b	Thu, Jun 15 2023 6:36:54 pm

(a) Fonte: Próprio Autor (2023)

No gerenciamento de *clusters*s, o Rancher simplifica tarefas complexas, como escalonamento, balanceamento de carga e alta disponibilidade. Você pode facilmente adicionar ou remover nós em seus *clusters*s, dimensionando a infraestrutura conforme as necessidades do seu aplicativo. O Rancher também oferece recursos avançados de gerenciamento de configuração, permitindo a definição e aplicação de políticas consistentes em todos os *clusters*s.

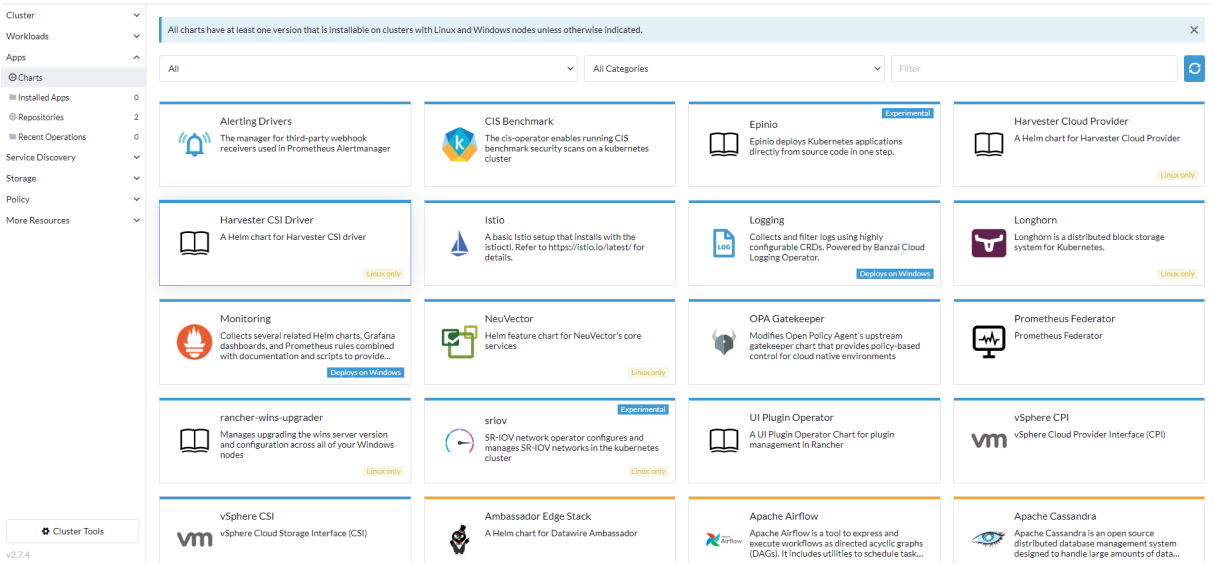
Figura 4.39: Escalando Deploy no Rancher



(a) Fonte: Próprio Autor (2023)

Com o Rancher, você pode aproveitar um catálogo de aplicativos pré-construídos e prontos para uso. Esses aplicativos são chamados de ”Charts”. O catálogo do Rancher oferece uma ampla seleção de Charts que abrangem desde aplicativos populares, como bancos de dados e servidores web, até ferramentas de monitoramento e logging.

Figura 4.40: Lista de Charts Rancher



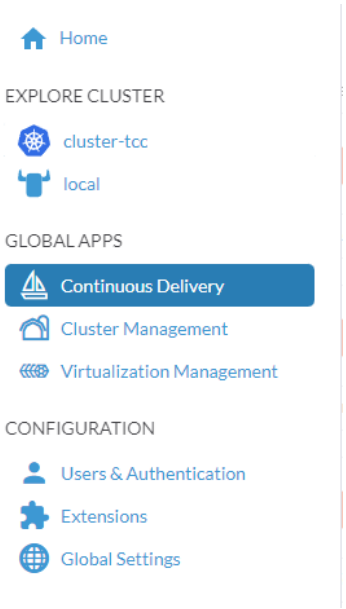
(a) Fonte: Próprio Autor (2023)

4.8

IMPLEMENTANDO O FLEET

O Rancher Fleet é uma ferramenta poderosa para gerenciamento de múltiplos *clusterss* Kubernetes de forma centralizada. Com o Fleet, pode-se automatizar a implantação de deploys em múltiplos *clusterss* Kubernetes em diferentes provedores de nuvem ou regiões, a partir de apenas um repositório Git, a equipe de DevOps tem um ganho gigantesco de esforço com a utilização do Fleet.

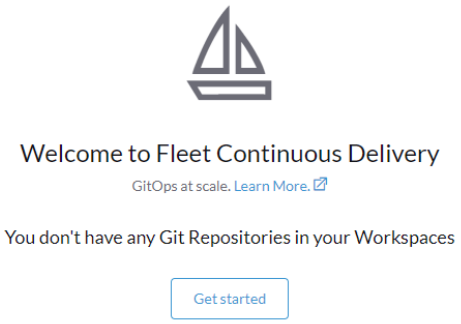
Figura 4.41: Acessar o Fleet Rancher



(a) Fonte: Próprio Autor (2023)

Para acessar o Fleet é necessário entrar na parte de Continous Delivery no Rancher.

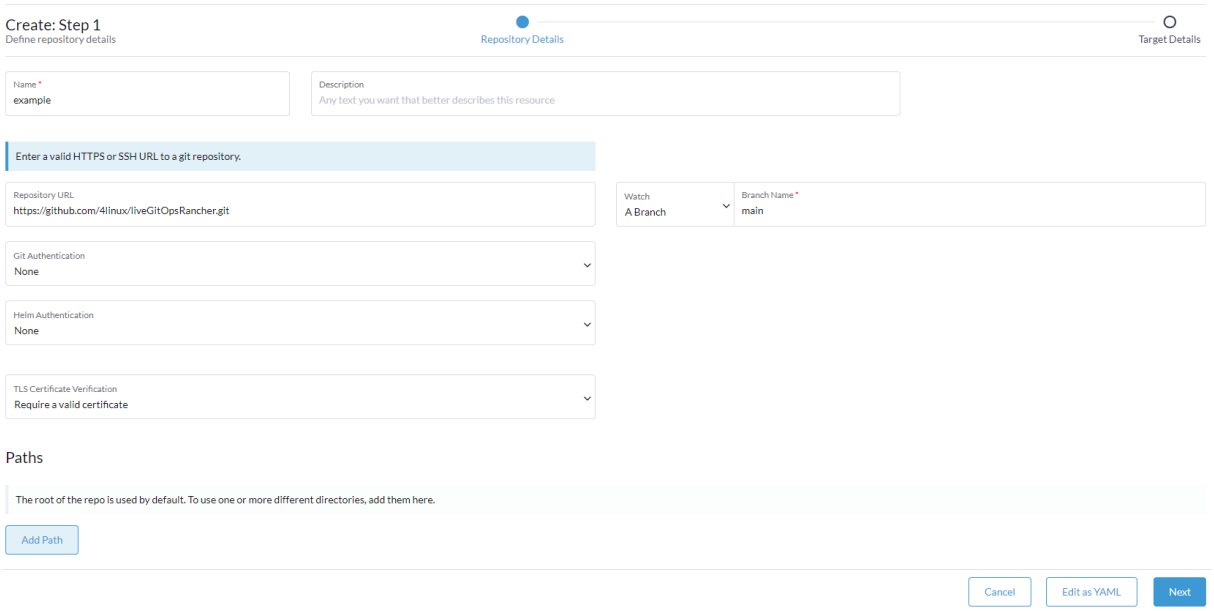
Figura 4.42: Dashboard do Fleet Rancher



(a) Fonte: Próprio Autor (2023)

A primeira vez ao acessar o Fleet, é exibida uma mensagem como essa, é necessário fazer a criação desse serviço.

Figura 4.43: Configurando o repositório do Fleet Rancher

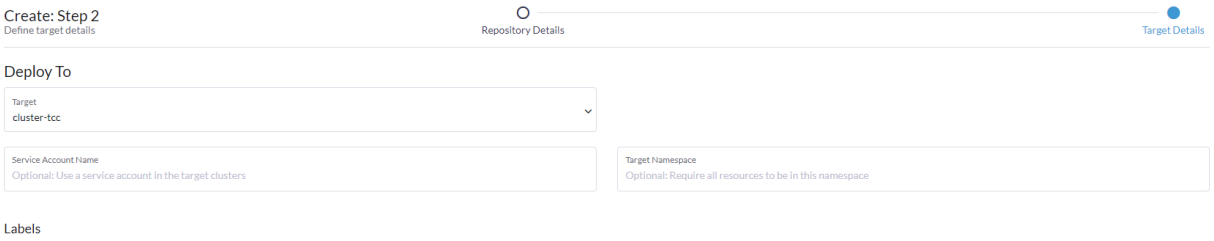


(a) Fonte: Próprio Autor (2023)

A primeira etapa na configuração do Rancher Fleet envolve a definição do nome e descrição do Fleet, bem como a especificação do repositório que o Fleet irá monitorar. Nesta etapa, é necessário fornecer o link para o repositório desejado e também definir qual branch desse repositório será observada pelo Fleet.

O repositório pode estar hospedado em um sistema de controle de versão, como o Git, e pode incluir arquivos YAML contendo definições de políticas, aplicativos e outras configurações relacionadas aos *clusterss* Kubernetes.

Figura 4.44: Configurando alvo do Fleet Rancher

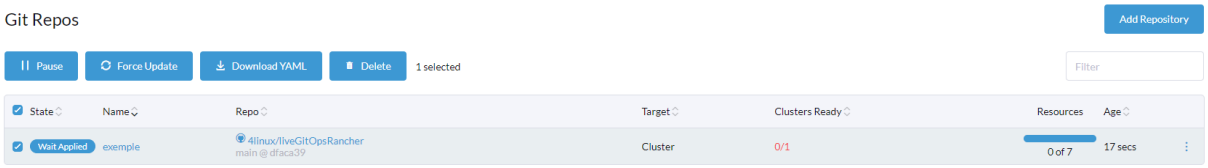


(a) Fonte: Próprio Autor (2023)

A segunda etapa na implementação do Rancher Fleet envolve a definição do *cluster* no qual o Fleet irá replicar os recursos presentes no repositório definido anteriormente. Essa etapa é fundamental para garantir que os recursos, como configurações de políticas, aplicativos e configurações personalizadas, sejam implantados corretamente nos *clusterss* específicos.

Ao configurar o Rancher Fleet, é possível especificar quais *clusters*s receberão as implantações e atualizações dos recursos definidos no repositório. Essa flexibilidade permite que você direcione diferentes recursos para *clusters*s específicos, considerando requisitos específicos do aplicativo, ambientes de produção ou requisitos de segurança.

Figura 4.45: Aplicando recurso do Fleet Rancher



(a) Fonte: Próprio Autor (2023)

Uma vez que os *clusters*s estejam configurados no Rancher Fleet, você pode começar a aproveitar os benefícios de gerenciamento centralizado. Isso inclui a capacidade de implantar aplicativos em escala, monitorar o desempenho dos *clusters*s, gerenciar atualizações e rollbacks, e aplicar políticas de segurança e governança em todos os *clusters*s.

Ao utilizar o Rancher Fleet, pode simplificar o gerenciamento e a operação de múltiplos *clusters*s Kubernetes, aumentando a eficiência e a escalabilidade dos seus ambientes de contêineres. É importante ressaltar que a implementação do Fleet deve ser cuidadosamente planejada e executada, levando em consideração as necessidades específicas da organização e seguindo as práticas recomendadas pelo Rancher.

5

RESULTADOS

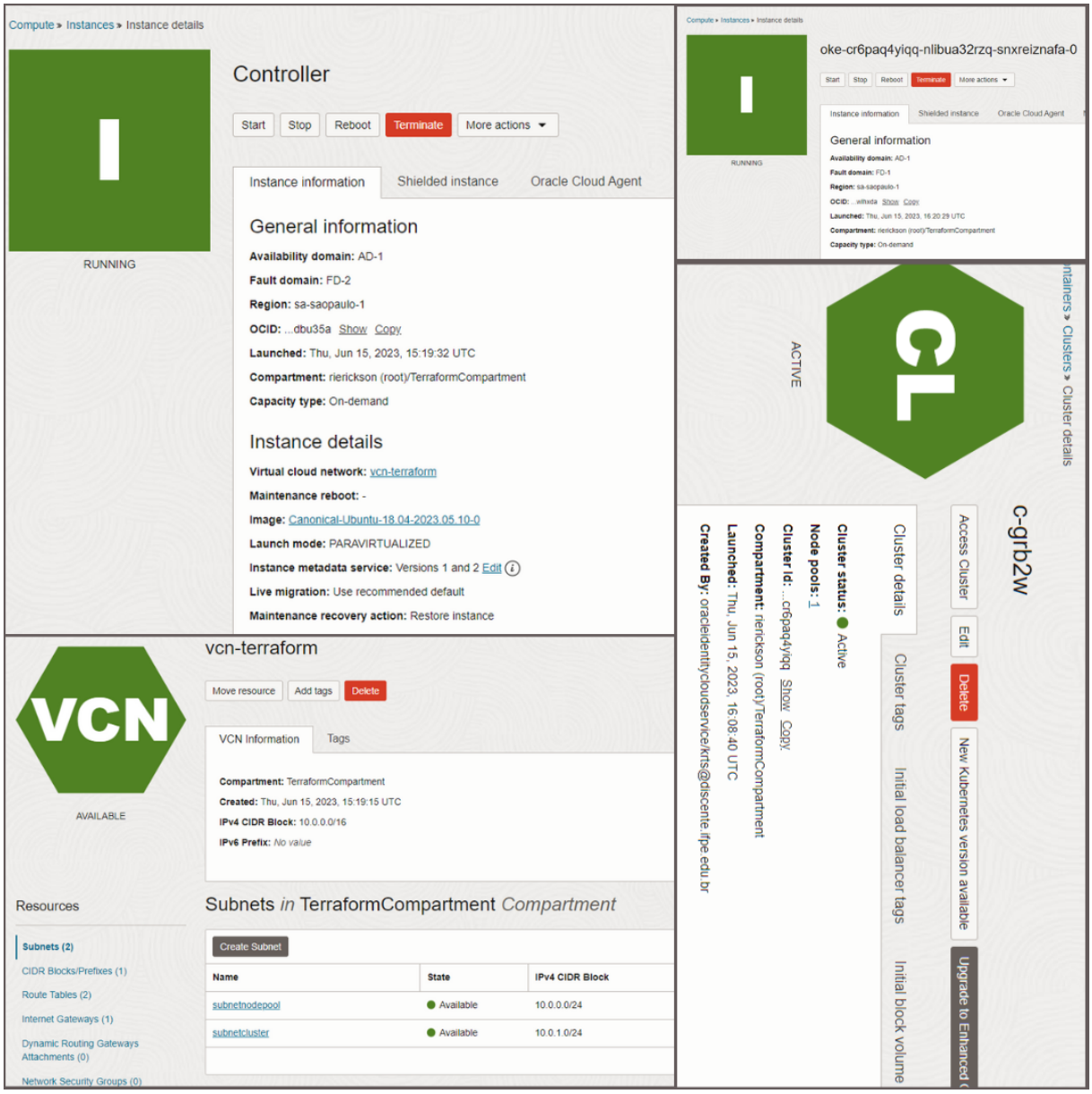
5.1

RESULTADOS POR OBJETIVOS

Neste capítulo, será feito um levantamento dos resultados obtidos na realização deste trabalho, com base nos objetivos estabelecidos. Os objetivos gerais eram a criação e disponibilização de uma infraestrutura de deployment de aplicações web na nuvem, baseada em Kubernetes e construída com algumas ferramentas de IaC, com foco em aproximar os engenheiros de software das tecnologias utilizadas. Para o objetivo geral ser satisfeito, foram estabelecidos mais alguns objetivos específicos, os quais serão apresentados um a um logo a seguir.

Objetivo 1: Criar a infraestrutura de um *cluster* em uma nuvem pública utilizando IaC Para atingir esse objetivo, foi realizada a configuração de um cluster Kubernetes em uma nuvem pública, utilizando ferramentas de Infrastructure as Code (IaC). Foi criado um conjunto de *scripts* e definições de infraestrutura, que permitiram a criação automatizada da infraestrutura, com todas as configurações necessárias. Essa infraestrutura proporcionou uma base sólida para o deployment de aplicações web.

Figura 5.1: Todos os recursos criados na OCI

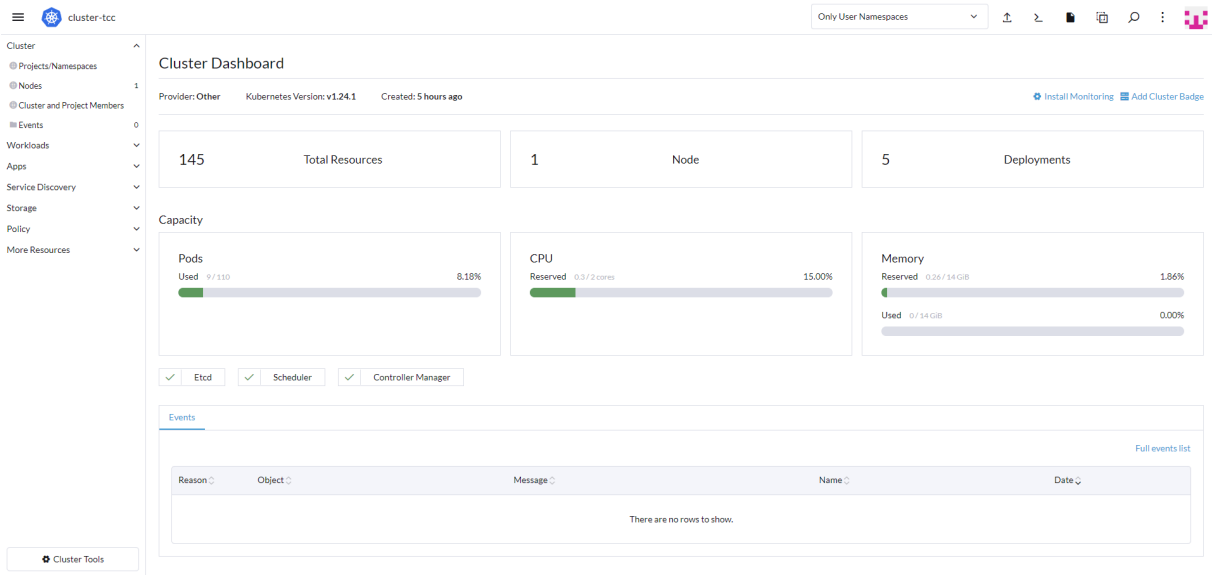


(a) Fonte: Próprio Autor (2023)

Mas para que essa automação não se limitasse a infraestrutura e fosse aplicada a todo projeto seria necessário utilizar o kubernetes não gerenciado por dois motivos: O primeiro é que como uma *cloud* mais recente a OCI ainda não conta com um plugin de importação de *cluster* para o Rancher e para a realização desse trabalho não seria viável utilizar outra nuvem, pois não havia acesso a uma conta de teste com créditos em nenhuma outra e esses créditos foram essenciais para a realização do trabalho em tempo hábil. O segundo é que no Rancher atualmente os *clusters* só podem ser gerenciados, criados e/ou importados via código quando se utiliza o k8s não gerenciado e isso também não seria viável vide o tempo e esforço que seriam necessários para conclusão desse trabalho não estarem no prazo final. Outro impeditivo que aconteceu

decorrente a falta de uma ferramenta que una Rancher com OCI é a não utilização do ArgoCD, que tem um papel semelhante com o Fleet, porém ele é responsável por monitorar o repositório de infraestrutura e cada alteração feita seria replicada na OCI, especificamente no *cluster* OKE.

Figura 5.2: Cluster implementado no Rancher



(a) Fonte: Próprio Autor (2023)

Objetivo 2: Introduzir desenvolvedores à utilização de práticas DevOps no deployment de sistemas web.

Em todo o processo de desenvolvimento do trabalho e com o cluster Kubernetes em funcionamento, foi possível introduzir os desenvolvedores às práticas do DevOps no processo de deployment de sistemas web. Foram adotados conceitos e ferramentas como integração contínua (CI), entrega contínua (CD) e automação de testes no repositório da IaC. Os desenvolvedores puderam se familiarizar com essas práticas e entender sua importância para a qualidade e eficiência do desenvolvimento de software e na implementação de infraestrutura também.

Objetivo 3: Mostrar a importância do passo de implantação no desenvolvimento de softwares web.

Durante a realização deste trabalho, foi possível destacar a importância do passo de implantação no ciclo de desenvolvimento de softwares web. Muitas vezes, esse processo é negligenciado ou considerado apenas um detalhe, mas sua correta execução é fundamental para garantir a estabilidade, escalabilidade e disponibilidade de uma aplicação. Os desenvolvedores puderam compreender a relevância desse passo e adotar boas práticas para sua execução.

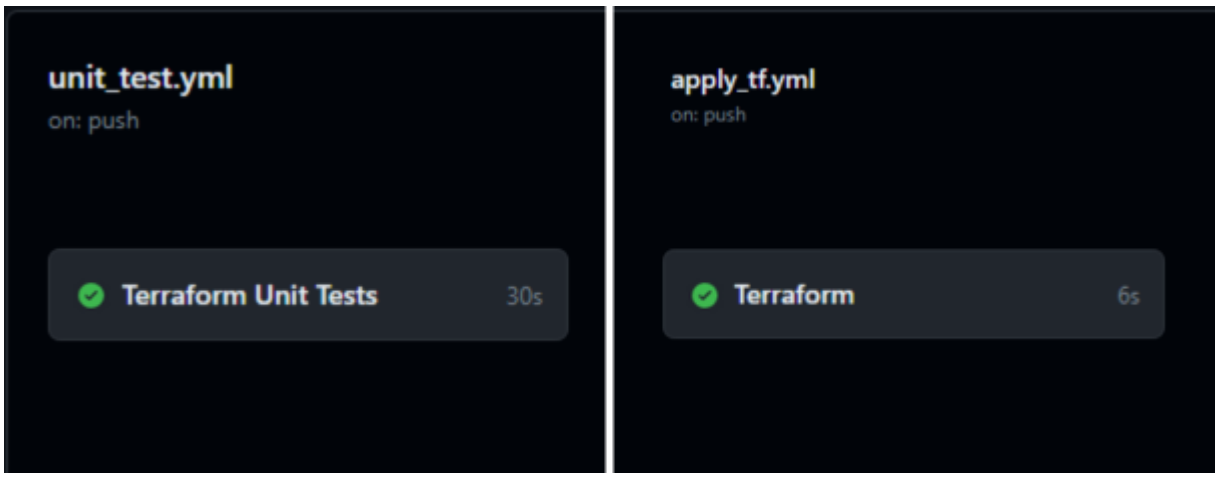
Objetivo 4: Instalar e configurar uma ferramenta de gerenciamento de clusters, com foco em facilitar o gerenciamento desses clusters pelos desenvolvedores.

Foi realizada a instalação e configuração de uma ferramenta de gerenciamento de clusters Kubernetes: O Rancher, com o objetivo de facilitar o trabalho dos desenvolvedores na administração e operação do cluster. Essa ferramenta proporcionou recursos adicionais, como monitoramento, escalabilidade automática e gerenciamento simplificado dos recursos do cluster. Os desenvolvedores e operadores foram capacitados a utilizar essa ferramenta e colher os benefícios de sua utilização.

Objetivo 5: Criar uma infraestrutura simples de pipeline de CI/CD integrada à infraestrutura criada.

Foi desenvolvida uma infraestrutura simples de pipeline de integração contínua e entrega contínua (CI/CD), integrada à infraestrutura criada. Esse pipeline permitiu a automação de diversas etapas do processo de deployment, como build, testes automatizados, empacotamento e implantação. Com a infraestrutura de pipeline, os desenvolvedores puderam acelerar o ciclo de desenvolvimento e reduzir erros humanos na implantação de suas aplicações.

Figura 5.3: Actions do Pipeline do repositorio



(a) Fonte: Próprio Autor (2023)

Objetivo 6: Aplicar conceitos de GitOps para versionamento, controle e provisionamento da infraestrutura criada.

Os conceitos de GitOps foram aplicados para o versionamento, controle e provisionamento da infraestrutura criada. Utilizando um repositório Git como fonte única de verdade, foi possível manter o histórico das alterações realizadas na infraestrutura e controlar as versões das configurações. Além disso, o provisionamento da infraestrutura foi automatizado, permitindo a replicação e recuperação rápida em caso de falhas.

6

CONCLUSÕES E TRABALHOS FUTUROS

Com base nos objetivos estabelecidos, podemos afirmar que todos foram atingidos neste trabalho. Foi criada uma infraestrutura de deployment de aplicações web na nuvem, baseada em Kubernetes e construída com algumas ferramentas de IaC. Os desenvolvedores foram introduzidos às práticas do DevOps, compreenderam a importância do passo de implantação no desenvolvimento de softwares web e utilizaram uma ferramenta de gerenciamento de clusters para facilitar suas tarefas. Além disso, uma infraestrutura simples de pipeline de CI/CD foi implementada, juntamente com a aplicação de conceitos de GitOps para versionamento, controle e provisionamento da infraestrutura.

Esses resultados demonstram que a infraestrutura criada e as práticas adotadas podem trazer benefícios para o desenvolvimento de softwares web, como maior agilidade, qualidade e confiabilidade nos processos de deployment. Recomenda-se a continuidade da utilização dessas práticas e a exploração de outras tecnologias e ferramentas relacionadas, visando a melhoria contínua dos processos de desenvolvimento e implantação de aplicações web.

6.1

TRABALHOS FUTUROS

Trabalhos Futuros:

- Criação de uma infraestrutura com o Kubernetes (k8s) não gerenciado, possibilitando a automação de 100 por cento do ambiente. Nessa melhoria do projeto, nosso objetivo é estabelecer uma infraestrutura escalável e altamente disponível utilizando o k8s não gerenciado, assim possibilitando a criação de todos os recursos via IaC e realizar esse gerenciamento como código também.
- Implementação de um plugin para a Oracle Cloud Infrastructure (OCI) no Rancher. Reconhecido a importância de fornecer suporte nativo ao OCI no Rancher,

uma plataforma de gerenciamento de contêineres amplamente utilizada. O objetivo é desenvolver um plugin que permita uma integração perfeita entre o Rancher e os clusters da OCI. Com esse plugin, será possível provisionar e gerenciar clusters OKE. Essa integração simplificada proporcionará aos usuários uma experiência unificada, permitindo que aproveitem ao máximo as capacidades do OKE em seus ambientes gerenciados pelo Rancher.

- Criação de plugins para o gerenciamento do Rancher em infraestruturas na nuvem. Com o aumento da adoção de serviços em nuvem, é essencial oferecer suporte eficiente para as principais provedoras de infraestrutura. Nesse sentido, desenvolver plugins que facilitem a gestão do Rancher em diferentes ambientes de nuvem, como AWS, Azure e Google Cloud Platform. Esses plugins permitirão a integração perfeita do Rancher com as funcionalidades específicas de cada provedor, proporcionando uma experiência unificada e simplificada para o gerenciamento dos clusters criados em diferentes clouds.

- Adicionar o ArgoCD no fluxo de criação, para monitoramento e atualização automática da infraestrutura mediante um repositório no github.

Através dessas iniciativas, pretende melhorar a eficiência, a escalabilidade e a integração do ambiente de infraestrutura, aproveitando as tecnologias emergentes e garantindo uma base sólida para o crescimento e o sucesso contínuo do projeto desenvolvido.

REFERÊNCIAS

ABSAM. **Vantagens de utilizar Docker para montar seu ambiente**. [S.l.: s.n.], 2023. Disponível em: https://absam.io/blog/vantagens-de-utilizar-docker-para-montar-seu-ambiente/. Acessado em: 04/06/2023.

AMAZON. **AWS Cloud Formation**. [S.l.: s.n.]. Disponível em: https://aws.amazon.com/pt/cloudformation/. Acessado em: 03/06/2023.

_____. **O que é DevOps?** [S.l.: s.n.]. Disponível em: https://aws.amazon.com/pt/devops/what-is-devops/. Acessado em: 14/05/2023.

AZURE, Microsoft. **O que é o DevOps?** [S.l.: s.n.], - 2022. Disponível em: https: //azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-devops/. Acessado em: 14/12/2022.

BERNARDO, Fernanda. **Git:** o que é, para que serve e principais comandos Git! [S.l.: s.n.], 2022. Disponível em: https://blog.betrybe.com/git/. Acessado em: 31/05/2023.

BHARDWAJ, Sushil; JAIN, Leena; JAIN, Sandeep. Cloud computing: A study of infrastructure as a service IAAS. **International Journal of engineering and information Technology**, v. 2, n. 1, p. 60–63, 2010.

BITTMAN, Tom. The future of cloud services is a mix of private and public clouds. **Gartner**, 2010.

BUCHANAN, IAN. **Infraestrutura como código**. [S.l.: s.n.], 2023. Disponível em: https://www.atlassian.com/br/microservices/cloud-computing/infrastructure-as-code. Acessado em: 03/06/2023.

BURILLO, MATEO. **Kubernetes monitoring with Prometheus, the ultimate guide**. [S.l.: s.n.], 2021. Disponível em: https://sysdig.com/blog/kubernetes-monitoring-prometheus/. Acessado em: 04/06/2023.

CAREY, Scott. **Por que ninguém quer mais gerenciar Kubernetes**. [S.l.: s.n.], 2021. Disponível em: https://itforum.com.br/noticias/por-que-ninguem-quer-mais-gerenciar-kubernetes/. Acessado em: 05/06/2023.

CHIA, William. **Push vs. Pull in GitOps:** Is There Really a Difference? [S.l.: s.n.], 2021. Disponível em: https://thenewstack.io/push-vs-pull-in-gitops-is-there-really-a-difference/. Acessado em: 26/05/2023.

CODEBLOG. **Quais são os quatro pilares da DevOps? Descubra como essa cultura é estruturada.** [S.l.: s.n.], mai. 2022. Disponível em: https://codebit.com.br/blog/empresas/quais-sao-quatro-pilares-devops-descubra-essa-cultura-estruturada. Acessado em: 22/05/2023.

COHN, Mike. **Succeeding with Agile:** Software Development Using Scrum. [S.l.]: Addison-Wesley Signature Series, 2009.

CONTAINERIZE. **Gerenciar contêineres com a plataforma de gerenciamento de Kubernetes**. [S.l.: s.n.]. Disponível em: https://products.containerize.com/pt/deployment-tools/rancher/. Acessado em: 07/06/2023.

CRONAPPS, Redação. **Sistema On-premise: o que é preciso para a sua criação?** [S.l.: s.n.], fev. 2022. Disponível em: https://blog.cronapp.io/sistema-on-premise/O_que_e_um_sistema_on-premise. Acessado em: 20/12/2022.

DANIELA. **O Que é Docker e Como Ele Funciona?** [S.l.: s.n.], 2023. Disponível em: https://www.hostinger.com.br/tutoriais/o-que-e-docker. Acessado em: 04/06/2023.

DELGADO, Caio. **Ansible, Puppet, Chef, Terraform:** Qual ferramenta de Infrastructure As Code devo utilizar? [S.l.: s.n.], 2020. Disponível em: https://caiodelgado.dev/iac-tools/. Acessado em: 03/06/2023.

DEVMEDIA. **Introdução ao Maven**. [S.l.: s.n.]. Disponível em: https://www.devmedia.com.br/introducao-ao-maven/25128. Acessado em: 31/05/2023.

DOCKER. **Develop faster. Run anywhere.** [S.l.: s.n.], 2022. Disponível em: https://www.docker.com. Acessado em: 04/06/2023.

_____. **Docker Developer Tools**. [S.l.: s.n.], 2023. Disponível em: https://www.docker.com/products/developer-tools/. Acessado em: 04/06/2023.

DOCUSIGN, Colaborador. **Conheça os tipos de cloud computing e entenda suas diferenças**. [S.l.: s.n.], 16 2019. Disponível em: https://www.docusign.com.br/blog/tipos-de-cloud-computing. Acessado em: 11/05/2023.

DUBEY, Abhijit; WAGLE, Dilip. Delivering software as a service. **The McKinsey Quarterly**, v. 6, n. 2007, p. 2007, 2007.

FENSTERER, Markus. **Managed Kubernetes vs self-managed Kubernetes**. [S.l.: s.n.], 2022. Disponível em: https://www.x-cellent.com/posts/managed-kubernetes-vs-self-managed-kubernetes. Acessado em: 05/06/2023.

FOUNDATION, Cloud Native Computing. **Cri-o - Lightweight Container runtime for Kubernetes**. [S.l.: s.n.]. Disponível em: https://cri-o.io. Acessado em: 04/06/2023.

GAEA. **O que é Continuous Delivery?** [S.l.: s.n.], mar. 2021. Disponível em: https://gaea.com.br/o-que-e-continuous-delivery/. Acessado em: 24/05/2023.

_____. **O que é continuous deployment?** [S.l.: s.n.], mar. 2022. Disponível em: https://gaea.com.br/o-que-e-continuous-deployment/. Acessado em: 24/05/2023.

_____. **Pilares do DevOps, você sabe quais são?** [S.l.: s.n.], mai. 2019. Disponível em: https://gaea.com.br/quais-sao-os-pilares-do-devops/. Acessado em: 22/05/2023.

GAEA. **Guia completo:** DevOps e a cultura do código. [S.l.: s.n.], fev. 2019. Disponível em: https://gaea.com.br/guia-completo-devops-e-a-cultura-do-codigo/. Acessado em: 21/05/2023.

GARTNER. **"Gartner Says Four Trends Are Shaping the Future of Cloud, Data Center and Edge Infrastructure**. [S.l.: s.n.], mai. 2023. Disponível em: https://www.gartner.com/en/newsroom/press-releases/2023-05-16-gartner-says-4-trends-are-shaping-the-future-of-cloud-data-center-and-edge-infrastructure. Acessado em: 25/05/2023.

_____. **Gartner Forecasts Worldwide Public Cloud End-User Spending to Reach Nearly $600 Billion in 2023**. [S.l.: s.n.], 2023. Disponível em: https://www.gartner.com/en/newsroom/press-releases/2023-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023. Acessado em: 25/05/2023.

GITLAB. **What is GitOps?** [S.l.: s.n.]. Disponível em: https://about.gitlab.com/topics/gitops. Acessado em: 26/05/2023.

GONCALVES, Fernando. **O que é Kubernetes? Tudo que você precisa saber sobre!** [S.l.: s.n.], 2021. Disponível em: https://blog.geekhunter.com.br/kubernetes-a-arquitetura-de-um-cluster/. Acessado em: 04/06/2023.

GUEDES, Marylene. **Afinal, o que é um container?** [S.l.: s.n.], jan. 2021. Disponível em: https://www.treinaweb.com.br/blog/afinal-o-que-e-um-container. Acessado em: 20/12/2022.

HARRER, Florian Beetz/ Anja Kammer/ Dr. Simon. **GitOps**. [S.l.: s.n.]. Disponível em: https://www.gitops.tech. Acessado em: 26/05/2023.

HASHICORP. **What is Terraform?** [S.l.: s.n.]. Disponível em: https://developer.hashicorp.com/terraform/intro. Acessado em: 03/06/2023.

HELM. **Helm Docs**. [S.l.: s.n.], 2023. Disponível em: https://helm.sh/pt/docs/. Acessado em: 04/06/2023.

HUMBLE JEZ - FARLEY, David. **Continuous Delivery:** Reliable Software Releases through Build, Test, and Deployment Automation. [S.l.]: Addison-Wesley Signature Series, 2010.

IBM. **The power of cloud:** Driving business model innovation. [S.l.: s.n.], 2012. Disponível em: https://www.ibm.com/services/us/gbs/thoughtleadership/cloudstudy2012/. Acessado em 12/05/2023.

JULIAKM. **O que é IaC**. [S.l.: s.n.]. Disponível em: https://learn.microsoft.com/pt-br/devops/deliver/what-is-infrastructure-as-code. Acessado em: 10/07/2023.

KELLY, Allan. **Testing triangles, pyramids and circles, and UAT**. [S.l.: s.n.], mai. 2013. Disponível em: https://www.allankelly.net/archives/628/testing-triangles-pyramids-and-circles/#comments. Acessado em: 24/05/2023.

KNORR, Eric; GRUMAN, Galen. What cloud computing really means. **InfoWorld**, v. 7, n. 20-20, p. 1–17, 2008.

KUBERNETES. **Instalando a ferramenta kubeadm**. [S.l.: s.n.], 2023. Disponível em: https://kubernetes.io/pt-br/docs/setup/production-environment/tools/kubeadm/install-kubeadm/. Acessado em: 08/06/2023.

KUBERNETS. **Componentes do Kubernetes**. [S.l.: s.n.], abr. 2022. Disponível em: https://kubernetes.io/pt-br/docs/concepts/overview/components/. Acessado em: 13/12/2022.

_____. **Orquestração de contêineres prontos para produção**. [S.l.: s.n.]. Disponível em: https://kubernetes.io/pt-br/. Acessado em: 20/12/2022.

LABS, Grafana. **Get started with Grafana and Prometheus**. [S.l.: s.n.], 2022. Disponível em: https://grafana.com/docs/grafana/latest/getting-started/get-started-grafana-prometheus/. Acessado em: 04/06/2023.

LUCAS. **Infrastructure as Code (IaC):** a automação de infraestrutura para agilizar projetos de TI. [S.l.: s.n.], 2023. Disponível em: https://www.dio.me/articles/infrastructure-as-code-iac-a-automacao-de-infraestrutura-para-agilizar-projetos-de-ti. *Infrastructure as Code (IaC)*: a automação de infraestrutura para agilizar projetos de TI.

MARCIO. **Docker e Containers**. [S.l.: s.n.], 2019. Disponível em: https://medium.com/tecnologia-e-afins/o-que-é-docker-188e283088dd. Acessado em: 04/06/2023.

MELL, Peter; GRANCE, Timothy. **The NIST Definition of Cloud Computing**. [S.l.], 2011.

ORACLE. **Container Engine for Kubernetes**. [S.l.: s.n.], 2022. Disponível em: https://www.oracle.com/br/cloud/cloud-native/container-engine-kubernetes/. Acessado em: 19/06/2023.

PIRES, Aécio. **Primeiros passos com SaltStack**. [S.l.: s.n.], 2018. Disponível em: https://blog.aeciopires.com/primeiros-passos-com-saltstack/. Acessado em: 03/06/2023.

PODMAN. **Get Started with Podman**. [S.l.: s.n.], 2023. Disponível em: https://podman.io. Acessado em: 04/06/2023.

PORTNOY, Matthew. **Virtualization Essentials**. [S.l.]: John Wiley & Sons, 2016.

RANCHER. **Overview**. [S.l.: s.n.], 2023. Disponível em: https://fleet.rancher.io. Acessado em: 03/06/2023.

REDES, Escola Superior de. **Containers e Docker:** o que são e como utilizar. [S.l.: s.n.], 2021. Disponível em: https://esr.rnp.br/administracao-de-sistemas/containers-docker-como-utilizar/. Acessado em: 04/06/2023.

REDHAT. **Containers e máquinas virtuais VMs**. [S.l.: s.n.], 2020. Disponível em: https://www.redhat.com/pt-br/topics/containers/containers-vs-vms. Acessado em: 04/06/2023.

_____. **Introdução ao DevOps**. [S.l.: s.n.], mai. 2022. Disponível em: https://www.redhat.com/pt-br/topics/devops#visão-geral. Acessado em: 14/05/2023.

_____. **Kubernetes cluster**. [S.l.: s.n.], jan. 2020. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-a-kubernetes-cluster. Acessado em: 13/12/2022.

_____. **O que é orquestração de containers?** [S.l.: s.n.], 2019. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-container-orchestration. Acessado em: 03/06/2023.

REDHAT. **What is Infrastructure as Code IaC?** [S.l.: s.n.], mai. 2022. Disponível em: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac. Acessado em: 13/12/2022.

_____. **Why Ansible?** [S.l.: s.n.], 2020. Disponível em: https://www.ansible.com/overview/it-automation. Acessado em: 03/06/2023.

REDHAT. **What is GitOps?** [S.l.: s.n.], 2023. Disponível em: https://www.redhat.com/en/topics/devops/what-is-gitops. Acessado em: 26/05/2023.

REHKOPF, MAX. **What is continuous integration?** [S.l.: s.n.]. Disponível em: https://www.atlassian.com/continuous-delivery/continuous-integration. Acessado em: 24/05/2023.

RIERICKSON, Kaique. **oke-terraform**. [S.l.: s.n.], 2023. Disponível em: https://github.com/RiericksonT/oke-terraform. Acessado em: 20/06/2023.

RITTINGHOUSE, J.W.; RANSOME, J.F. **Cloud Computing:** Implementation, Management, and Security. [S.l.]: CRC Press, 2017. ISBN 9781351615365. Disponível em: <Dispon%C3%ADvel%20em:%20https: //books.google.com.br/books?id=jB-9DgAAQBAJ>.

RODRIGUES, Helias. **Quais as vantagens do Rancher e sua relação com o Kubernetes?** [S.l.: s.n.], 2022. Disponível em: https://blog.o2b.com.br/quais-as-vantagens-do-rancher-e-sua-relacao-com-o-kubernetes/. Acessado em: 07/06/2023.

SHAUL, Michael. **Kubernetes Scaling:** The Comprehensive Guide to Scaling Apps in Kubernetes. [S.l.: s.n.], 2022. Disponível em: https://bluexp.netapp.com/blog/cvo-blg-kubernetes-scaling-the-comprehensive-guide-to-scaling-apps. Acessado em: 04/06/2023.

SOUZA, Luiz Eduardo Eleuterio de. **Gerenciador de contêiner Docker Rancher.** [S.l.: s.n.], 2023. Disponível em: https://www.linkedin.com/pulse/gerenciador-de-contêiner-docker-rancher-eleuterio-de-souza/?originalSubdomain=pt. Acessado em: 07/06/2023.

STATEN, James. Private cloud's not dead- it's just on hold. **Forrester Research**, 2011.

STORZ, Chris. **What Is GitOps? Learn About Benefits, Challenges, and More**. [S.l.: s.n.], 2022. Disponível em: https://www.harness.io/blog/gitops. Acessado em: 26/05/2023.

TAURION, Cezar. **Cloud computing-computação em nuvem**. [S.l.]: Brasport, 2009.

TEAM, MJV. **8 melhores ferramentas de gestão de projetos**. [S.l.: s.n.], 2021. Disponível em: https://www.mjvinnovation.com/pt-br/blog/8-ferramentas-de-gestao-de-projetos. Acessado em: 31/05/2023.

TOTVS, EQUIPE. **PaaS:** o que é, como funciona, modelos, benefícios e exemplos. [S.l.: s.n.], abr. 2022. Disponível em: https://www.totvs.com/blog/negocios/paas/. Acessado em: 10/05/2023.

TSAI, Wei Tek; HUANG, Yu; SHAO, Qihong. **EasySaaS:**A SaaS development framework. English US. In: PROCEEDINGS - 2011 IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2011. [S.l.: s.n.], dez. 2011. 2011 IEEE International Conference on Service-Oriented Computing and Applications, SOCA 2011 ; Conference date: 12-12-2011 Through 14-12-2011. ISBN 9781467303194. DOI: 10.1109/SOCA.2011.6166262.

WATTS, Stephen. **Kubernetes 101:** How To Set Up “Vanilla” Kubernetes. [S.l.: s.n.], 2022. Disponível em: https://www.splunk.com/en_us/blog/learn/kubernetes-vanilla-setup.html. Acessado em: 05/06/2023.

WAVEWORKS. **Guide To GitOps**. [S.l.: s.n.], ago. 2018. Disponível em: https://www.weave.works/technologies/gitops/. Acessado em: 25/05/2023.