PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
ESCOLA POLITÉCNICA
GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

TORNANDO ACESSÍVEL A CULTURA *DEVOPS* A PEQUENAS EMPRESAS E *STARTUPS*

LUIZ HENRIQUE BUFFA

GOIÂNIA
2021

LUIZ HENRIQUE BUFFA

TORNANDO ACESSÍVEL A CULTURA *DEVOPS* A PEQUENAS EMPRESAS E *STARTUPS*

Trabalho de Conclusão de Curso apresentado à Escola Politécnica, da Pontifícia Universidade Católica de Goiás, como parte dos requisitos para a obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Prof. M.E.E. Marcelo Antonio Adad de Araújo

GOIÂNIA
2021

LUIZ HENRIQUE BUFFA

TORNANDO ACESSÍVEL A CULTURA *DEVOPS* A PEQUENAS EMPRESAS E *STARTUPS*

Trabalho de Conclusão de Curso aprovado em sua forma final pela Escola Politécnica, da Pontifícia Universidade Católica de Goiás, para obtenção do título de Bacharel em Engenharia de Computação, em 8/12/2021.

_____

Prof. Ma. Ludmilla Reis Pinheiro dos Santos
Coordenadora de Trabalho de Conclusão de Curso

Banca examinadora:

_____

Orientador: Prof. M.E.E. Marcelo Antonio Adad de Araújo

_____

Prof. M.S.C. Pedro Araújo Valle

_____

Prof. M.E.E. Carlos Alexandre Ferreira de Lima

GOIÂNIA
2021

Com gratidão, a Deus por tudo que tem feito em minha vida.

Aos meus pais, sem eles nada seria possível, pelo amor e assistência nos momentos difíceis.

Aos meus familiares, minha base.

A todos que me ajudaram e estiveram presentes em minha trajetória acadêmica.

AGRADECIMENTOS

A Deus por me proporcionar a perseverança, me mantendo no caminho certo durante a elaboração deste trabalho, providenciando a força e saúde necessárias.

Aos meus pais Cristiane da cunha Buffa e Carlos Buffa Neto pelo apoio incondicional ao longo de toda minha vida e por sempre investirem em mim e em minha educação.

Ao meu professor orientador Marcelo Antonio Adad de Araújo pela paciência e confiança depositada em mim, bem como todas as instruções necessárias para a realização deste trabalho. Aos meus amigos por suas valiosas contribuições durante todo o curso.

RESUMO

O presente trabalho apresenta um estudo prático sobre a implementação e viabilização da cultura *DevOps* (aproximação entre desenvolvedores e operadores) para pequenas empresas e startups. Para que isso seja aplicável, o estudo prático será desenvolvido utilizando apenas ferramentas de código aberto. Nesse trabalho serão exploradas as ferramentas: *Ansible*, *Docker*, *Kubernetes*, *GitHub Actions*, *Cypress*, Ubuntu, *Postgres* e *Redis*. Para demonstrar a utilização da cultura *DevOps* será implementado um sistema simples de votação. Esse sistema será desenvolvido explorando as linguagens: *Python*, .NET, Node.js e Shell Script. O objetivo central do trabalho é viabilizar e implementar a cultura *DevOps* gerando soluções para suprir a demanda diária do processo de desenvolvimento de código utilizando as tecnologias mencionadas acima, auxiliando nas tarefas de *Deploy*, *Rollback, continuous delivery/Deployment* e *continuous integration.*

Palavras-Chave: *DevOps*. *Ansible*. *Kubernetes*. Continuous delivery/*Deploy*ment. Continuous integration.

ABSTRACT

This work presents a practical study on the implementation and viability of the *DevOps* culture for small companies and startups, so that this is possible, the practical study will be developed using only open-source tools. In this work, the tools that will be explored: *Ansible*, *Docker*, *Kubernetes*, *GitHub Actions*, *Cypress,* Ubuntu, *Postgres*, and *Redis*. To demonstrate the implementation of the *DevOps* culture, a simple *voting* system will be implemented, this system will be developed exploring the languages: *Python*, .NET, Node.js, and Shell Script. The main objective of the work will be to enable and implement the *DevOps* culture, generating solutions to meet the daily demand of the code development process using the technologies mentioned above, assisting in the tasks of *Deploy*, *Rollback*, continuous delivery/*Deploy*ment and continuous integration.

Keywords: *DevOps*; *Ansible*. *Kubernetes*. Continuous delivery/*Deploy*ment. Continuous integration.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

Ma. Mestra

M.E.E Mestre em engenharia elétrica

Prof. Professor

PUC Pontifícia Universidade Católica

M.S.C Mestre em Ciência

LISTA DE SIGLAS

AWS	*Amazon Web Services*
CALMS	*Culture, Automation, LeanIT, Measurement e Sharing*
CD	*Continuous Delivery/Continuous Deployment*
CI	*Continuous Integration*
CPU	*Central Process Unit*
CRT	*Cathodic Ray Tube*
DNS	*Domain Name System*
EC2	*Amazon Elastic Compute Cloud*
GB	*Gigabytes*
HTTPS	*Hyper Text Transfer Protocol Secure*
IaC	*Infrastructure as Code*
MB	*Megabyte*
RAM	*Random Access Memory*
SO	*Sistema Operacional*
SRE	*Site Reliability Engineering*
SSD	*Solid Disk Driver*
SSH	*Secure Socket Shell*
SSL	*Secure Sockets Layer*
TI	Tecnologia da Informação
VM	*Virtual Machine*

SUMÁRIO

1 INTRODUÇÃO

O presente trabalho de conclusão de curso visa demonstrar de forma prática a implementação da cultura *DevOps*, utilizando apenas ferramentas de código aberto ou gratuitas com a finalidade de otimizar os processos diários no dia a dia de pequenas empresas de desenvolvimento e *startups*.

DevOps pode ser traduzido para o português como *Dev* (desenvolvimento) e *Ops* (operação), a cultura *DevOps* é a união de pessoas, processos, metodologias e tecnologia para gerenciar e automatizar o processo de desenvolvimento de códigos computacionais. Adotar a cultura *DevOps* significa automatizar os processos de: *Deploy* (que consiste em implementar, de forma a disponibilizar um sistema para uso), *Rollback* (consistindo em reverter, retornar um sistema para um estado anterior que seja funcional no caso de erros ou falhas que ocorram em novas versões), CI ou *continuous integration* (consiste na integração contínua dos códigos computacionais, testando assim novas mudanças realizadas no código regularmente antes mesmo da disponibilização para uso), CD ou *continuous delivery/Deployment* (é a implantação/entrega contínua do código computacional, depois dos testes realizados no processo de CI, disponibiliza-se o novo código criado em um repositório de códigos como o *GitHub* ou em repositório de *containers* como o *Docker Hub* em seguida realiza-se o *Deploy* deste código computacional ou *container* (empacotamento de um código e suas dependências em uma imagem computacional)).

Ao se gerenciar a automatização dos processos já citados, é imprescindível utilizar a ferramenta *Ansible* (é a ferramenta utilizada para transformar o gerenciamento da infraestrutura em um código de programa). A mesma ferramenta também será utilizada para configurar o servidor onde será hospedada a aplicação de votação que será executada pelo usuário. Toda a otimização do *Ansible* é realizada através de *playbooks* (um código para execução de tarefas de automação complexas com envolvimento humano limitado, com pouquíssima intervenção humana ou mesmo sem qualquer intervenção).

A cultura *DevOps* pode ser implementada em qualquer processo de desenvolvimento de código de programa. Para se ter maior visibilidade dessa cultura em ação, é utilizada uma simples aplicação de votação de código aberto, onde o usuário final poderá votar a partir de uma interface *web* entre a escolha de um *dog*

(cachorro) ou *cat* (gato), o resultado dessa votação será exibido em uma outra interface *web* desenvolvida para apresentar o resultado.

Todos os códigos utilizados são empacotados em *containers* e enviados para o *Docker Hub*, em seguida são baixados e executados dentro de uma máquina na *AWS* (Plataforma em nuvem da Amazon) que roda o sistema operacional Ubuntu *18.04* (Sistema operacional baseado em Linux) instalado somente para esse fim. Os *containers* serão orquestrados pela ferramenta *Kubernetes* utilizada para tanto.

Sendo assim, o presente trabalho busca responder a seguinte questão de pesquisa:

- É possível implementar a cultura *DevOps* utilizando apenas ferramentas de código aberto, com isso viabilizando a sua implementação para pequenas empresas de desenvolvimento de códigos e *startups*?

1.1 Objetivos

Esta seção descreve os objetivos geral e específicos do presente trabalho de conclusão de curso.

1.1.1 Objetivo Geral

O trabalho tem por objetivo implementar uma infraestrutura automatizada, utilizando os princípios da cultura *DevOps* e apenas ferramentas e aplicações de código livre.

1.1.2 Objetivos Específicos

- Criar o ambiente na *AWS* (plataforma em nuvem para gerenciamento e criação de máquinas virtuais) para o ambiente de produção.
- Criar os repositórios do *GitHub* e *Docker Hub*, de forma a manter os códigos e suas versões e as imagens encapsuladas da aplicação;
- Implementar a *playbook* de instalação da infraestrutura, transformando a criação da infraestrutura em um código;

- Implementar a *playbook* de configuração do CI e CD, automatizando o processo de teste de código de programa e o processo de implementação automatizada do código;
- Criar os testes ponto a ponto (testar o fluxo de execução da aplicação do começo ao fim de forma a obter um resultado satisfatório) utilizando a ferramenta *Cypress* (ferramenta utilizada em testes automatizados de código de programa);
- Implementar o ambiente de CI utilizando *GitHub Actions* (plataforma responsável por acionar e executar os testes de códigos computacionais automatizados);
- Implementar os *containers* dos serviços utilizados pela aplicação, de forma a obter o envelopamento da aplicação (a totalização das ações realizadas pela aplicação em uma imagem computacional);
- Implementar a *playbook* de instalação e configuração do *Kubernetes*;
- Implementar a orquestração dos *containers* com *Kubernetes;*
- Implementar *playbooks* de *Deploy* e *Rollback;*
- Simular o processo de *Deploy* e *Rollback* através do *Kubernetes.*

1.2 Justificativa

Justifica-se a utilização da cultura *DevOps* unida a ferramentas e aplicações de código aberto que viabiliza a automatização de vários processos rotineiros no dia a dia de uma empresa de desenvolvimento de *software*.

A implementação dessa cultura afeta de forma positiva a empresa, automatizando tarefas existentes, rotineiras e demoradas, reimaginando como as orquestrações são implementadas para garantir que aconteçam por padrão dentro do processo, sem a interferência externa que geralmente causa gargalos e falhas. (Lalatendu Das, Ling Lau, Chris Smith).

2 ESTADO DA ARTE

O presente capítulo contém a fundamentação teórica necessária para a elaboração do trabalho, com conceitos e definições importantes para a compreensão, incluindo trabalhos relacionados ao tema principal.

2.1 *DevOps*

DevOps é um termo relativamente novo na área computacional, mas que vem se tornando cada vez mais popular e que abrange vários conceitos que requerem que sejam explorados para serem completamente compreendidos.

Para entender melhor o que seria *DevOps* é interessante observar a sua história e entender de onde o conceito surgiu e evoluiu.

Volmar (2017) escavando os primórdios da origem do termo *DevOps* inicia pela a história dos computadores. Relembrando, o conceito de computador tem origem na ideia de Alan Turing, em 1936. Como um dispositivo analógico na época, a ideia por trás da máquina de Turing era ler tabelas de dados finitos usando fitas e computação mecânica.

A ideia de Turing levou à criação de computadores eletromecânicos usados para ajudar o governo dos Estados Unidos a decifrar a criptografia do inimigo na Segunda Guerra Mundial. Esses primeiros computadores abriram caminho para os *mainframes*.

Segundo Volmar (2017) desde a criação destes dispositivos, houve a necessidade de os operadores programá-los para realizar tarefas. Os primeiros desenvolvedores, no sentido moderno (utilizar um terminal e escrever o código di*Git*al a ser armazenado) surgiram com a criação da linguagem de programação *Fortran* em 1957.

No final dos anos 1970 e 1980, quando os monitores CRT (tubo de raios catódicos) se tornaram cada vez mais acessíveis, os desenvolvedores poderiam utilizar suas estações de trabalho, escrever o código em desenvolvimento, compilá-lo e testá-lo no mesmo computador. A implantação do código consistia em compilar, copiar para dezenas de dispositivos, disquetes e repetir a operação. As empresas começaram a anunciar posições de programação para escrever “códigos-fonte”.

Porém, poucos desenvolvedores atuavam na engenharia de redes de computadores, pois a *internet* ainda não havia sido inventada. A ideia de “tempo de atividade” e “tempo de inatividade” geralmente era limitada ao fato de um computador estar ligado ou desligado, a menos que se fosse um dos poucos terminais conectados ao que mais tarde se tornaria a origem da *Internet*.

Hyder (2017) explica que antes da *World Wide Web* (WWW), havia a *ARPANET*, uma rede de computadores patrocinada pelo governo nos Estados Unidos que se interconectam usando comutação de pacotes por linhas telefônicas. A *ARPANET* entrou em operação em 1969, com os primeiros “centros de operações de rede”, criando os primeiros trabalhos de engenharia de rede e centros de operações de rede.

Avançando para 2003, já com o uso da *internet*, a Google contratou Ben Treynor para liderar um time de operações em um "ambiente de produção", separado do "ambiente de desenvolvimento". Esse time futuramente faria história, como os primeiros Site *Reliability Engineering* (SREs) ou Engenharia de Confiabilidade de Sites.

Murphy (2016) explica que o foco do SRE está na confiabilidade do sistema, é uma abordagem da engenharia de *software* às operações em TI. Ben Treynor ao desenvolver o termo SRE, cunhou que a principal característica de um sistema deve ser a confiabilidade, e que a SRE tem o objetivo de encontrar formas para aprimorar a operação e o design dos sistemas para torná-los escaláveis, confiáveis e mais eficientes. Antes de Ben Treynor, as operações eram gerenciadas de uma maneira mais voltada para o manual e com poucas automações.

Delgado (2019) em um artigo, contando a história do SRE comenta que sem a automatização dos processos, sempre que a infraestrutura tivesse que aumentar seria necessário crescer também o time de operações para acompanhar esse crescimento, então, um dos objetivos principais de Ben Treynor seria “manter um time de operações de um porte razoável e poder crescer a infraestrutura o quanto fosse necessário”.

O SRE e os *DevOps* têm muito em comum e embora o termo ainda não tenha surgido naquela época, foi aí que surgiram as primeiras nuances do que venha a ser um *DevOps*.

Mas, como explica Delgado (2019), a Google não foi a única a empregar SREs, logo, grandes empresas de tecnologia começaram a implementar filosofias de confiabilidade para reduzir o tempo de inatividade e aumentar a satisfação do cliente.

Volmar (2017) conta que em 2008, por exemplo, o *MobileMe* da *App*le experimentou um tempo de inatividade significativo no qual o serviço de nuvem recebeu muito mais tráfego para seus servidores do que era previsto, se tornando necessário reestruturar a infraestrutura da empresa.

Em 2009 o *Flickr,* um site de hospedagem e compartilhamento de imagens, muito popular na época, com mais de 3 bilhões de fotos e com outros números surpreendentes de imagens armazenadas, até mesmo para época atual, necessitava fazer diversas implementações no dia a dia o que não apresentou bons resultados. Foi aí que entraram John Allspaw e Paul Hammond para resolver este ciclo diário de implementações e notaram que a equipe de operações, encarregada de gerenciar servidores, e a equipe de desenvolvimento, encarregada de criar o código, pareciam estar sempre em desacordo.

Eles então propuseram combinar o "Dev" com o "Ops" e contratar "operadores que pensam como desenvolvedores" e "desenvolvedores que pensam como operadores".

Volmar (2017) relata que, em uma apresentação na *Conferência O'Reilly Velocity* de 2009, Allspaw e Hammond propuseram integrar o desenvolvimento e as operações em uma única infraestrutura automatizada com controle de versão compartilhado com a construção e a implantação, trabalhando juntas.

Nesta conferência de *O'Reilly Velocity*, Allspaw e Hammond apresentaram a palestra de título: *"10+ Deploys Per Day: Dev and Ops Cooperation at Flickr"* que contava sobre os resultados e desafios da maior aproximação necessária entre a equipe de desenvolvimento e de operações no Flickr.

O “*DevOps*” dava seus primeiros passos, mas não tinha formado seu nome contextual ainda.

Mezark (2018) conta que, paralelo aos acontecimentos, na Bélgica, Patrick Debois, um engenheiro que, em 2017, trabalhava com o governo belga para ajudar em migrações do *data center*, passou por um processo frustrante semelhante ao de Allspaw e Hammond, pois encontrou uma barreira entre os desenvolvedores e a

equipe de operações, que tornava seu trabalho muito mais difícil e a entrega muito mais lenta.

Patrick Debois assistiu a palestra online “*10+ Deploys Per Day: Dev and Ops Cooperation at Flickr*” e se identificou com a situação, inspirou-se com a palestra de Allspaw e Hammond e decidiu organizar sua própria conferência, sobre "administração de sistema ágil" e procurando anunciar a conferência no Twitter, criou o termo “*DevOps*” como uma *hashtag* derivada das palavras *“*desenvolvimento” *e “*operações*”.*

Hoje, como conta Mezark (2018), onde desenvolvedores e profissionais de operações se reúnem para discutir automação de processos, testes, segurança e a cultura organizacional necessária para evitar que *Dev* e *Ops* se confrontem.

O termo "*DevOps*" foi popularizado através de uma série de eventos, tendo início com o "*DevOps Days*" e não demorou muito para que as empresas de todos os tamanhos, começassem a reunir práticas de *DevOps*, bem como ferramentas construídas para ajudar essas equipes recém-formadas.

O *DevOps* assim nasceu, da colaboração de desenvolvedores e líderes de operações reunidos para expressar suas ideias e preocupações, procurando a melhor forma de realizar o trabalho, sempre pensando na necessidade de adaptação, agilidade e melhoria contínua. Isso significa que o processo de transição *DevOps* nunca é realmente concluído, pois o próprio sistema deve estar em um estado constante de evolução e melhoria.

Como explica Mala (2019, p.16), com uma visão moderna do termo, o *DevOps* pode ser considerado como uma cultura, que capacita as equipes a construir, testar e implementar com confiança e segurança, em velocidades mais rápidas e com um padrão de qualidade mais alto, que para isso, promovem um conjunto de processos e métodos que melhora a colaboração e a comunicação entre o desenvolvimento e as operações, automatizando a maior quantidade possível de processos operacionais.

2.1.1 Estrutura *DevOps*

CALMS (*Culture Automation LeanIT Measurement Sharing*) é uma sigla em inglês, formada com as iniciais de (*Culture* (Cultura), *Automation* (Automação), *LeanIT* (Metodologia Lean), *Measurement* (Mensuração) e *Sharing* (Compartilhamento). Muller (2020) explica que o termo surgiu em meados de 2008 por John Willis e Damon Edwards e posteriormente, foi aprimorado por Jez Humble**,** co-autor de “O manual *DevOps*” e o acrônimo aborda os cinco elementos fundamentais dos *DevOps* que são:

C*ulture****:*** cultura de aprendizagem e experimentação contínuas, colaborativas

A*utomation*: buscar oportunidades para eliminar o trabalho manual, documentar e padronizar cada parte do processo.

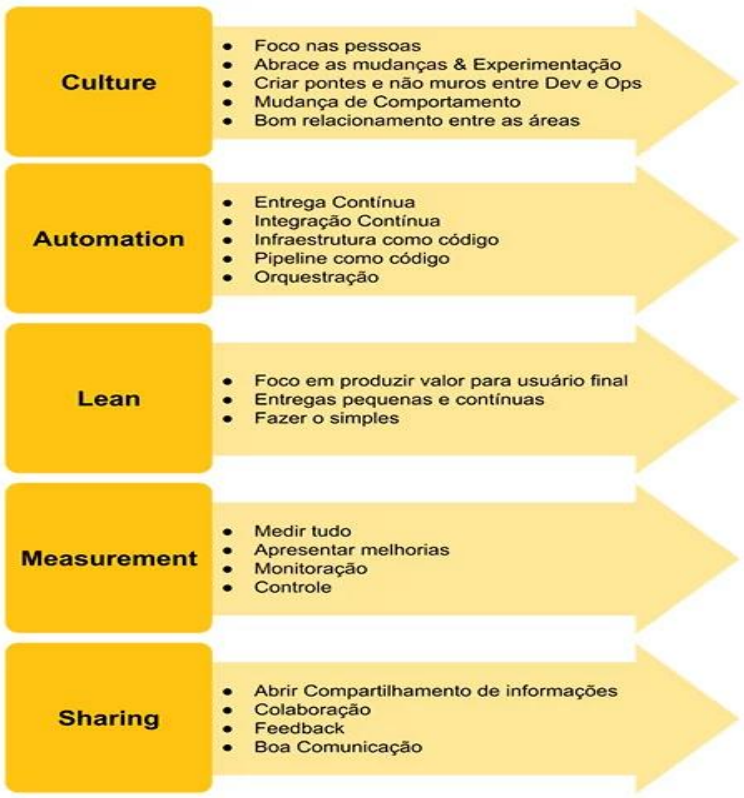
L*ean it*: foco na melhoria contínua, entregando mais velocidade e maior eficiência.

M*easurement*: resultados mensuráveis, cultura orientada em dados com base na medição de seu ambiente de produção e processos.

S*haring:* criar abertura e transparência, descentralizar conhecimento, visualizar soluções locais e escalá-las para a organização global.

A figura 1 apresenta a estrutura *DevOps*:

Figura 1 – Pilares da cultura *DevOps*



Fonte: Resille, 2017.

Conhecer este conjunto de principais práticas é uma boa maneira de compreender um pouco, do que se trata o trabalho de um *DevOps*. Toda a ideia do *DevOps* se baseia na automação dos processos entre as equipes de desenvolvimento de *software* e de operações de TI, para que possam construir, aprimorar, testar e lançar produtos com mais agilidade, rapidez e menos problemas. *DevOps* não é uma ferramenta mágica, é uma mudança cultural que promove a comunicação e estimula a colaboração, para criar um ciclo de desenvolvimento eficiente para que as empresas possam fazer lançamentos e atualizações contínuas, sem sacrificar a qualidade.

2.1.1 Engenheiro *DevOps*

Um engenheiro de computação que atua com *DevOps*, deve ter um amplo conhecimento de desenvolvimento e operações, incluindo codificação, gerenciamento de infraestrutura, administração de sistemas e conjuntos de ferramentas *DevOps*. Além disso, como um dos elementos principais do *DevOps*, deve possuir habilidades

interpessoais para criar um ambiente colaborativo nas empresas de desenvolvimento de *software*.

Hall (2020) explica que a função de um engenheiro de *DevOps* varia de uma empresa para outra, mas de maneira geral, cabe ao profissional *DevOps* introduzir processos, ferramentas e metodologias para equilibrar as necessidades de um *software* e ele opera desde a criação do código até a sua implantação e continua trabalhando em sua manutenção e atualização. A citação inicial é apresentada na sequência:

> Alguns consideram essa pessoa um administrador de sistemas que sabe codificar ou um desenvolvedor com as habilidades de um administrador de sistemas. De certa forma, ambas as definições são justas. A principal função de um engenheiro de *DevOps* é introduzir o fluxo de trabalho de entrega e integração contínua, o que requer o entendimento das ferramentas [*DevOps*] e o conhecimento de várias linguagens de programação**.** *(*Fruhlinger, 2019)

Portanto, os engenheiros de *DevOps* ficam entre o desenvolvimento, implantação e integração contínua, fazendo-se necessário dominar uma série de linguagens e tecnologias que inclusive mudam à medida que a indústria computacional evolui.

2.2 Código Aberto

Código Aberto ou *Open Source* é um modelo de desenvolvimento que incentiva a colaboração aberta para possíveis modificações e *Redis*tribuições.

Como afirma a *Open Source Initiative* (2006) o uso do termo se originou com *software*, mas se expandiu para além do setor de *software* para cobrir outros conteúdos abertos e formas de colaboração aberta. Esses produtos, por assim dizer, abertos, podem ter sido criados por uma comunidade ou por uma empresa, que mantém o código aberto para que possam ser feitas modificações, conforme a necessidade do usuário além de liberarem gratuitamente a consulta, sem que haja a necessidade de se pagar uma licença comercial pelo produto, fomentando um modelo colaborativo de produção intelectual.

Como já visto anteriormente, *DevOps* pode ser visto como um movimento cultural que se apropria de novas práticas e tecnologias para melhorar a integração entre desenvolvimento e o operacional.

Para fazer isso, os profissionais que atuam como *DevOps* utilizam uma série de ferramentas para automatizar as etapas dos processos de desenvolvimento, manutenção e entrega do produto, ferramentas como plataformas de códigos, ferramentas de teste, contêineres, bancos de dados além de outros, que fazem o *DevOps* acontecer e a maioria das melhores ferramentas *DevOps* que existem hoje no mercado são *Open Source*.

Um artigo na EXIN Brasil (2021) esclarece sobre o porquê de as ferramentas *Open Source* serem uma ótima alternativa para o *DevOps*.

O artigo explica que as ferramentas feitas com Código Aberto possuem, como já abordado, uma construção coletiva, mantida por uma comunidade de usuários. Por este motivo, estas ferramentas estão sempre evoluindo de forma a trazer uma melhoria contínua no gerenciamento, impulsionando o desenvolvimento e inovação, o que é observado no texto original a seguir:

> O processo de desenvolvimento de *software*, por si só, é algo extremamente complexo. Quando pensamos em todas as atividades que cercam esse trabalho, como a gestão de um ambiente de testes, publicação para produção, entre outros, a complexidade tende a aumentar. Contar com ferramentas *Open Source* para realizar diversas dessas atividades pode ser uma saída para otimizar todos os processos envolvidos no desenvolvimento de *software*, ao mesmo tempo em que se simplifica tudo isso. (EXIN, 2021)

Outro ponto discutido no artigo é que as ferramentas de Código Aberto contam com uma alta disponibilidade, com soluções que não existem em versões privadas e são constantemente atualizadas pela comunidade que as desenvolve.

Além disso, o *Open Source* ou Código Aberto, como modelo de desenvolvimento adaptável, em que se faz possível realizar alterações no código para adicionar ou retirar funcionalidades, possibilita que os profissionais tenham total controle sobre a ferramenta que se está usando e as customizações; sendo possível criar algo especialmente desenvolvido para as demandas necessárias, ao mesmo tempo em que há redução de gastos, por ser uma tecnologia livre.

2.3 *Containers*

As palavras *Containers* e *DevOps*, geralmente, aparecem sempre juntas em uma mesma conversa. Como abordado anteriormente, o *DevOps* pode ser visto como

um conjunto de práticas relacionadas à integração, ao desenvolvimento ágil, entrega contínua e qualidade de um produto.

O *DevOps* não está vinculado a nenhum tipo específico de tecnologia. Adotar as práticas *DevOps* não significa simplesmente usar um determinado conjunto de ferramentas ou estruturas para construir um *software*.

Embora o trabalho do *DevOps* não signifique usar uma caixa de ferramentas específicas, adiante será mostrado que os *Containers* podem ser uma ótima ferramenta para tornar mais fácil, a vida de um *DevOps*, pois fornecem valores adicionais às suas práticas para manter e automatizar o ambiente de produção.

Burns (2019) explica que um *container* é uma unidade padrão de *software* que empacota o código e todas as suas dependências para que o aplicativo seja executado de forma rápida e confiável de um ambiente de computação para outro.

Assim, os *containers* fornecem uma abordagem para empacotar o código-fonte do seu aplicativo, arquivos de configuração, bibliotecas e dependências em um único objeto. Com o uso de *containers* consegue-se praticamente lotear toda a capacidade computacional dos serviços.

A documentação da Google Cloud (2020) explica que o empacotamento permite que aplicativos baseados em *container* sejam implantados de maneira fácil e consistente, independentemente de o ambiente de destino ser um *data center* particular, a nuvem pública ou até mesmo o laptop pessoal de um desenvolvedor. Dessa forma, eles estão ajudando a manter o desenvolvimento ágil e garantindo entrega contínua.

Assim, os *containers* fornecem um ambiente preciso e controlado para construir uma integração contínua e *pipelines*[1] de entrega contínua. Como os *containers* são imutáveis, o *software* testado e verificado será igual ao *software* implantado.

Os *containers* costumam ser comparados com máquinas virtuais (VMs) pois tem benefícios semelhantes de isolamento e alocação de recursos, no entanto, como explica *Docker* (2019), eles funcionam de maneira diferente pois os *containers* virtualizam o sistema operacional em vez do *hardware*. Além disso, os *containers* são

[1] *Pipeline* é o processo de armazenar e enfileirar tarefas e instruções que são executadas simultaneamente pelo processador com alguma forma de organização.

mais portáteis e eficientes, pois como os *containers* compartilham o mesmo *kernel*[2] do sistema operacional, tornam-se mais eficientes do que as VMs, que exigem instâncias separadas e também mais econômicas, pois em comparação com as VMs, os *containers* exigem menos custos de *hardware* e também há economia no licenciamento de VMs e sistemas operacionais.

> Os ***containers*** são uma abstração na camada do aplicativo que agrupa o **código e as dependências**. Vários *containers* podem ser executados na mesma máquina e compartilhar o *kernel* do sistema operacional com outros contêineres, cada um executando como processos isolados no espaço do usuário. Os *containers* ocupam menos espaço do que as VMs (as imagens dos *containers* geralmente têm dezenas de MBs de tamanho), podem lidar com mais aplicativos e requerem menos VMs e sistemas operacionais.
> As **máquinas virtuais** (VMs) são uma abstração do ***hardware* físico** que transforma um servidor em vários servidores. O hipervisor permite que várias VMs sejam executadas em uma única máquina. Cada VM inclui uma cópia completa de um sistema operacional, o aplicativo, binários e bibliotecas necessários - ocupando dezenas de GBs. As VMs também podem ser lentas para inicializar. (*DOCKER*, 2019, grifo acrescido)

As vantagens do uso dos *containers* são muitas. TOZZI (2017) ao listar os benefícios importantes que ajudam a habilitar fluxos de trabalho *DevOps*, explica que os *containers* apresentam uma solução fácil para tornar os ambientes de desenvolvimento, teste e produção mais consistentes.

Diversos problemas e acontecimentos estranhos podem ocorrer quando se move o código de um ambiente de computação para outro, é um problema bastante comum no processo de *DevOps*, exigindo por vezes, mais trabalho do que se planejava ter TOZZI (2017).

Quando se escreve um código, testa e implementa dentro de *containers,* o ambiente não muda em diferentes partes da cadeia de entrega. Isso torna a colaboração entre diferentes equipes muito mais fácil TOZZI (2017).

> Os *containers* fornecem um formato padronizado para empacotamento e retenção de todos os componentes necessários para executar o aplicativo desejado. Isso resolve o problema comum de "Funciona no meu computador" e permite a portabilidade entre plataformas de SO e entre nuvens. Sempre que um container é implantado em qualquer lugar, ele é executado em um ambiente consistente que permanece inalterado de uma implantação para outra. Agora você tem um formato consistente, da caixa de desenvolvimento para a produção. (Microsoft *Azure*, 2019)

[2] Kernel é uma palavra usada na computação para denominar o núcleo do sistema operacional, que é o componente central do sistema operativo da maioria dos computadores, ele gerencia os recursos do sistema, ou seja, a comunicação entre o *hardware* e o *software.*

Outro benefício dos *containers* está relacionado à aplicação de atualizações de um aplicativo. A entrega contínua de *software* exige a implementação de atualizações de aplicativos de maneira constante e simplificada e os *containers* ajudam a fazer isso porque facilitam a aplicação de atualizações.

Quando o aplicativo é distribuído em vários *micros* serviços, cada um hospedado em um *container* separado, pode-se atualizar uma parte do aplicativo reiniciando o *container*, sem que interrompa o resto do aplicativo.

Quin (2019) explica que, com os *containers*, os desenvolvedores podem fazer pequenas alterações em elementos específicos do aplicativo, o que lhes permite mitigar o risco de impacto negativo nos negócios, reduzindo o raio de explosão em torno dessas alterações. Em caso de falha em uma única mudança há menos impacto nos serviços dependentes adjacentes.

Isso permite que se faça alterações com mais frequência para acompanhar as demandas do negócio.

TOZZI (2017) explica que é importante para um *DevOps* ter agilidade para alternar entre diferentes estruturas de programação ou plataformas de implantação e “os *containers* permitem essa agilidade porque são relativamente agnósticos em relação a linguagens de programação e plataformas de implantação”.

Pode-se executar quase qualquer tipo de aplicativo dentro de um *container*, independentemente da linguagem em que está escrito. Também é possível mover *containers* facilmente entre diferentes tipos de sistemas *host*; por exemplo, Red Hat para o Ubuntu e pode-se fazer isso rapidamente com *containers*.

Fácil escalabilidade, simplicidade, economia de custos, mudanças ágeis, permitindo que as empresas respondam rapidamente às necessidades dos clientes, ambientes consistentes, tendo maior portabilidade aos ambientes em produção, permitindo entregas mais rápidas e maior desempenho, enfim, os benefícios do uso de *containers* são muitos e embora empresas tradicionais possam ter resistência com essa nova tecnologia, o impacto que terá na transformação dos negócios não pode ser ignorado.

Neste trabalho será explorado os *containers Docker* que, como explica Monus (2018) é considerada uma das ferramentas de *DevOps* mais importantes que existem por ser a plataforma de *container* mais usada desde seu lançamento em 2013.

O *Docker* tornou a conteinerização popular no mundo da tecnologia em principal porque torna o desenvolvimento distribuído possível e automatiza a implantação de aplicativos.

2.4 Repositórios

Segundo o dicionário Michaelis (2021), repositório é um “lugar onde se guardam coisas, depósito” e essa definição pode ser transportada para o mundo da computação, sendo um local centralizado para armazenar e manter dados.

De acordo com a Documentação da Microsoft (2021) um projeto fornece um repositório para o código-fonte e um local para os usuários planejarem, acompanharem o progresso e colaborarem na construção de soluções de *software.*

Também fornece um sistema de versio*name*nto que ajuda a identificar quando determinada mudança foi feita e aplicada, tornando fácil a reversão para um estado funcional da aplicação em caso de um erro no código.

O engenheiro *DevOps* precisa ter um repositório para armazenar com segurança e gerenciar de forma fácil os códigos de suas aplicações e as imagens utilizadas em aplicações dos *containers*.

Os dois repositórios que serão explorados neste trabalho são o *GitHub* e o *Docker Hub* que são os repositórios mais usados no ramo de desenvolvimento de *softwares.*

2.4.1 *GitHub*

De acordo com a documentação do *GitHub* (2021) ele é uma plataforma de hospedagem de código para controle de versão e colaboração que permite que pessoas trabalhem de forma simultânea no mesmo projeto. O *GitHub* é subsidiário da Microsoft desde 2018.

A documentação do *GitHub* (2021) explica que um repositório contém todos os arquivos de um projeto e o histórico de revisão de cada arquivo., permitindo discutir e gerenciar o trabalho do projeto dentro do próprio repositório.

Monus (2018) explica que *GitHub* é uma das ferramentas *DevOps* mais populares, amplamente utilizada na indústria de *software*. É uma ferramenta que

oferece a funcionalidade de controle de versão distribuída e gerenciamento de código-fonte distribuído, amada por equipes remotas e colaboradores de código aberto.

O *GitHub* oferece seus serviços básicos gratuitamente, mas também possui serviços mais avançados que são pagos.

2.4.2 *Docker Hub*

Como afirma a documentação do *Docker Hub* (2021), ele é o maior repositório do mundo de imagens de *container* com uma variedade de “fontes de conteúdo, incluindo desenvolvedores de comunidades, projetos de código aberto e fornecedores de *software* independentes (ISV) construindo e distribuindo seu código em contêineres”.

Os usuários têm acesso a repositórios públicos gratuitos para armaze*nam*ento e compartilhamento de imagens, mas também tem a opção de escolha de um plano de assinatura para repositórios privados.

Assim, o *Docker Hub* é um repositório público baseado em nuvem no qual os usuários e parceiros do *Docker* criam, testam, armazenam e distribuem imagens de contêiner. Por meio do *Docker Hub*, um usuário pode acessar repositórios de imagens públicos e de código aberto, desfrutar de outras opções pagas como usar um espaço para criar seus próprios repositórios privados, funções de construção automatizadas e grupos de trabalho.

2.5 CI/CD

CI e CD são acrônimos muito usados quando se fala em práticas de desenvolvimento modernas, essas siglas fazem parte do conjunto de práticas da cultura *DevOps* que permitem que as equipes forneçam alterações de código com mais frequência e confiabilidade.

Shahin, Ali Babar, & Zhu (2017) ajuda a explicar os conceitos:

- *Continuous Integration* (CI): refere-se à integração contínua de todo trabalho em progresso, com testes de um novo código-fonte (*branch*) ao principal (*main*), permitindo frequentes ajustes nos códigos e entregas rápidas. Essa integração contínua (CI) permite que vários desenvolvedores trabalhem em

diferentes recursos ou módulos do mesmo aplicativo e envie suas atualizações individualmente para um repositório de código compartilhado, de forma que toda equipe tenha feedback constante dos desenvolvimentos.

Assim, como afirma RedHat (2020), “uma CI bem-sucedida é quando novas mudanças no código de uma aplicação são desenvolvidas, testadas e consolidadas regularmente em um repositório compartilhado” e, explica que, as mudanças são consolidadas e depois, validadas através da criação automática da aplicação. Vários testes automatizados, geralmente de unidade e integração, são feitos para garantir que as mudanças não corrompam a aplicação.

- *Continuous Delivery* (CD): refere-se à entrega ou distribuição contínua, uma prática que garante que o código compilado esteja pronto para ser disponibilizado nos diversos ambientes de produção. A Amazon (2021) em seu *site*, explica que a entrega contínua “expande com base na integração contínua implantando todas as alterações de código em um ambiente de teste ou ambiente de produção, após o estágio de criação”.

  RedHat (2020) complementa explicando que depois de realizar a automação de compilação e da unidade e os testes de integração na CI “a entrega contínua automatiza o lançamento desse código validado em um repositório”. Com isso, para ter um processo eficaz de entrega contínua, é importante que a CI já esteja integrada ao pipeline de desenvolvimento.
- *Continuous Deployment* (CD): refere-se à implantação contínua, a etapa final de um pipeline necessita de uma implementação contínua, que, como explica RedHat (2020) “automatiza o lançamento de compilações prontas para produção em um repositório de códigos”.

> *Deploy* contínuo, também conhecido como CD (*continuous Deployment*) é uma prática de desenvolvimento de *software* na qual cada alteração de código passa por todo o pipeline e é colocada em produção automaticamente, resultando em muitas implantações de produção todos os dias. Faz tudo o que a Entrega Contínua faz, mas o processo é totalmente automatizado, não há intervenção humana. (Faraday, 2019)

Ou seja, como não há um canal manual na etapa do *pipeline* antes da produção, a implantação contínua irá depender muito da automação otimizada dos testes realizados.

Assim, o *Continuous Integration* íntegra o que está sendo trabalhado e tanto o *Continuous Delivery* como o *Continuous Deployment* abordam a etapa de *Deploy* para a Produção, com a diferença de que, no *Continuous Delivery* se faz o *Deploy* quando há um comando manual para o *Deploy* ocorrer e o *Continuous Deployment* ocorre de forma automática, passando nos testes já será feito o *Deploy*.

Os quadros a seguir disponibilizados pelo RedHat (2020) ilustram a explicação:

Figura 2 – Abordagem CI/CD

Fonte: RedHat, 2020.

O *DevOps* trabalhando com a integração, entrega e implantação contínuas permite que as empresas aproveitem a automação para desenvolver, construir, testar e implantar um código de alta qualidade mais rapidamente, permitindo que as empresas respondam às mudanças do mercado com mais facilidade e rapidez o que têm correlação com o sucesso dos negócios.

2.6 *Ansible*

Ansible é uma ferramenta, adquirida em 2015 pela empresa RedHat, escrita em *Python*, de código aberto, de provisio*name*nto de *software* (processo de deixar o servidor pronto para operação) que, de acordo com o próprio RedHat (2021) serve para gerenciar, automatizar, configurar servidores, implementar serviços entre outras funções da área de TI, a partir de uma localização central.

Existem várias ferramentas de gerenciamento de configuração como *Chef*, *Puppet* e *Terraform*. Neste trabalho explora-se apenas sobre o *Ansible*, por ser uma das ferramentas mais populares de acordo com Gill (2020), que fornece uma linguagem de configuração limpa, sendo considerada a maneira mais simples de

automatizar o fornecimento, configuração e gerenciamento de aplicativos e infraestrutura de TI.

Nallamala (2019) também acrescenta que o *Ansible* é uma das ferramentas *IaC* (*Infrastructure as Code*), mais flexíveis do mercado atualmente, pois o usuário não se limita aos recursos que o *Ansible* oferece, possibilitando que se desenvolva módulos próprios, plug-ins e rotinas para atender a necessidades específicas.

Ansible como uma ferramenta *IaC* "*Infrastructure as Code*", refere-se ao processo de gerenciamento e provisio*name*nto automático de tecnologia por meio do código-fonte de um aplicativo, em vez da configuração manual de dispositivos. Como explica a documentação da Microsoft:

> Sem *IaC*, as equipes devem manter as configurações de ambientes de implantação individuais. Com o tempo, cada ambiente se torna um floco de neve, ou seja, uma configuração única que não pode ser reproduzida automaticamente. A inconsistência entre os ambientes leva a problemas durante as implantações. Com flocos de neve, a administração e manutenção da infraestrutura envolve processos manuais difíceis de rastrear e que contribuíam para erros. (MICROSOFT, 2021)

Assim, como explica a documentação da Microsoft a *IaC* permite que as equipes de *DevOps* usem diferentes ferramentas e abordagens para controlar e personalizar automaticamente a infraestrutura necessária, em vez de configurar manualmente os servidores e sistemas operacionais. Gill (2021) afirma que a *IAC* é uma das práticas de *DevOps* mais importantes usadas em conjunto com CD-entrega contínua.

Ao aprofundar sobre a importância da ferramenta *IaC Ansible*, Alura (2021) por meio de uma publicação de Alexandre Jardim, especialista de *DevOps*, discutindo os motivos do engenheiro *DevOps* adotar uma ferramenta como o *Ansible*, relembra que:

> O *DevOps* busca maneiras de encurtar o seu ciclo de feedbacks entre os times de desenvolvimento e estrutura, tornar o processo com repetições tanto do lado da infraestrutura - como já é feito no lado do desenvolvimento. Se se pensar em uma forma simples de definir *DevOps*, trata-se de levar a cultura ágil para a cultura de infraestrutura e integrá-las...Cada cultura organizacional entenderá de qual é a melhor forma de aplicar os princípios de *DevOps*. Ao se falar de *DevOps*, conseguir-se-á automatizar tudo aquilo que é feito na infraestrutura e garantir o estado dos servidores. ALURA (2021)

Olhando para o passado, os computadores eram utilizados para realizar uma gama limitada de tarefas – o profissional de Infraestrutura criava diversos *scripts* para modificar um ambiente ou fazia configurações manualmente. No entanto, o poder da computação aumentou e, como resultado, um administrador de sistema teve que gerenciar dezenas, centenas ou até milhares de dispositivos e aí que a situação se complicaria, quando se fazia necessário configurar centenas de servidores e garantir da mesma forma os estados das máquinas. O *Ansible* é uma ferramenta que garante esse mesmo estado das máquinas.

Kelly (2020) explica que, quando uma configuração é escrita para um aplicativo como o *Ansible*, basta adicionar os servidores e as variáveis ao arquivo de configuração e, em seguida, vários servidores podem ser configurados com um comando.

Ao explorar mais a fundo o funcio*name*nto do *Ansible*, a documentação da *Open Source Initiative* (2021) explica de forma simples, que, no *Ansible*, existem duas categorias de computadores: o nó de controle e os nós gerenciados.

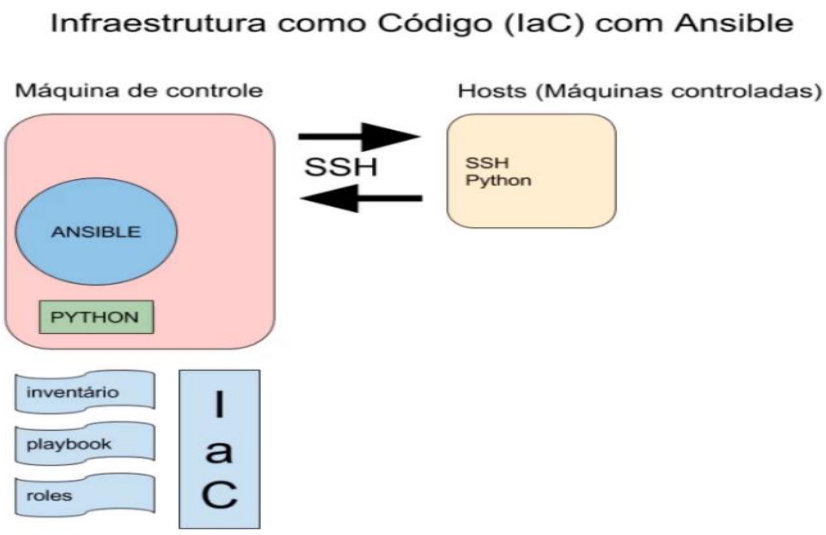
O nó de controle é um computador que executa o *Ansible* e deve haver pelo menos um nó de controle, embora também possa existir um nó de controle de *backup*. Um nó gerenciado é qualquer dispositivo gerenciado pelo nó de controle.

O *Ansible* funciona conectando-se a nós (clientes, servidores ou o que quer que esteja sendo configurado) em uma rede e, em seguida, envia um pequeno programa chamado módulo *Ansible* para esse nó. O *Ansible* executa esses módulos por SSH (*Secure Socket Shell*) e os remove quando concluído. O único requisito para essa interação é que o nó de controle *Ansible* tenha acesso de *login* aos nós gerenciados. As chaves SSH são a forma mais comum de fornecer acesso, mas outras formas de autenticação também são suportadas.

Bicca (2016) explica que o *Ansible* trabalha com arquivos *Yaml* e, o principal arquivo de configuração é chamado de *Playbook*, é o arquivo mais elementar do *Ansible*, onde se coloca todas as tarefas que serão executadas *“yum”,”mkdir”, ”useradd*” e no arquivo *.ini* são adicionados os servidores onde o *Ansible* irá executar essa *Playbook*. Cada um deles possui *roles*, que são as informações de provisio*name*nto, que permitem modularizar o código. Após definir as *roles*, será definido em quais *hosts* essas *roles* serão executadas.

A figura 3 ilustra uma descrição simplificada da ferramenta *Ansible*:

Figura 3 – Apresentação do *Ansible*



Fonte: Alura, 2021.

A máquina de controle é onde o servidor *Ansible* será executado, podendo até ser um servidor remoto que aceita a instalação do *Ansible*. Há os *host*s que devem ser controlados. O *Ansible* não é dependente da instalação nas máquinas que devem ser configuradas. Ambas as máquinas necessitam de uma conexão SSH - *Secure Shell* funcional e o *Python* instalado. A alteração do estado da máquina é feita pela chave SSH - *Secure Shell* , sem que seja necessário entrar na máquina e di*Git*ar algum comando, o *Ansible* fará isso. O *DevOps* ficará responsável por desenvolver os scripts que o *Ansible* usa.

Com o *Ansible* pode-se realizar uma série de tarefas que, sem ele, seriam demoradas, complexas, repetitivas e que poderiam vir com muitos problemas ou erros. A StarAgile (2020) lista algumas das principais tarefas que o *Ansible* pode fazer:

- Gerenciamento de configuração - as informações de *hardware* e *software* da empresa são registradas e atualizadas em detalhes, mantendo assim a consistência do desempenho do produto.
- Implementação de aplicativos - Os aplicativos podem ser gerenciados no *Ansible* do desenvolvimento à produção quando se define e gerencia os aplicativos usando o *Ansible*.
- Orquestração - Para gerenciar como um todo e como as configurações interagem.

- Segurança e conformidade - uma política de segurança ampla pode ser implantada em toda a infraestrutura quando a política é definida no *Ansible*
- Provisio*name*nto - Ajuda a automatizar e gerenciar o processo

São muitas as funcionalidades que o uso da ferramenta *Ansible* proporciona que juntas fornecem grandes ganhos de produtividade onde as equipes de TI e principalmente os *DevOps* podem se beneficiar muito do seu uso, superando os desafios de automação que surgirem.

2.7 Processos (*Deploy* e *Rollback*)

Há dois processos rotineiros no mundo do desenvolvimento de *software* que são o *Deploy* e *Rollback*. Como explicado nos tópicos anteriores, o *DevOps* é o profissional que permite a manutenção, a implantação (iniciar o desenvolvimento) e a implementação (colocar em prática) de forma mais rápida e segura, aplicações, sejam elas localizadas em um ambiente local, na própria máquina do desenvolvedor ou dentro de um servidor, com todas as funcionalidades planejadas ou solicitadas pelo cliente.

Da mesma forma, é também o profissional de *DevOps* que se preocupa, quando ocorre uma falha, em fazer um plano de reversão das implementações que geraram a falha na aplicação. E, de forma bem simplificada, é disso que se trata os processos de *Deploy* (implantar) e *Rollback* (reverter) que serão explicados neste tópico.

Ao explorar o significado do termo "*Deploy*" *Host*Gator (2020) explica que *Deploy* é um termo muito usado entre os desenvolvedores de *software* e ele pode significar muitas coisas a depender do ambiente e da tecnologia, porém "os significados que condizem mais com a prática e podem resumir o seu funcio*name*nto são: implantar, colocar em posição, disponibilizar para uso ou simplesmente colocar no ar".

Ainda de acordo com *Host*Gator (2020) a prática de *Deploy* evoluiu muito ao longo das décadas. No começo, os *Deploys* eram feitos em ritmo menos acelerado, principalmente porque este processo era feito basicamente de forma manual. Com o

avanço tecnológico, tudo é feito de forma mais automática, o que melhora a experiência do usuário e agiliza a obtenção dos resultados.

2.7.1 Formas de realizar *Deploy*

Como explicado por *Host*Gator (2020) anteriormente, os processos de *Deploys* evoluíram muito durante o tempo e atualmente, com a adoção da cultura *DevOps*, quanto mais os processos *Deploy* foram feitos de forma confiável e mais automática mais se otimizará o tempo médio necessário para levar o produto ao cliente, sendo o objetivo principal do *Continuous Delivery*. Mas o modo automático não é a única forma de se realizar um *Deploy*, há basicamente três formas possíveis de se fazer um *Deploy*.

Deploy Manual:

Como explica a documentação da IBM (2021) a configuração manual normalmente é realizada quando se tem mais controle do ambiente do servidor de aplicativos se faz necessário. No *Deploy* manual é necessário que o desenvolvedor realize todos os principais comandos.

*Host*Gator (2020) dá um exemplo bastante comum do uso de *Deploy* manual que é o *FTP* ou Protocolo de Transferência de Arquivos, que se refere à um tipo de conexão que permite que dois computadores com acesso à *internet* troquem arquivos, neste caso, o *Deploy* é realizado de forma manual pois necessita de um comando de uma pessoa. Outro exemplo é quando um desenvolvedor faz um rápido ajuste no código para logo subi-lo para o ambiente de produção.

Etmajer (2014) lista algumas desvantagens do *Deploy* manual como o fato de serem lentas e não repetíveis, risco de conflitos (por exemplo, se mais de um desenvolvedor subir arquivos ao mesmo tempo pode vir a causar um conflito na aplicação da melhoria), implantações manuais requerem documentação extensa (que geralmente é desatualizada), outro fato é que as implantações manuais não são consistentes em todos os ambientes.

Deploy Parcialmente Automatizado:

Neste *Deploy* uma parte do processo não precisa ser realizada pelo desenvolvedor. Um exemplo deste *Deploy* seria a atualização de um repositório *Git*. *Host*Gator (2020) explica que o *push do branch master* que se faz no repositório *Git* que roda um pequeno *hook*, atualiza o servidor de hospedagem, ou seja, apesar de ser necessário um comando, a atualização do sistema é totalmente automática.

As vantagens do *Deploy* parcialmente automático, como explica *Host*Gator (2020) são o controle de versão e o estado de cada *Deploy*.

Deploy Completamente Automatizado:

Este *Deploy* seria o que há de mais moderno atualmente. Com este *Deploy*, segundo Roveda (2021) é possível garantir mais segurança, qualidade e eficiência na hora de realizar atualizações e melhorias nas aplicações, *softwares* ou *sites*.

Com este *Deploy* é possível transmitir as atualizações automaticamente para o servidor e realizar o *Continuous Integration*, ou seja, este *Deploy* não apenas copia as alterações automaticamente para o servidor, ele une diversas alterações de código – permitindo integrações, em um repositório central, automaticamente, com o objetivo de fazer testagens e execuções antes do *Deploy* final.

*Host*Gator (2020) ao listar as vantagens do *Deploy* totalmente automatizado explica que a ferramenta de *Deploy* “realiza todos os testes necessários para que não haja problemas para juntar todas as integrações em produção. E as possibilidades são infinitas, desde: atualizar bibliotecas; testar conectividade de servidores e simular visitas e entradas de dados”. E se por algum motivo, algo der errado, o *Deploy* é revertido.

No mercado atualmente existem ferramentas que tornam isso possível, voltadas à automatização total do processo de *Deploy*, entre as mais populares, de acordo com Roveda (2021) estão: *Jenkins*, *GitLab*, *Azure Pipelines* e *Circle CI*.

2.7.2 Estratégias para se realizar *Deploy*

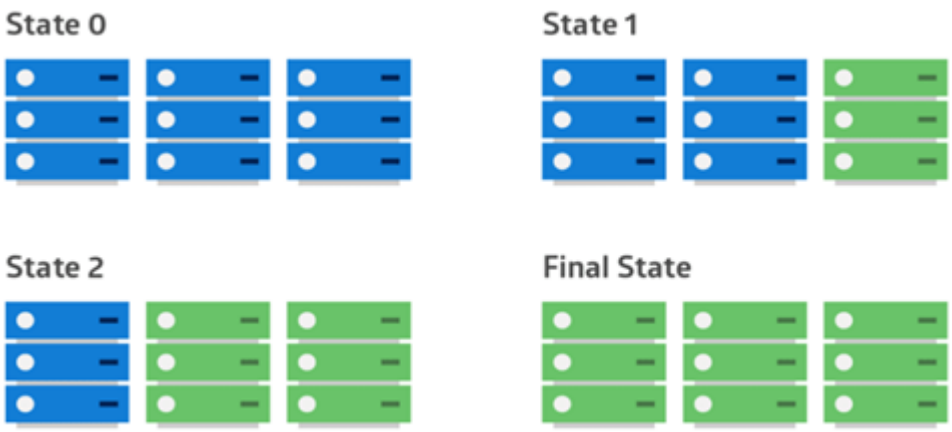
Jachja (2021) explica que há estratégias práticas usadas para alterar ou atualizar uma instância em execução de um aplicativo, ou seja, para se executar o *Deploy* e as mais populares são:

Rolling: essa estratégia consiste, de acordo com Jachja (2021) em substituir lentamente as instâncias atualmente em execução do aplicativo por outras mais recentes. Ou seja, duas versões de uma mesma aplicação coexistem enquanto o *Deploy* é executado, substituindo totalmente o código fonte da aplicação pelo novo código.

*Host*Gator (2020) explica que a versão antiga é substituída “apenas quando a nova estiver apta para ser executada. Logo, o *Deploy* é feito gradualmente e, enquanto a nova versão não está 100% completa, elas coexistem em produção”.

Jachja (2021) ao listar os contras desta estratégia explica que como os nós são atualizados em lotes, esta estratégia exige serviços para oferecer suporte a versões novas e antigas. A verificação da implantação de um aplicativo a cada mudança incremental também torna essa implantação lenta. A figura 4 ilustra esta estratégia:

Figura 4 – Estratégia Rolling



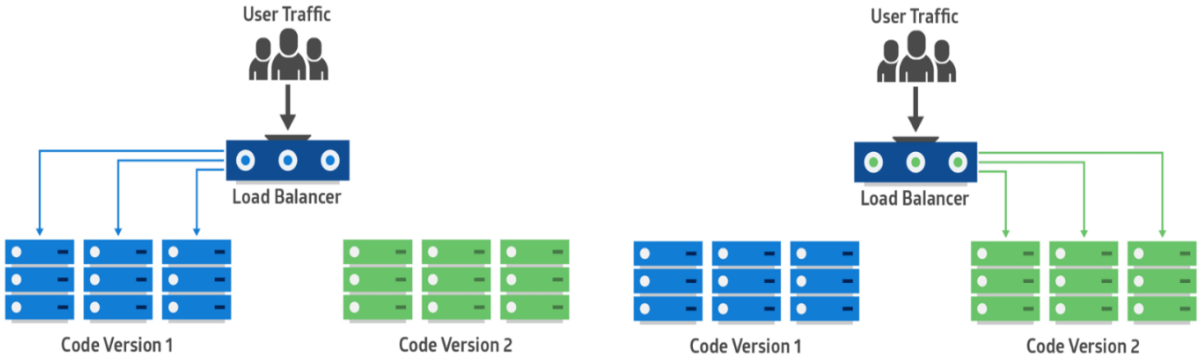
Fonte: Skowronski, 2018.

Blue-Green: essa estratégia, como explica *Host*Gator (2020) “permite realizar os testes em produção, antes que os usuários tenham acesso ao serviço” nesta estratégia há dois ambientes idênticos (*mirror*). Um é chamado de *blue (azul)*, representando o ambiente de teste e o outro ambiente é chamado de *green* (verde) representando o ambiente de produção, são ambientes com diferentes versões de um aplicativo ou serviço. Os testes e atualizações são normalmente feitos no ambiente *blue (azul)* que hospeda novas versões ou mudanças. Existe o *load balancer* que permite o direcio*name*nto do tráfego para o ambiente desejado. O tráfego do usuário é deslocado do ambiente *blue (azul)* para o green (verde), uma vez que novas

alterações são testadas *e aceitas* no ambiente verde. Se houver um problema depois que o verde ficar ativo, o tráfego pode ser roteado de volta para o azul.

Jachja (2021) ao listar os contras desta estratégia explica que a replicação de um ambiente de produção pode ser complexa e cara, especialmente ao trabalhar com micros serviços. Apresentado na figura 5.

Figura 5 - Estratégia Blue-Green



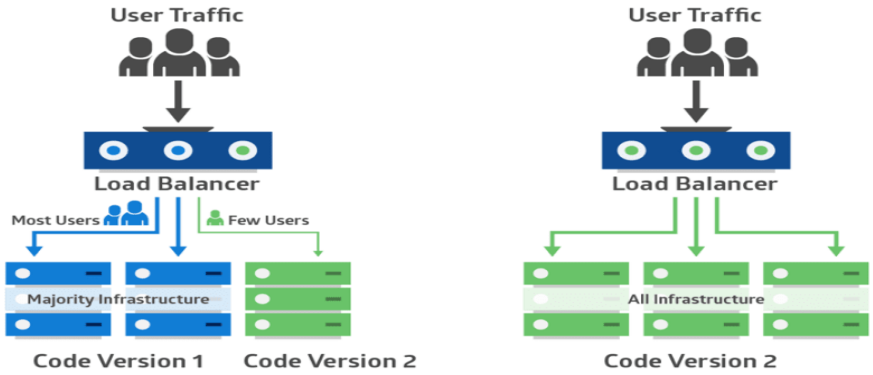
Fonte: Skowronski, 2018.

Canary: esta estratégia é a mais complexa de ser adotada, como explica *Host*Gator (2020) consiste em colocar uma nova versão em produção para apenas uma parcela de usuários, como teste inicial. Por meio desta estratégia é possível, por exemplo, liberar que determinado serviço só será acessado por usuárias do sexo feminino que tenham menos de 20 anos e acompanhar a reação deste público antes de liberar o acesso para todos.

A ideia então é primeiramente implantar a mudança em um pequeno subconjunto de servidores, testá-lo e, em seguida, sem erros relatados, implantar a mudança no restante dos servidores.

Skowronski (2018) explica que o principal desafio da implantação canário é conceber uma maneira de rotear alguns usuários para o novo aplicativo. Além disso, alguns aplicativos podem sempre precisar do mesmo grupo de usuários para teste, enquanto outros podem exigir um grupo diferente a cada vez. Além disso, há o custo alto para manter várias instâncias do servidor. Apresentado na figura 6.

Figura 6 - Estratégia Canary



Fonte: Fonte: Skowronski, 2018.

As implantações de *software* e aplicativos podem e devem ser diferentes a depender do projeto que está sendo trabalhado. Como foi visto, é possível realizar o *Deploy* substituindo lentamente uma versão (A) pela outra (B), é possível lançar uma versão (A) juntamente com uma versão (B) e, em seguida, alterar o tráfego para a versão (B) e ainda, é possível lançar uma versão para um subconjunto de usuários e, em seguida, prosseguir para uma implementação completa, disponibilizando para todos os usuários.

Estas são as principais estratégias, e escolher a estratégia certa, é uma decisão importante, pesando as opções em termos do impacto da mudança no sistema e nos usuários finais.

2.7.3 *Estratégias para se realizar Rollback*

O termo *Rollback*, refere-se à sua tradução literal, “reversão”, como explica Aloia e Ribeiro (2017), significa descartar todas as alterações (*updates, deletes*, *inserts*) realizadas.

A cultura *DevOps*, as práticas CI/CD e as estratégias de *Deploy* podem tornar as implantações muito mais seguras e eficientes, mas não eliminam o risco da necessidade de um *Rollback*.

Por mais que se trabalhe para que um código funcione perfeitamente, há situações em que os principais *bugs* são descobertos em uma versão depois de ser implementada em um ambiente. Nesses casos, há a necessidade de se fazer um

Rollback da implantação a um estado anterior, enquanto se trabalha na correção dos problemas.

Em um artigo, Srivatsav (2017) comenta que há um medo na comunidade *DevOps* quando se fala sobre *Rollback* e geralmente a comunidade não entende muito bem a necessidade de se ter um plano para reverter a implantação de uma aplicação que venha a apresentar falhas.

Com a necessidade de implantações cada vez mais frequentes, é provável que o código implantado possa apresentar falhas, afetando negativamente o uso, a confiabilidade na aplicação ou a experiência do cliente. É por isso que é importante, como afirma Srivatsav (2017), implementar uma arquitetura que suporte a necessidade de reversão.

Da mesma forma, a documentação do Redgate (2021) explica que ao fazer o planejamento de contingência para os *Deployments*, é importante considerar um cenário no qual o *Rollback* seja necessário.

Com o código do aplicativo, Redgate (2021) explica que reverter para uma versão anterior é bastante simples: pode-se simplesmente substituir os artefatos implantados pela versão de trabalho anterior. Já os *Rollbacks* de banco de dados são difíceis devido à natureza do armaze*name*nto persistente: os dados em seu esquema devem ser preservados ao avançar para permitir que um estado anterior seja restaurado em uma data posterior (se um *backup*/restauração simples de banco de dados não for uma opção). Assim, o planejamento de *Rollback* no caso de falha depende inteiramente das alterações feitas no código do aplicativo e no banco de dados.

Ainda sobre o planejamento de *Rollback* Robete (2016) explica que este planejamento deve ter um documento escrito que descreve todas as etapas necessárias para recuperar o banco de dados ou o aplicativo a sua versão anterior desde o início do *Deployment.* A autora ainda explica que este planejamento deve ser testado, para determinar todos os componentes do *Rollback*. Além disso, precisa conter a localização dos scripts que precisam ser executados e conter o nome do banco de dados e o usuário que precisa executar os scripts

Assim, ter um plano de *Rollback* pode ajudar a esclarecer qual impacto que a reversão terá em outros sistemas e quais outras etapas devem ser executadas.

3 APLICAÇÃO

Esse capítulo apresentará uma descrição detalhada de como a aplicação irá se comportar e como ela será criada.

3.1 Resumo da aplicação

Como descrito anteriormente, para demonstrar a implementação da cultura *DevOps* será utilizado uma aplicação de código aberto de um sistema de votação retirado do repositório do *GitHub* (2021).

Nessa aplicação, o usuário poderá votar a partir de uma interface *web* (desenvolvido utilizando a linguagem de programação *Python*) entre a escolha de um *dog* ou *cat*, o resultado dessa votação será exibido em uma outra interface *web* (desenvolvido utilizando a linguagem de programação *JavaScript*).

Para que esse resultado da votação seja exibido na interface *web* final, primeiro o usuário deve acessar a interface *web* de votação e escolher entre as duas opções descritas anteriormente. Após confirmar a escolha, a informação será armazenada em um banco de dados chamado *Redis* (banco de dados temporário), essa informação por sua vez será consumida por um *worker* (desenvolvido utilizando a linguagem de programação .NET) e persistida em um outro banco de dados denominado *Postgres* (banco de dados utilizado para persistir os dados da votação).

Com os dados da votação já persistidos no *Postgres*, a interface *web* que mostra os resultados da votação, irá consumir os dados do *Postgres* e mostrar na tela.

Ressaltando que o foco do trabalho não é a aplicação, mas sim os processos relacionados à cultura *DevOps*. Desde a criação da máquina virtual na *AWS* até a execução do *continuous integration*, *Deploys* e *Rollbacks*.

Para que isso seja possível, será criada uma infraestrutura para suprir as demandas necessárias para prover todos os conceitos relacionados a cultura *DevOps* citados neste trabalho, com a utilização da ferramenta *Ansible* serão criados os códigos computacionais para realizar a instalação, *Deploy*, *Rollback*, criação das imagens *Docker*, envio das imagens *Docker* para um repositório do *Docker Hub*, criação das *pipelines* de *continuous integration* e *continuous Deployment/delivery*.

Utilizando a ferramenta *Kubernetes* para orquestração dos *containers* e dos serviços para comunicação interna da aplicação, gerenciamento dos processos de *Deploys* e *Rollbacks*, iniciados a partir de um comando do *Ansible*.

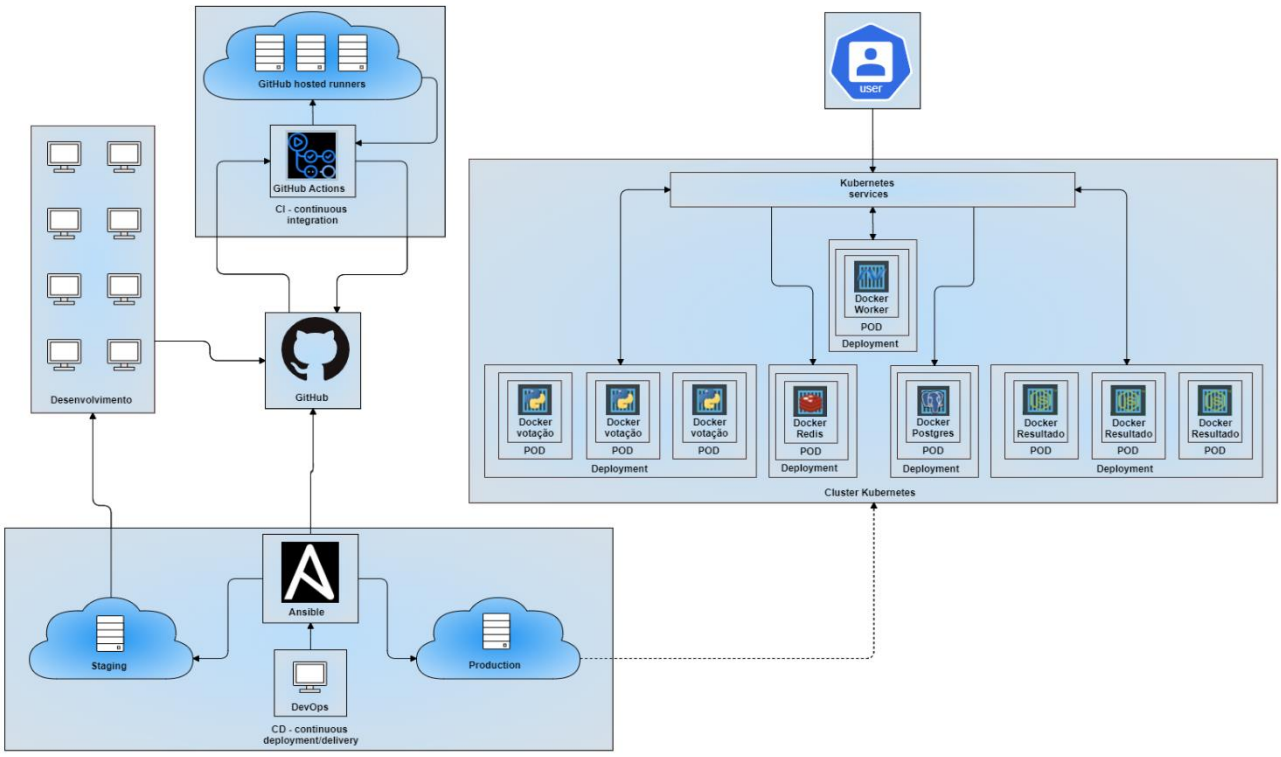
Será utilizado a ferramenta *GitHub Actions* para gerenciar os processos de CI.

Todos os códigos utilizados pela aplicação são códigos abertos. Será criado um repositório no *GitHub* para armazená-los para maior organização, o *Ansible* irá utilizar os códigos desse repositório para orquestrar todos os passos descritos acima.

Para fornecer uma maior disponibilidade da aplicação, serão fornecidas três instâncias da aplicação *web* para coletar os votos e três instâncias da aplicação *web* que irão mostrar os resultados, isso é feito para distribuir a carga de acesso caso ocorra vários acessos simultâneos o que pode vir a causar lentidão ou até mesmo indisponibilidade da aplicação.

A seguir, há um esboço apresentado na figura 7 de como será construído a infraestrutura para hospedar a aplicação e suportar todos os conceitos da cultura *DevOps* descritos neste trabalho.

Figura 7 – Infraestrutura da Aplicação de Votação



Fonte: Elaborado pelo autor (2021)

3.2 Criação dos recursos computacionais

Essa sessão descreve a criação dos recursos computacionais que serão utilizados.

3.2.1 Servidor na nuvem

Esse trabalho utiliza o recurso da nuvem da *AWS* denominado EC2 (*Elastic Compute Cloud*) como servidor para executar o *cluster* de *Kubernetes*. Será instalado o sistema operacional Ubuntu 18.04 nesse servidor, para configurações de *hardware* do servidor será utilizado uma instância EC2 do tipo *m3.large* como recomendado na documentação do *Kubernetes*, essa instância possui 2 núcleos de processadores (CPU), 7.5 *gigabytes* de memória RAM e um disco rígido do tipo SSD (*solid disk driver*) de 32 gigabytes de espaço para armaze*name*nto.

Para que seja possível a comunicação para execução de comandos computacionais no servidor, será criado uma chave de acesso SSH-*Secure Socket Shell* que será utilizada para habilitar uma comunicação segura entre o servidor e o engenheiro responsável por sua configuração. Essa chave será um arquivo criptografado nomeado “tcc.pem” e será utilizado posteriormente neste trabalho para realizar as ações mencionadas anteriormente.

3.2.2 Repositório de código

Para armazenar os códigos computacionais utilizados neste trabalho, será criado um repositório no *GitHub* chamado “tcc-*DevOps*” que irá conter além dos códigos computacionais públicos que serão utilizados para executar a aplicação de votação (https://*GitHub*.com/kodekloudhub/example-*voting-app*), também irá armazenar os códigos computacionais utilizados para gerenciar o *cluster* de *Kubernetes* e os códigos computacionais utilizados pelo *Ansible* para configurar o servidor e executar os comandos para orquestração da aplicação, tanto no seu momento inicial de criação, quanto no seu momento de execução.

No repositório criado acima, será criado também a *pipeline* de execução do processo de CI e CD, que será iniciada a partir de uma atualização do código

computacional, executando uma série de testes, que caso sejam bem-sucedidos, o processo de *Build* da imagem *Docker* da aplicação será construída e enviada para o repositório do *Docker Hub*.

3.2.3 Repositório de imagens de *containers*

O armaze*name*nto das imagens dos contêineres geradas a partir do processo de CI na *pipeline* do *GitHub Actions*, é uma parte importante do processo de resiliência da aplicação, pois em caso de problemas ou *bugs,* o rastreamento da versão da imagem do container utilizada pela aplicação no ambiente de produção. Com todas as imagens armazenadas no repositório do *Docker Hub*, pode-se facilmente escolher qual versão da aplicação deseja se realizar o *Deploy* ou *Rollback* no ambiente de produção.

A utilização do *Docker Hub* como repositório para armaze*name*nto das imagens dos contêineres da aplicação é interessante para a cultura *DevOps* pois o *Docker Hub* armazena as imagens *Docker* de *containers* em *layers* (camadas), o que acaba agilizando o processo de geração e uso das imagens de *containers*, pois uma vez baixado a primeira imagem da aplicação, caso uma nova imagem com poucas mudanças no código computacional seja gerada, no momento de enviar e baixar essa imagem do *Docker Hub*, o processo será rápido pois somente algumas camadas dessa imagem foram alteradas, ou seja, ao baixar ou enviar uma imagem para e repositório do *Docker Hub*, somente as alterações feitas precisaram ser enviadas, enquanto as demais (redundantes) podem ser reaproveitadas das outras imagens de contêineres previamente geradas.

Para este trabalho será criado um repositório nomeado “tcc*DevOps*”, dentro dele serão armazenadas as imagens geradas para as três aplicações que serão utilizadas: *result, vote* e *worker*.

A comunicação entre o *cluster* de *Kubernetes* e o repositório do *Docker Hub* precisa ser previamente configurado, caso contrário a aplicação não conseguirá baixar e utilizar a imagem, causando assim uma falha geral na aplicação. Para que essa comunicação ocorra, possibilitando o *cluster* de *Kubernetes* se comunicar com o *Docker Hub*, será criado um arquivo denominado *Docker-login-yaml*, ele irá conter as

credenciais de acesso para acessar o *Docker Hub* e baixar as imagens necessárias para executar a aplicação.

3.2.4 Configurações do *Kubernetes*

Primeiramente o *Kubernetes* precisa ser instalado em um servidor, para isso será criado uma *playbook* de instalação no *Ansible*, que será explicado posteriormente neste trabalho.

Após instalado, o *Kubernetes* pode ser configurado e orquestrado a partir de instruções de códigos computacionais utilizando a linguagem *YAML*. Esses arquivos podem ser escritos para gerar qualquer recurso disponível no *Kubernetes*, neste trabalho irei utilizar dois tipos de recursos: *Deployments* e *services*.

- *Deployments*: Um recurso do tipo *Deployment* descreve um estado desejado de uma aplicação a partir de um arquivo *YAML* de configuração, nesse arquivo é possível configurar o número de réplicas que a aplicação irá possuir, etiquetar o recurso criado pela aplicação, configurar qual imagem de container essa aplicação irá utilizar, qual porta essa aplicação irá abrir para se comunicar com o ambiente fora de seu escopo, quais a variáveis de ambientes essa aplicação possui além de vincular recursos do *Kubernetes* entre eles mesmos, através da configuração de um *Selector*;
- *services*: Um recurso do tipo *service* é uma maneira abstrata de expor uma aplicação como um serviço de rede. Esse tipo de recurso possibilita que não seja necessário alterar o código da aplicação para expor em uma rede, basta vincular esse recurso com o recurso que se deseja expor na rede através de um *Selector*, após essa configuração a aplicação está disponível para ser acessada a partir da rede.

Neste trabalho será realizado uma configuração para vincular cinco recursos de *Deployment* com cinco recursos de *service*. Esses recursos serão criados com a seguinte nomenclatura:

- *Postgres-Deploy.yaml*: Esse arquivo irá conter as configurações de *Deployment* do *Postgres*, essa configuração irá prover uma instância do

banco de dados *Postgres* que será utilizada pela aplicação para armazenar o resultado da votação. Esse arquivo será configurado da seguinte forma:

- Nome do container: *Postgres*;
- Imagem de container que será utilizada para o *Deploy*: *Postgres*;
- Porta: 5432;
- Variável de ambiente: *POSTGRES*_USER;
- Variável de ambiente: *POSTGRES*_PASSWORD;
- Etiqueta: *name*;
- Etiqueta: *app*;
- *Selector*: *name*;
- *Selector*: *app*.

O código com a implementação completa se encontra no Apêndice A localizado na página 67.

- *Redis-Deploy.yaml*: Esse arquivo irá conter as configurações de *Deployment* do *Redis*, essa configuração irá prover uma instância do banco de dados *Redis* que será utilizada pela aplicação para capturar temporariamente os resultados da votação. Esse arquivo será configurado da seguinte forma:
  - Nome do container: *Redis*;
  - Imagem de container que será utilizada para o *Deploy*: *Redis*;
  - Porta: 6379;
  - Etiqueta: *name*;
  - Etiqueta: *app*;
  - *Selector*: *name*;
  - *Selector*: *app*.

  O código com a implementação completa se encontra no Apêndice B.
- *result-app-Deploy.yaml*: Esse arquivo irá conter as configurações de *Deployment* da aplicação *result*, essa configuração irá prover uma instância da aplicação *result* que será utilizada para mostrar os resultados da votação. Esse arquivo será configurado da seguinte forma:
  - Nome do container: *result-app*;

    - Imagem de container que será utilizada para o *Deploy*: tcc*DevOps*/*result*:*tagname* (*tagname* é um valor mutável que na prática irá mostrar a versão da aplicação de *result*);
    - Porta: 80;
    - Etiqueta: *name*;
    - Etiqueta: *app*;
    - *Selector*: *name*;
    - *Selector*: *app*.

O código com a implementação completa se encontra no Apêndice C.

- *voting-app-Deploy.yaml*: Esse arquivo irá conter as configurações de *Deployment* da aplicação *voting*, essa configuração irá prover uma instância da aplicação *voting* que será utilizada para inserir os votos na aplicação de votação. Esse arquivo será configurado da seguinte forma:
    - Nome do container: *voting-app*;
    - Imagem de container que será utilizada para o *Deploy*: tcc*DevOps*/*vote*:*tagname* (*tagname* é um valor mutável que na prática irá mostrar a versão da aplicação de *voting*);
    - Porta: 80;
    - Etiqueta: *name*;
    - Etiqueta: *app*;
    - *Selector*: *name*;
    - *Selector*: *app*.

O código com a implementação completa se encontra no Apêndice D.

- *worker-app-Deploy.yaml*: Esse arquivo irá conter as configurações de *Deployment* da aplicação *worker*, essa configuração irá prover uma instância da aplicação *worker* que será utilizada para inserir os votos na aplicação de votação. Esse arquivo será configurado da seguinte forma:
    - Nome do container: *worker-app*;
    - Imagem de container que será utilizada para o *Deploy*: tcc*DevOps*/*worker*:*tagname* (*tagname* é um valor mutável que na prática irá mostrar a versão da aplicação de *worker*);
    - Etiqueta: *name*;
    - Etiqueta: *app*;

    - *Selector*: *name*;
    - *Selector*: *app*.

O código com a implementação completa se encontra no Apêndice E.

- *Postgres*-service.*yaml*: Esse arquivo irá conter as configurações de *service* da aplicação *Postgres*, essa configuração irá prover uma abstração de rede para o banco de dados *Postgres* que será utilizada pela aplicação para habilitar a comunicação via rede entre os demais serviços e o *Postgres*. Esse arquivo será configurado da seguinte forma:
    - Porta: 5432;
    - *TargetPort*: 5432;
    - Etiqueta: *name*;
    - Etiqueta: *app*;
    - *Selector*: *name*;
    - *Selector*: *app*.

O código com a implementação completa se encontra no Apêndice F.

- *Redis*-service.*yaml*: Esse arquivo irá conter as configurações de *service* da aplicação *Redis*, essa configuração irá prover uma abstração de rede para o banco de dados *Redis* que será utilizada pela aplicação para habilitar a comunicação via rede entre os demais serviços e o *Redis*. Esse arquivo será configurado da seguinte forma:
    - Porta: 6379;
    - *TargetPort*: 6379;
    - Etiqueta: *name*;
    - Etiqueta: *app*;
    - *Selector*: *name*;
    - *Selector*: *app*.

O código com a implementação completa se encontra no Apêndice G.

- *result-app*-service.*yaml*: Esse arquivo irá conter as configurações de *service* da aplicação *result*, essa configuração irá prover uma abstração de rede para aplicação *result* que será utilizada pela aplicação para habilitar a comunicação via rede entre os demais serviços e a aplicação *result*. Esse arquivo será configurado da seguinte forma:
    - *Type*: LoadBalancer;

- Porta: 80;
- *TargetPort*: 80;
- Etiqueta: *name*;
- Etiqueta: *app*;
- *Selector*: *name*;
- *Selector*: *app*.

O código com a implementação completa se encontra no Apêndice H.

- *voting-app*-service.*yaml*: Esse arquivo irá conter as configurações de *service* da aplicação *voting*, essa configuração irá prover uma abstração de rede para aplicação *voting* que será utilizada pela aplicação para habilitar a comunicação via rede entre os demais serviços e a aplicação *voting*. Esse arquivo será configurado da seguinte forma:
  - *Type*: LoadBalancer;
  - Porta: 80;
  - *TargetPort*: 80;
  - Etiqueta: *name*;
  - Etiqueta: *app*;
  - *Selector*: *name*;
  - *Selector*: *app*.

O código com a implementação completa se encontra no Apêndice I.

3.2.5 Configurações do *Ansible*

Neste trabalho será criado três *playbooks* de configuração computacional utilizando *Ansible*, serão elas:

- configurar_*Kubernetes*.*yaml*: será responsável por conectar no servidor previamente configurado e realizar todas as configurações e instalações necessárias para criar e disponibilizar um *cluster* de *Kubernetes*.

O código com a implementação completa se encontra no Apêndice J.

- *Deploy*_aplicacao.*yaml*: será responsável por conectar no servidor previamente configurado e realizar o *Deploy* da aplicação pela primeira vez, copiando os arquivos de configuração do *Kubernetes*, mencionados anteriormente, para dentro do servidor e em seguida,

através da utilização da ferramenta *Kubernetes*, criar toda a aplicação dentro no *cluster* de *Kubernetes*.

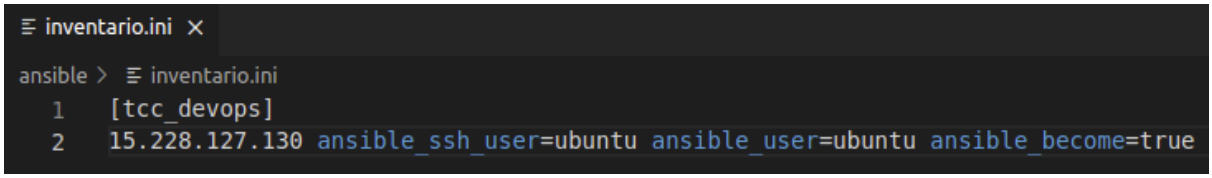
O código com a implementação completa se encontra no Apêndice K.

- *Rollback*_aplicacao.*yaml*: será responsável por conectar no servidor previamente configurado e realizar o *Rollback* da aplicação, o que irá refletir na regressão de versão da aplicação.

O código com a implementação completa se encontra no Apêndice L.

As *playbooks* neste trabalho tem como propósito configurar o servidor, realizar o primeiro *Deploy* da aplicação e o *Rollback* dela, caso haja necessidade, de forma automatizada, porém com a necessidade de ação de um engenheiro *DevOps*, a seguir, na figura 8, há uma imagem representando o arquivo mencionado anteriormente.

Figura 8 – Arquivo inventario.ini

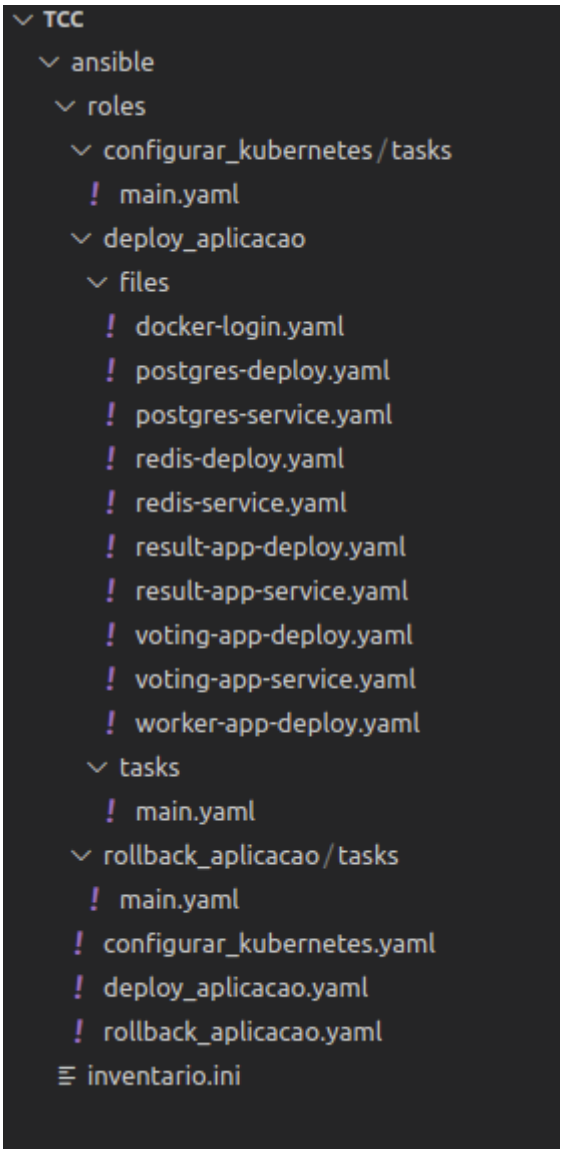


Fonte: Elaborado pelo autor (2021)

Já os demais *Deploys* serão feitos de forma 100% automatizados, utilizando gatilhos que iniciarão o processo de CI, que por sequência irá testar a aplicação e em caso de sucesso, criar uma imagem de container e realizar o *Deploy* da nova imagem de *container* no *cluster Kubernetes*. Para que o engenheiro *DevOps* consiga se comunicar com o servidor para executar os códigos computacionais necessários, foi criado um arquivo denominado inventario.ini que possui as configurações necessárias para que o *Ansible* consiga se conectar no servidor e executar os comandos informados pelo engenheiro *DevOps*.

Todos os códigos computacionais que serão utilizados neste trabalho, foram organizados em uma pasta denominada “tcc”, a seguir há um esboço apresentado na figura 8 de como foi organizado os arquivos mencionados.

Figura 9 – Estrutura de pastas para os códigos computacionais



Fonte: Elaborado pelo autor (2021)

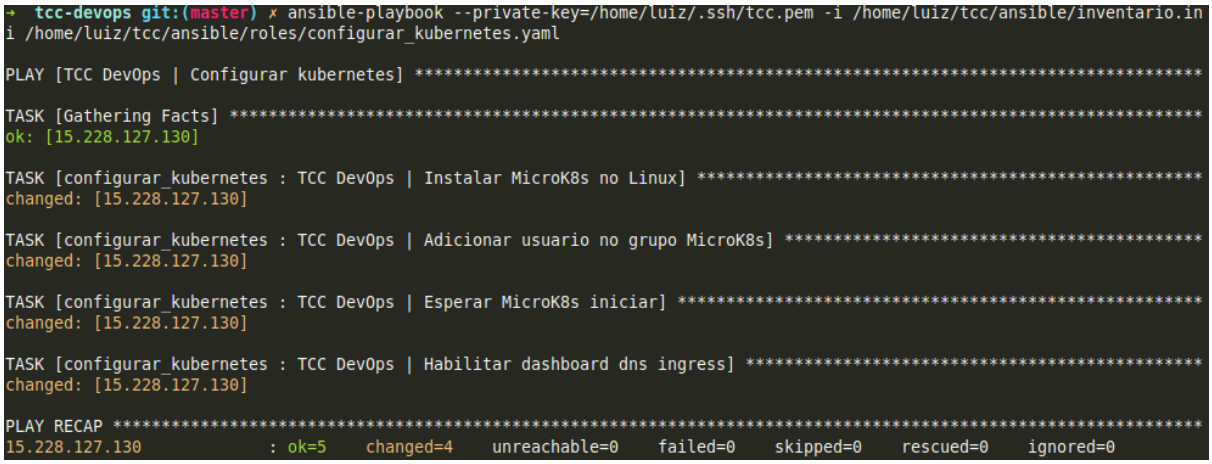
3.3 Demonstração da aplicação

Essa sessão irá demonstrar a aplicação em execução e os preparos para que isso seja possível.

3.3.1 Execução das *playbooks* do *Ansible*

Para configurar o servidor será executado a *playbook* configurar_*Kubernetes*.*yaml* utilizando o *Ansible*, a execução desse código computacional foi registrada a seguir, na figura 10.

Figura 10 – Execução da *playbook* configurar_*Kubernetes*.*yaml*



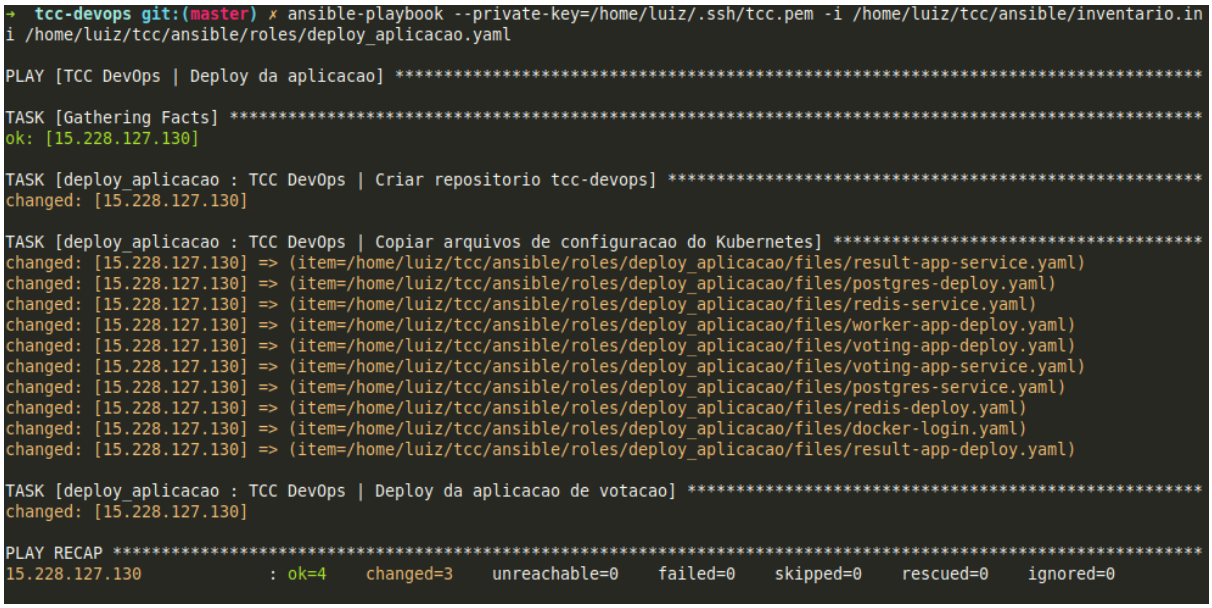
Fonte: Elaborado pelo autor (2021)

Após a execução dessa *playbook* o servidor passou por quatro etapas de configuração:

- TCC *DevOps* | Instalar MicroK8s no Linux: Esse passo é responsável por instalar o MicroK8s, um simplificador para instalar o *cluster* de *Kubernetes*;
- TCC *DevOps* | Adicionar usuario no grupo MicroK8s: Esse passo adiciona o usuário atual do servidor, no caso root, ao grupo de administradores do MicroK8s;
- TCC *DevOps* | Adicionar usuario no grupo MicroK8s: Esse passo aguarda até que o *cluster* de *Kubernetes* esteja disponível para uso;
- TCC *DevOps* | Habilitar dashboard dns ingress: Esse passo habilita a comunicação de rede entre o *cluster* e o servidor, assim como o dashboard para visualização do status do *cluster*.

Após configurar o servidor, será executado a *playbook Deploy*_aplicacao.*yaml* utilizando o *Ansible*, a execução desse código computacional foi registrada a seguir, na figura 11.

Figura 11 – Execução da *playbook Deploy*_aplicacao.*yaml* (v1)



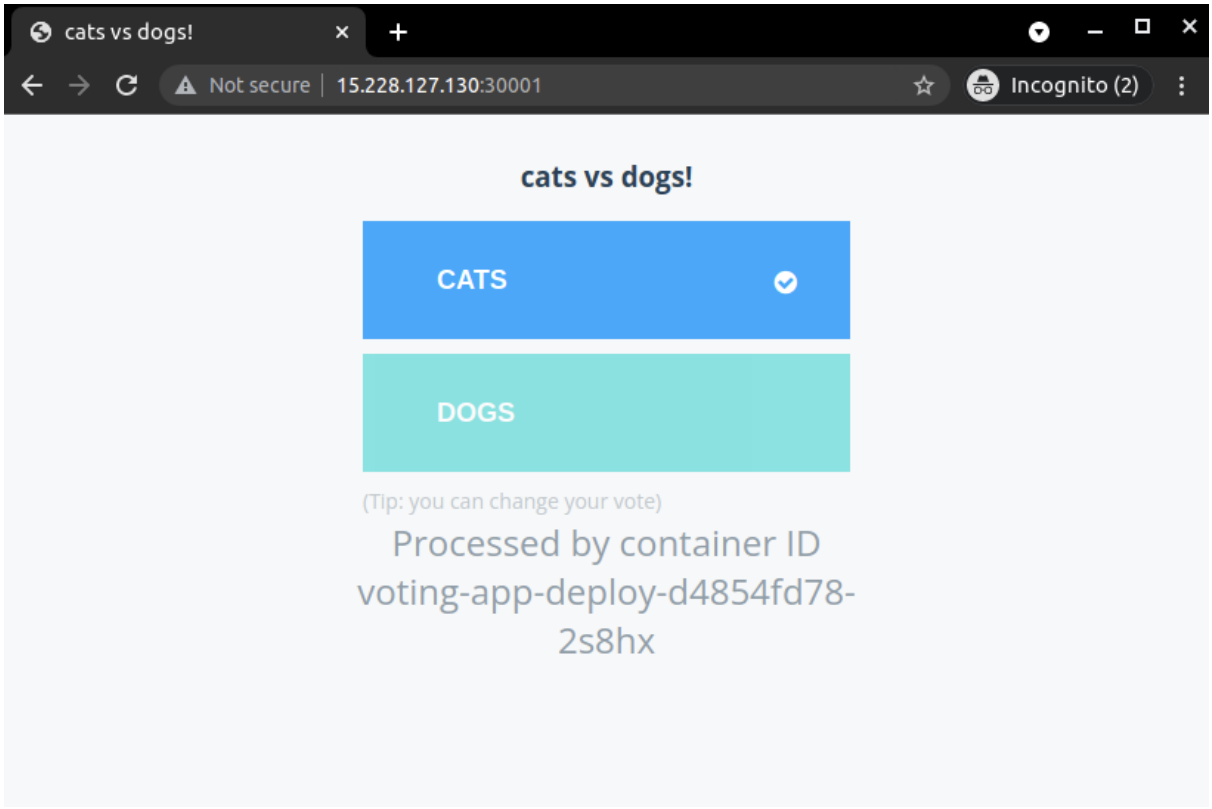
Fonte: Elaborado pelo autor (2021)

Após a execução dessa *playbook* o servidor passou por três etapas de configuração:

- TCC *DevOps* | Criar repositorio tcc-*DevOps*: Esse passo é responsável por criar o repositório no servidor para armazenar os arquivos de configuração do *Kubernetes*;
- TCC *DevOps* | Copiar arquivos de configuracao do *Kubernetes*: Esse passo é responsável por copiar os arquivos de configuração do *Kubernetes* da máquina local do engenheiro *DevOps* para o servidor;
- TCC *DevOps* | *Deploy* da aplicacao de votação: Esse passo é responsável por aplicar os arquivos configurações do *Kubernetes* dentro do *cluster*, criando e disponibilizando para uso toda a aplicação.

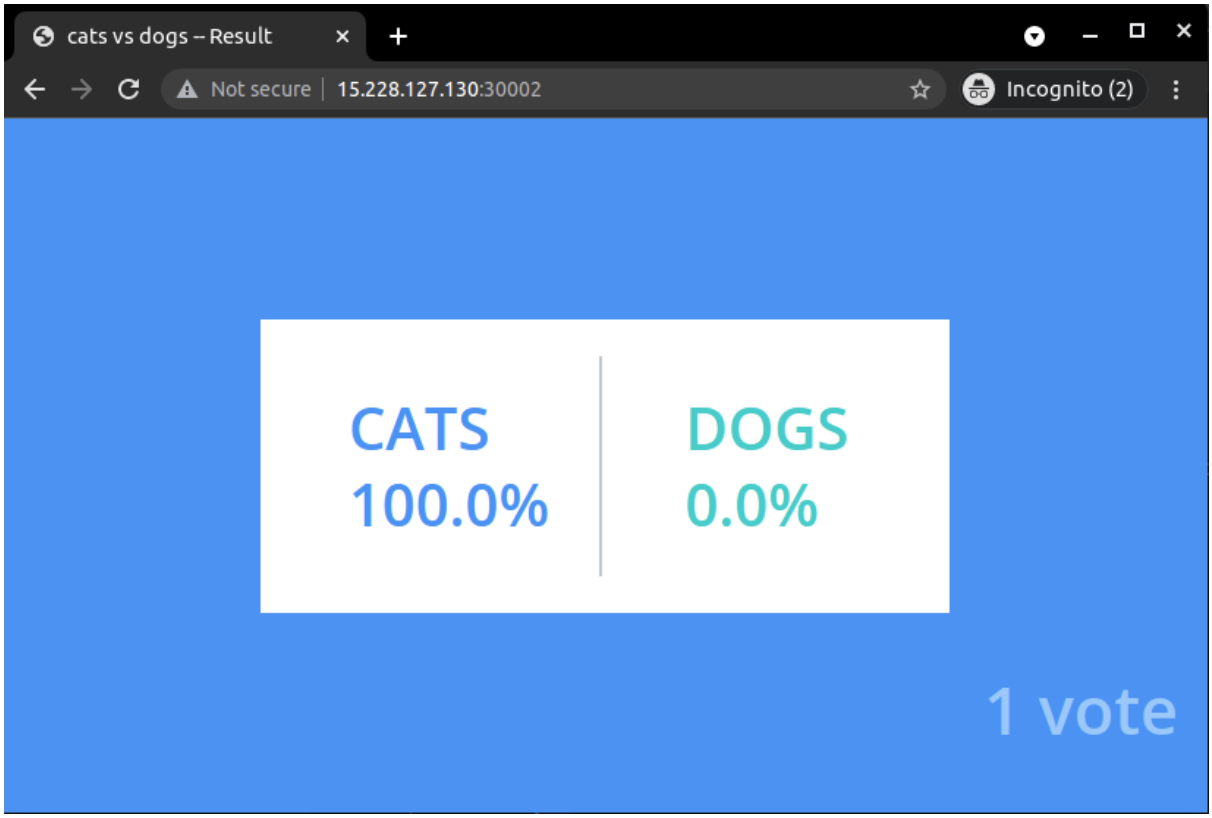
Ao final da execução das *playbooks* acima, a aplicação de *vote* e *result* ficam disponíveis para serem acessadas no endereço do servidor através das respectivas portas: 30001 (*vote*) e 30002 (*result*), como registrado a seguir, nas figuras 12 e 13.

Figura 12 – Aplicação de *vote* (v1)



Fonte: Elaborado pelo autor (2021)

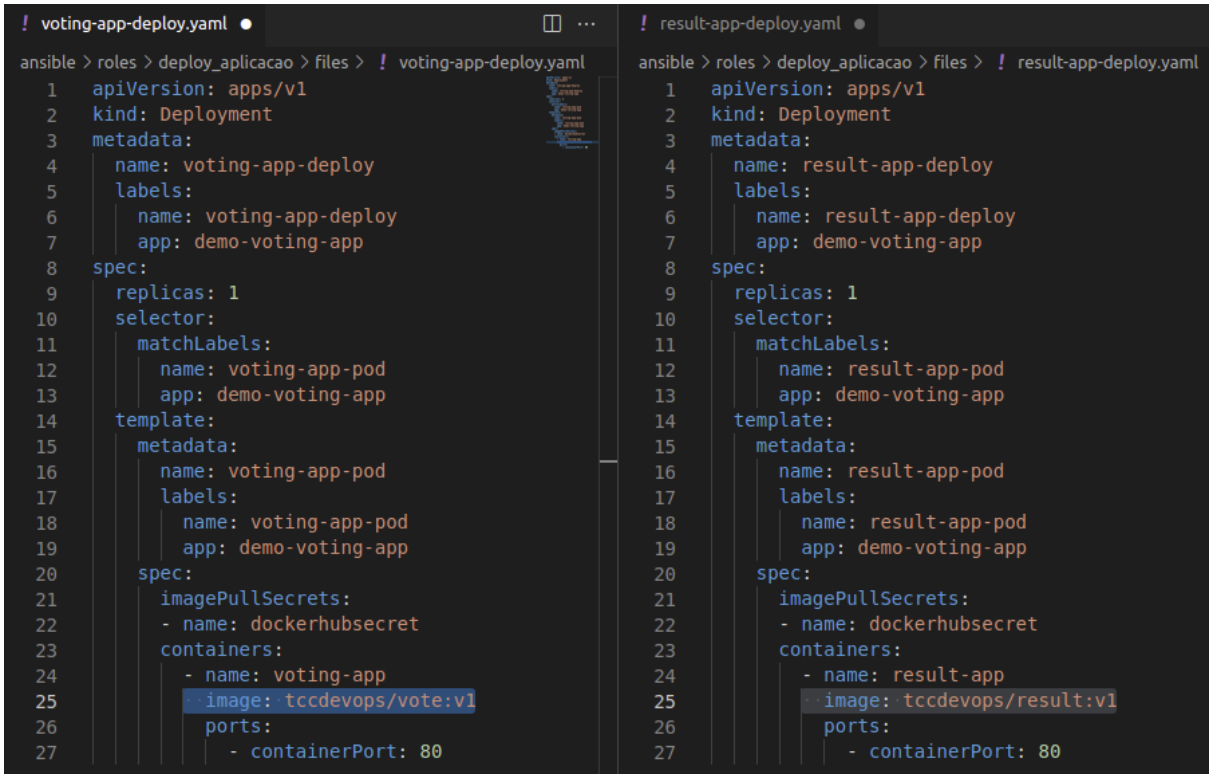
Figura 13 – Aplicação de *result* (v1)



Fonte: Elaborado pelo autor (2021)

Nessa primeira versão (v1) as aplicações de *vote* e *result* estão configuradas para utilizarem a imagem do *Docker Hub* com seu respectivo número de versão, como ilustrado a seguir, na figura 14.

Figura 14 – *voting-app-Deploy.yaml* e *result-app-Deploy.yaml* (v1)



voting-app-deploy.yaml

ansible > roles > deploy_aplicacao > files > voting-app-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-app-deploy
  labels:
    name: voting-app-deploy
    app: demo-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: voting-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: voting-app-pod
      labels:
        name: voting-app-pod
        app: demo-voting-app
    spec:
      imagePullSecrets:
      - name: dockerhubsecret
      containers:
        - name: voting-app
          image: tccdevops/vote:v1
          ports:
            - containerPort: 80
```

result-app-deploy.yaml

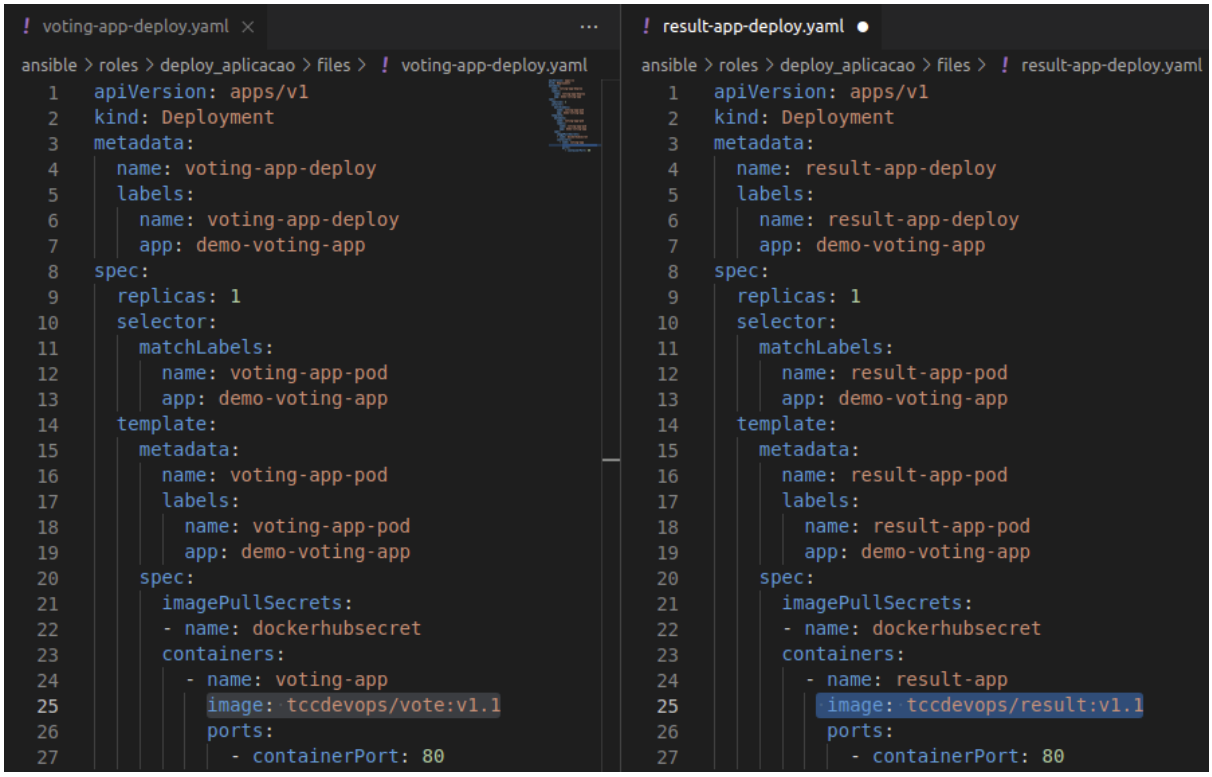
ansible > roles > deploy_aplicacao > files > result-app-deploy.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: result-app-deploy
  labels:
    name: result-app-deploy
    app: demo-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: result-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: result-app-pod
      labels:
        name: result-app-pod
        app: demo-voting-app
    spec:
      imagePullSecrets:
      - name: dockerhubsecret
      containers:
        - name: result-app
          image: tccdevops/result:v1
          ports:
            - containerPort: 80
```

Fonte: Elaborado pelo autor (2021)

Com intuito de demonstrar o processo de *Deploy* e *Rollback* foi alterado o número da versão de ambas as aplicações (v1.1) e em seguida executado a *playbook Deploy*_aplicacao.*yaml* novamente, para realizar o *Deploy* de uma nova versão utilizando o método *rolling update* previamente discutido neste trabalho. Os passos de alteração de versão das aplicações de *vote* e *result* foram registradas a seguir, na imagem 15 e a execução da *playbook* foi registrada na figura 16.

Figura 15 – *voting-app-Deploy.yaml* e *result-app-Deploy.yaml* (v1.1)



voting-app-deploy.yaml

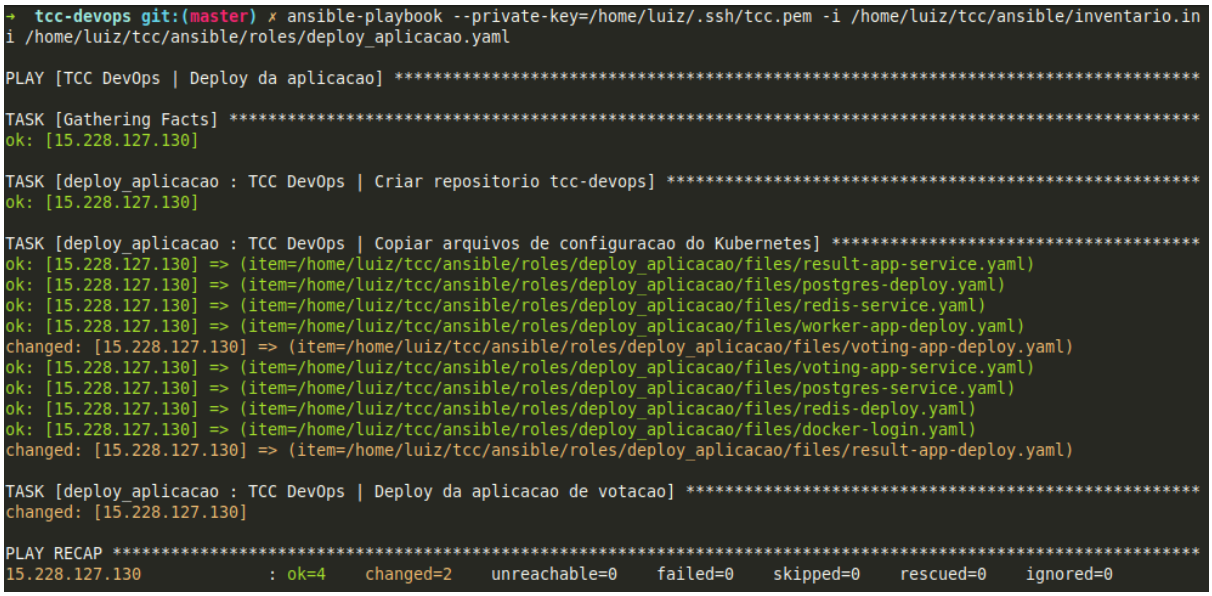
```
ansible > roles > deploy_aplicacao > files > ! voting-app-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-app-deploy
  labels:
    name: voting-app-deploy
    app: demo-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: voting-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: voting-app-pod
      labels:
        name: voting-app-pod
        app: demo-voting-app
    spec:
      imagePullSecrets:
      - name: dockerhubsecret
      containers:
        - name: voting-app
          image: tccdevops/vote:v1.1
          ports:
            - containerPort: 80
```

result-app-deploy.yaml

```
ansible > roles > deploy_aplicacao > files > ! result-app-deploy.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: result-app-deploy
  labels:
    name: result-app-deploy
    app: demo-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: result-app-pod
      app: demo-voting-app
  template:
    metadata:
      name: result-app-pod
      labels:
        name: result-app-pod
        app: demo-voting-app
    spec:
      imagePullSecrets:
      - name: dockerhubsecret
      containers:
        - name: result-app
          image: tccdevops/result:v1.1
          ports:
            - containerPort: 80
```

Fonte: Elaborado pelo autor (2021)

Figura 16 – Execução da *playbook Deploy*_aplicacao.*yaml* (v1.1)



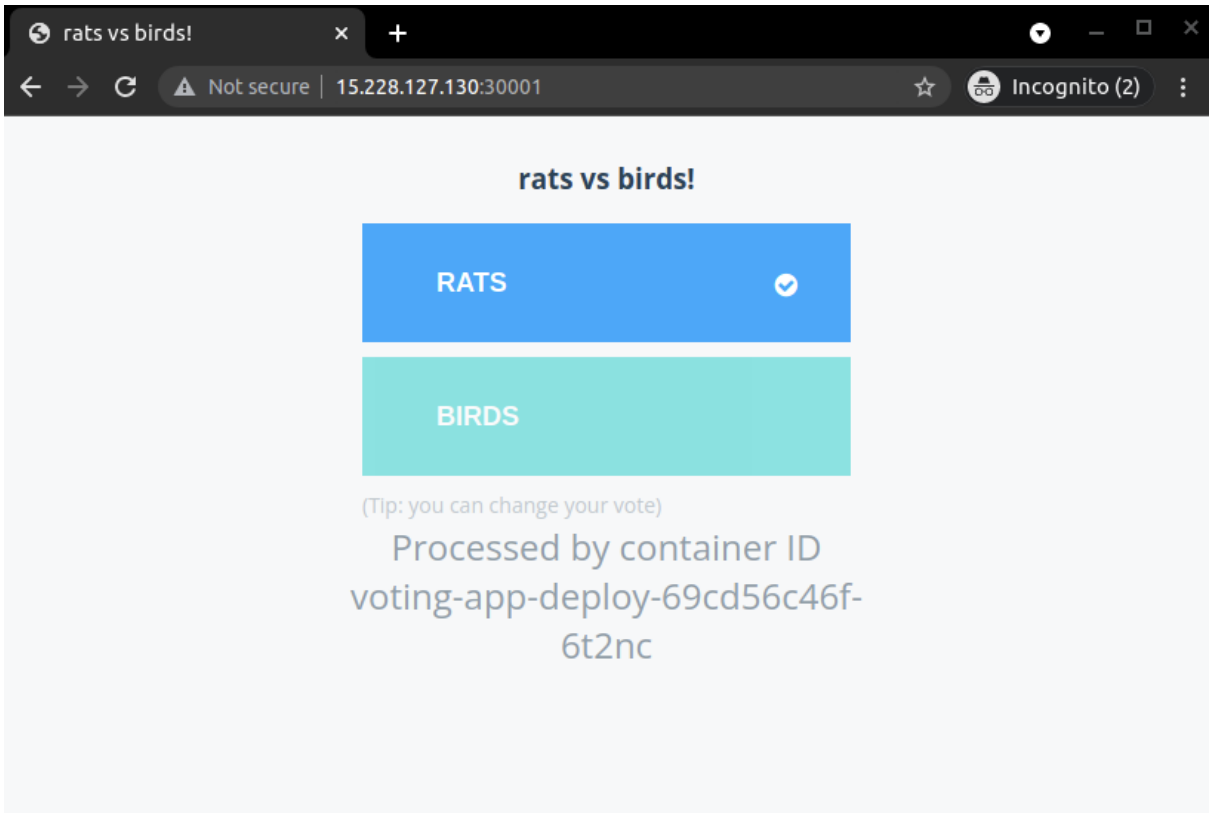
```
→  tcc-devops git:(master) ✗ ansible-playbook --private-key=/home/luiz/.ssh/tcc.pem -i /home/luiz/tcc/ansible/inventario.in
i /home/luiz/tcc/ansible/roles/deploy_aplicacao.yaml

PLAY [TCC DevOps | Deploy da aplicacao] ****************************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [15.228.127.130]

TASK [deploy_aplicacao : TCC DevOps | Criar repositorio tcc-devops] ************************************************
ok: [15.228.127.130]

TASK [deploy_aplicacao : TCC DevOps | Copiar arquivos de configuracao do Kubernetes] ******************************
ok: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/result-app-service.yaml)
ok: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/postgres-deploy.yaml)
ok: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/redis-service.yaml)
ok: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/worker-app-deploy.yaml)
changed: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/voting-app-deploy.yaml)
ok: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/voting-app-service.yaml)
ok: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/postgres-service.yaml)
ok: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/redis-deploy.yaml)
ok: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/docker-login.yaml)
changed: [15.228.127.130] => (item=/home/luiz/tcc/ansible/roles/deploy_aplicacao/files/result-app-deploy.yaml)

TASK [deploy_aplicacao : TCC DevOps | Deploy da aplicacao de votacao] **********************************************
changed: [15.228.127.130]

PLAY RECAP *********************************************************************************************************
15.228.127.130             : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Fonte: Elaborado pelo autor (2021)

Como pode ser observado, somente os arquivos *voting-app-Deploy.yaml* e *result-app-Deploy.yaml* possuem o status *changed* os demais possuem o status *ok*,

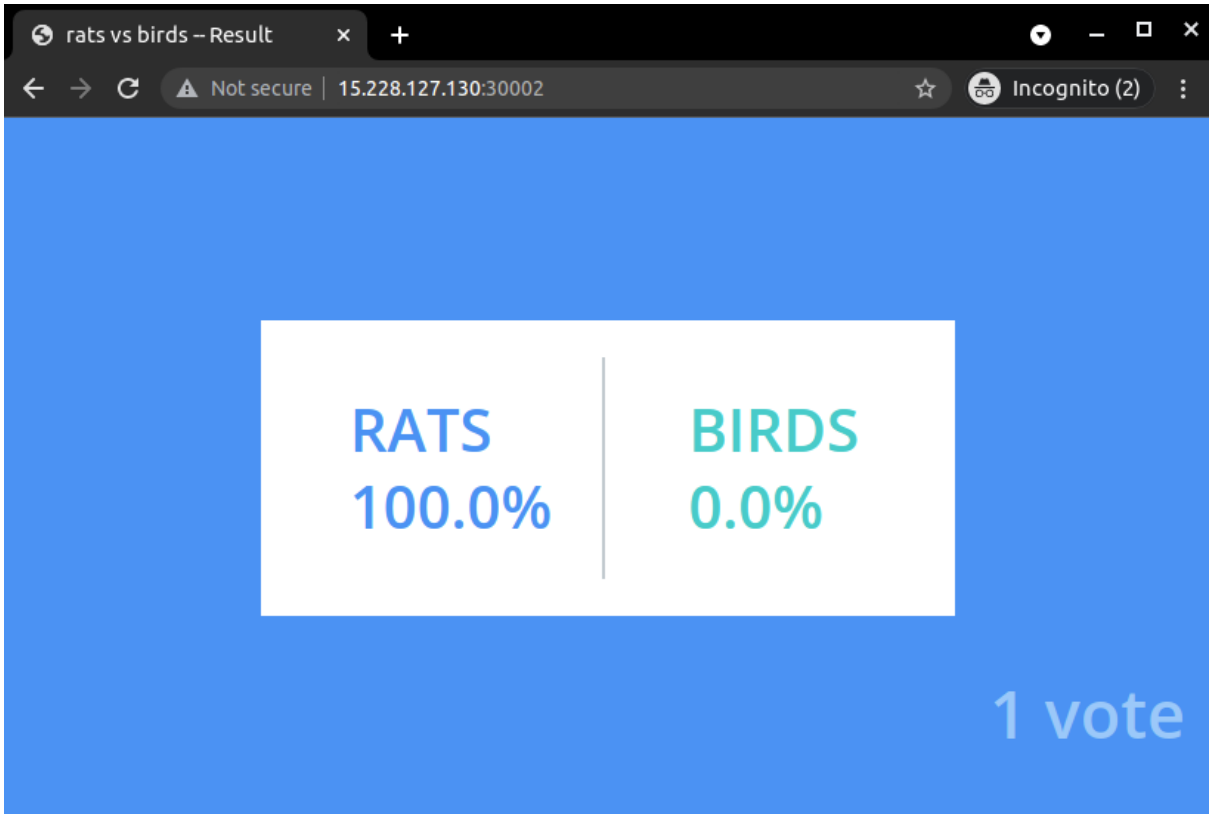
pois somente esses dois arquivos tiveram suas respectivas versões alteradas. Após o processo de *Deploy*, a versão v1.1 encontra-se em execução. Ao realizar acesso a aplicação, nota-se que o nome dos animais disponíveis para votação foi alterado, simulando assim uma alteração de código genérica. Pode ser notado que anteriormente a aplicação disponibiliza as opções *dogs* e *cats*, agora a mesma aplicação está disponibilizando as opções *rats* e *birds*, essas mudanças foram registradas nas figuras 17 e 18.

Figura 17 – Aplicação de *vote* (v1.1)



Fonte: Elaborado pelo autor (2021)

Figura 18 – Aplicação de *result* (v1.1)



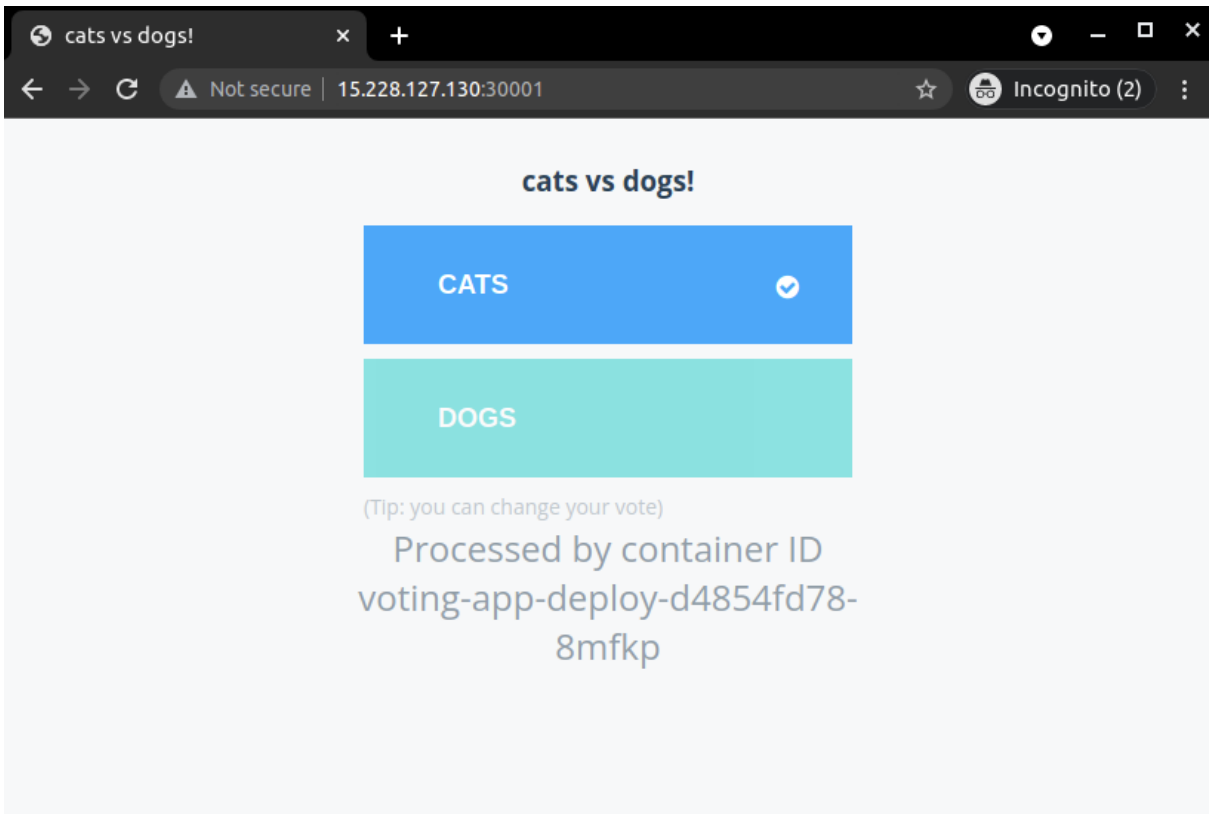
Fonte: Elaborado pelo autor (2021)

O processo de *Rollback*, como exposto previamente neste trabalho, realiza a reversão da última versão da aplicação para uma versão anterior, no caso dessa demonstração, através do processo de *Deploy*, foi instalada a versão v1 da aplicação e em seguida, foi realizado a mudança do código computacional da aplicação onde foi alterado o nome dos animais disponíveis para votação em sem seguida foi realizado um novo processo de *Deploy*, onde a versão v1.1 da aplicação foi instalada, agora para simular o processo de *Deploy*, será executado a *playbook Rollback*_aplicacao.*yaml*, que irá desfazer o último *Deploy* (v1.1) e retornar a versão anterior (v1), esse processo foi registrado nas figuras 19, 20 e 21

Figura 19 – Execução da *playbook Rollback*_aplicacao.*yaml*

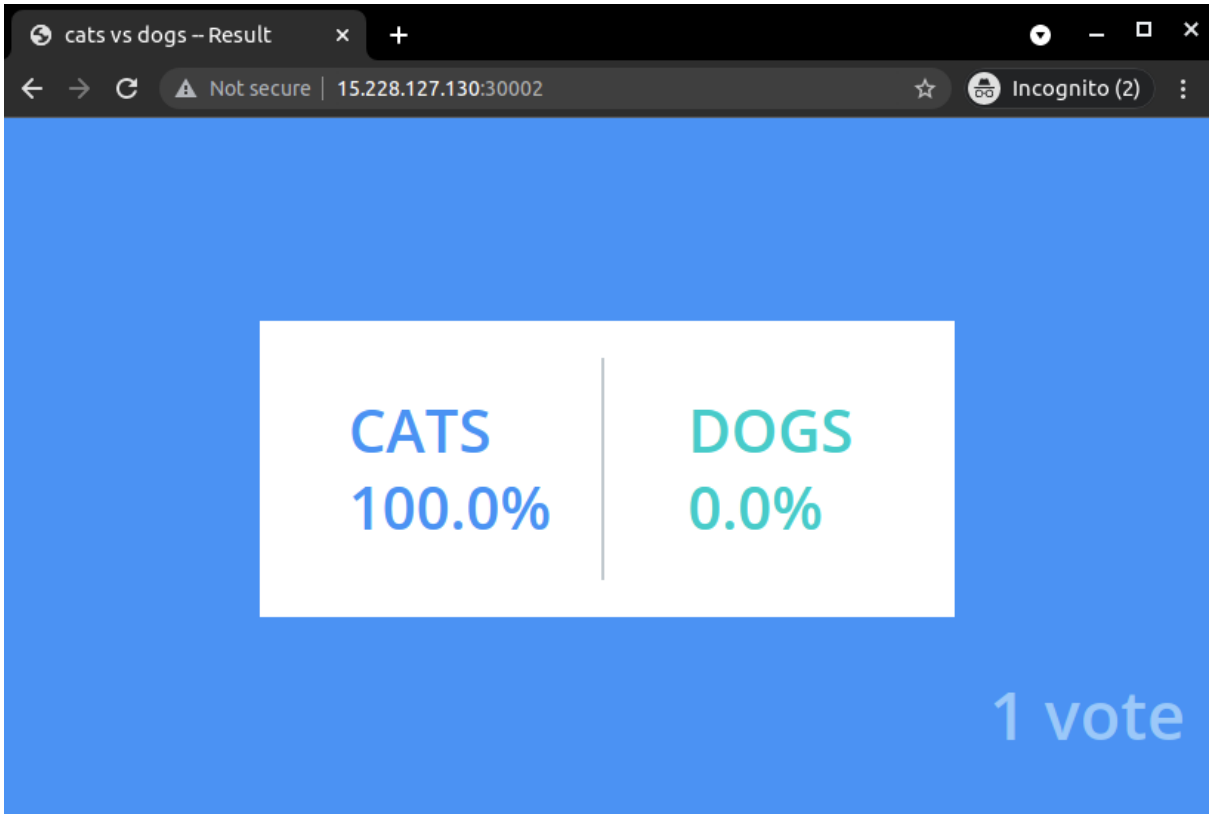
Fonte: Elaborado pelo autor (2021)

Figura 20 – Aplicação de *vote* (v1)



Fonte: Elaborado pelo autor (2021)

Figura 21 – Aplicação de *result* (v1)



Fonte: Elaborado pelo autor (2021)

3.3.2 Execução da *pipeline* de CI/CD no *GitHub Actions*

Com intuito de automatizar os processos mais frequentes em um ambiente de desenvolvimento de código computacional, foi criado uma *pipeline* denominada main.*yaml* que é responsável por executar uma série de passos sempre que uma atualização for feita no código computacional da aplicação, aplicando o conceito, previamente abordado neste trabalho, de CI e CD.

O código com a implementação completa se encontra no Apêndice M.

Os passos executados nessa *pipeline* descrevem um fluxo comum de CI e CD sempre que ocorrer alguma mudança no código computacional, os passos são:

- *Build app vote*: Esse passo realiza um clone do repositório de código computacional, com o código mais atualizado da aplicação *vote*, cria uma imagem de *container Docker* e em seguida envia essa imagem para o *Docker Hub*.

- *Build app result*: Esse passo realiza um clone do repositório de código computacional, com o código mais atualizado da aplicação *result*, cria uma imagem de container *Docker* e em seguida envia essa imagem para o *Docker Hub*.
- *Deploy app vote*: Esse passo faz uma conexão SSH com o servidor e realiza a atualização da imagem de container que a aplicação *vote* deve atualizar para ser executada, esse processo inicia um *rolling update* e atualizar a aplicação *vote* com a versão atual.
- *Deploy app vote*: Esse passo faz uma conexão SSH com o servidor e realiza a atualização da imagem de *container* que a aplicação *result* deve atualizar para ser executada, esse processo inicia um *rolling update* e atualizar a aplicação *result* com a versão atual.
- Esperar 15 segundos: Esse passo apenas espera 15 segundos para que o processo de *rolling update* seja concluído e as aplicações sejam iniciadas com a nova versão.
- *Test app vote*: Esse passo realiza a execução dos testes da aplicação *vote* utilizando a ferramenta *Cypress*, esse teste acessa a interface *web* da aplicação de *vote* e verifica se os animais disponíveis para votação são os configurados para verificação no arquivo test-*vote*.js, caso os nomes não estejam corretos, esse teste falha e posteriormente o passo "*Rollback app vote*" é executado.

  O código com a implementação completa se encontra no Apêndice N.
- *Test app result*: Esse passo realiza a execução dos testes da aplicação *result* utilizando a ferramenta *Cypress*, esse teste acessa a interface *web* da aplicação de *result* e verifica se os animais apresentados como resultado da votação são os configurados para verificação no arquivo *result*-*vote*.js, caso os nomes não estejam corretos, esse teste falha e posteriormente o passo *"Rollback app result*" é executado.

  O código com a implementação completa se encontra no Apêndice O.
- *Rollback app vote*: Esse passo é executado somente quando o passo "*test app vote*" falhar, caso essa condição seja atendida a *pipeline* conecta no servidor e executa um *Rollback* na versão da aplicação *vote*.

- *Rollback app result*: Esse passo é executado somente quando o passo “*test app result*” falhar, caso essa condição seja atendida a *pipeline* conecta no servidor e executa um *Rollback* na versão da aplicação *result*.
- Verificar falhas: Esse passo verifica se o passo “*test app vote*” ou o passo “*test app result”* falharam, caso essa condição seja atendida, a *pipeline* é marcada com o status *Failure* caso a condição não seja atendida, isto implica que a nova versão da aplicação está passando em todos os testes previamente executados, marcando a *pipeline* com o status *Success*.

4 CONSIDERAÇÕES FINAIS

O presente trabalho propôs-se a demonstrar como tornar a cultura *DevOps* acessível a pequenas empresas e *startups*, utilizando as principais e mais modernas ferramentas de código aberto da atualidade.

Durante o desenvolvimento deste trabalho consegue-se perceber claramente que manter uma aplicação computacional funcionando de forma adequada é um desafio para os engenheiros responsáveis, realizar essa tarefa sem a ajuda da cultura *DevOps* e as ferramentas de código aberto, pode se tornar um grande problema no dia a dia de pequenas empresa e *startups*.

Das ferramentas de código aberto que foram utilizadas, o *Kubernetes* provou-se ser de grande importância quando o assunto é lidar com a orquestração de aplicações em *containers*, a simplicidade dos *templates YAML* para abstrair e configurar a comunicação da aplicação com a rede de computadores acaba sendo um aliado de peso para o engenheiro *DevOps*, evitando grande partes dos problemas normalmente encontrados quando se precisa configurar essa comunicação de forma manual.

Como demonstrado no trabalho, os processos de *Deploy* e *Rollback*, comumente utilizados no dia a dia em um ambiente de desenvolvimento de códigos computacionais, foram executados manualmente pelo engenheiro *DevOps* em um primeiro momento do presente trabalho, após a automatização desses dois processos com o auxílio da ferramenta *GitHub Actions*, nota se que um comando que é executado com frequência, ao ser automatizado, reduz o tempo necessário de trabalho manual do engenheiro *DevOps* além de evitar possíveis falhas humanas ao se executar comandos computacionais manualmente. Com essa implementação em específico fica claro que a adesão à cultura *DevOps* tem muito a acrescentar a empresas de desenvolvimento de código computacional.

Nas dificuldades enfrentadas no presente trabalho cabe citar a dificuldade da configuração do *cluster Kubernetes*, sem utilizar um gerenciador de nuvem, além de ser uma ferramenta complexa, o *Kubernetes* não possui uma documentação de fácil entendimento e durante a execução do trabalho ocorreram alguns problemas para iniciar os gerenciadores do *cluster.* Porém foram sanados com a habilidade do engenheiro *DevOps*.

Para que o problema com a execução dos gerenciadores do *cluster*, foi utilizado uma ferramenta de código aberta denominada MicroK8s, que abstrai a configuração complexa dos gerenciadores do *cluster* e facilita a execução deles através de um CLI amigável e bem documentado no presente trabalho.

Esse trabalho possibilitou um grande aumento de conhecimento, tanto a respeito das ferramentas de código aberto utilizadas quanto da cultura *DevOps* em si, mesmo trabalhando diariamente como *DevOps*, o engenheiro *DevOps* deve aprender e se especializar mais dentro desse universo, rico e inovador.

O projeto apresenta uma demonstração de como a otimização de uma aplicação pode refletir em menos erros humanos e em economia de tempo. Os objetivos propostos neste trabalho foram todos cumpridos.

4.1 Trabalhos futuros

Como trabalhos futuros sugere-se:

- Realizar a criação do *cluster* de *Kubernetes* utilizando um gerenciador de nuvem;
- Modificar a maneira que a *pipeline* de testes está sendo executada, adicionando uma validação em ambiente local antes de realizar o *Deploy*;
- Adicionar *Test*es de cobertura no código computacional;
- Separar ambientes para uma maior resiliência da aplicação, dividindo o ambiente em três: desenvolvimento, homologação e produção;
- Alterar a arquitetura da aplicação para possuir mais réplicas dela, para que ações mais resilientes, como *Deploy blue/green*, possam ser exploradas;
- Automatizar a criação dos recursos de infraestrutura, como o servidor, utilizando ferramentas de infraestrutura como código, como *Terraform*;
- Adicionar rotina de *backup* para armazenar os dados da votação;
- Gerar um certificado SSL para que a aplicação possa ser acessada com maior segurança pelo protocolo HTTPS;
- Criar um ponteiro DNS para facilitar o acesso a aplicação;
- Criar um sistema de monitoramento para controlar o status da aplicação.

REFERÊNCIAS

ALOIA, João Victor Peria; RIBEIRO, José Eduardo. **Análise Comparativa entre Processos de Entrega de *Software* do Tradicional à Entrega Contínua**. UNIARA, Universidade de Araraquara, 2017. Disponível em: < https://semanaacademica.org.br/system/files/artigos/analise_comparativa_entre_processos_de_entrega_de_*software*_do_tradicional_a_entrega_continua_0.pdf> Acesso em 07 de maio de 2021.

ALURA, ***Ansible*: Sua infraestrutura como código**, 2021**.** Disponível em: <https://www.alura.com.br/conteudo/infraestrutura-como-codigo-com-*Ansible*> Acesso em 04 de maio de 2021.

AMAZON, **O que significa distribuição contínua?** Disponível em: <https://*AWS*.amazon.com/pt/*DevOps*/continuous-delivery/> Acesso em: 23 de abril de 2021.

BICCA, Cristhian. **Conheça a nova ferramenta queridinha do pessoal de *DevOps*: *Ansible***. iMasters, 2016. Disponível em: <https://imasters.com.br/devsecops/conheca-nova-ferramenta-queridinha-do-pessoal-de-*DevOps-Ansible*> Acesso em: 04 de maio de 2021.

BURNS, Brendan. **O que é um contêiner?** Microsoft *Azure*, 2019. Disponível em:<https://*Azure*.microsoft.com/pt-br/overview/what-is-a-container/#overview> Acesso em: 2 de abril de 2021.

DELGADO, Fabio. **Site Reliability Engineering**. Medium Blog, 2019. Disponível em: < https://medium.com/@fabio.delgado2/sre-bf4ca0cdb82c > Acesso em: 20 de março de 2021.

DOCKER. **What is a Container? A standardized unit of *software*.** *Docker*, 2019. Disponível em: <https://www.*Docker*.com/resources/what-container> Acesso em: 2 de abril de 2021.

ETMAJER, Martin. **Continuous Delivery 101: Automated *Deploy*ments**. Dynatrace, 2014. Disponível em: <https://www.dynatrace.com/news/blog/continuous-delivery-101-automated-*Deploy*ments/> Acesso em: 07 de maio de 2021

EXIN. **Open Source e o *DevOps*! Conheça os 5 principais benefícios**. Security Management ISO/IEC 27001 Foundation – ISFS. Disponível em: <https://www.exin.com/br-pt/open-source-e-o-*DevOps*-conheca-os-5-principais-beneficios >Acesso em: 23 de março de 2021.

FARADAY, Gabriel. **O que é CI/CD? Onde eu uso isso?** São Paulo, 2019. Disponível em: <https://gabriel-faraday.medium.com/o-que-%C3%A9-ci-cd-onde-eu-uso-isso-57e9b8ad8c73> Acesso em: 23 de abril de 2021.

FRUHLINGER, Josh. **O que é um engenheiro *DevOps* e como você pode se tornar um?** Cio, Infoword, 2019. Disponível em:< https://cio.com.br/carreira/o-que-e-um-engenheiro-*DevOps*-e-como-voce-pode-se-tornar-um/> Acesso em: 23 de março de 2021.

GILL, Navdeep Singh. **10 Best Infrastructure as Code (*IaC*) Tools to Boost Your Productivity**. NexaStack, 2020. Disponível em: < https://www.nexastack.com/en/blog/best-*IaC*-tools> Acesso em: 03 de maio de 2021.

GITHUB. ***GitHub* Docs**. Disponível em:< https://docs.*GitHub*.com/pt/*GitHub*> Acesso em: 2 de abril de 2021.

GOOGLE CLOUD. **Contêineres no Google**. Google, 2019. Disponível em: <https://cloud.google.com/*containers*?hl=pt-br> Acesso em: 2 de abril de 2021.

HALL, Tom. **How a *DevOps* engineer facilitates *DevOps*.** Atlassian Corporation, 2020. Disponível em: <https://www.atlassian.com/*DevOps*/what-is-*DevOps*/*DevOps*-engineer> Acesso em: 20 de março de 2021.

HYDER, Riyaz. **What is *DevOps*? How is it different from Agile?** Botmetric, 2017. Disponível em: < https://www.botmetric.com/blog/what-is-*DevOps*/ > Acesso em: 19 de março de 2021.

*HOST*GATOR. **Saiba o que é *Deploy* e as formas de realizá-lo**. 2020. Disponível em: <https://www.*host*gator.com.br/blog/o-que-e-*Deploy*-e-como-realiza-lo/> Acesso em: 07 de maio de 2021

IBM DOCS. **Manual *Deploy*ment**. Disponível em: <https://www.ibm.com/docs/en/sc-and-ds/8.1.0?topic=configuration-manual-*Deploy*ment> Acesso em: 07 de maio de 2021

JACHJA, Tiffany. **Intro to *Deploy*ment Strategies: Blue-Green, Canary, and More**. Harness, 2021. Disponível em: <https://harness.io/blog/continuous-verification/blue-green-canary-*Deploy*ment-strategies/> Acesso em: 08 de maio de 2021

KELLY, Katherine. **Using *Ansible* in *DevOps*: A Beginners Guide**. Liquid*Web*, 2020. Disponível em: <https://www.liquid*web*.com/kb/using-*Ansible*-in-*DevOps*-a-beginners-guide/> Acesso em 04 de março de 2021.

KUBERNETES. ***Build*ing large *cluster*s**. Disponível em: <https://v1-17.docs.*Kubernetes*.io/docs/setup/best-practices/*cluster*-large/> Acesso em 04 de outubro de 2021.

MEZAK, Steve. **The Origins of *DevOps*: What's in a *Name*?** *DevOps* Practice, 2018. Disponível em: <https://*DevOps*.com/the-origins-of-*DevOps*-whats-in-a-*name*/ > Acesso em: 20 de março de 2021.

MICHAELIS. **Moderno Dicionário da Língua Portuguesa**. São Paulo: Melhoramentos.Disponível em:< https://michaelis.uol.com.br/busca?id=YkYxK Michaelis> Acesso em: 6 de abril de 2021.

MICROSOFT. **O que é um projeto público?** Documentação Microsoft, 2019. Disponível em:< https://docs.microsoft.com/pt-br/*Azure*/*DevOps*/organizations/public/about-public-projects?view=*Azure-DevOps*> Acesso em: 6 de abril de 2021.

MONUS, Anna. **The 10 best *DevOps* tools for 2021 and beyond**, Raygun Blog, 2018. Disponível em:< https://raygun.com/blog/best-*DevOps*-tools/> Acesso em: 06 de abril de 2021.

MURPHY, N. Richard. **Site Reliability Engineering: How Google Runs Production Systems**. New York, USA: O'Reilly Media, 2016.

NALLAMALA, Narendar. **The Top 7 Infrastructure-As-Code Tools For Automation**. *DevOps* Zone, 2019. Disponível em: <https://dzone.com/articles/the-top-7-infrastructure-as-code-tools-for-automat> Acesso em: 03 de maio de 2021.

OPEN SOURCE INITIATIVE. **The Open Source Definition**. Open Source Org, 2006. Disponível em: < https://*web*.archive.org/*web*/20070611152544/https://opensource.org/docs/osd> Acesso em: 20 de março de 2021.

_____. **What is *Ansible*?** Open Source Org, 2021. Disponível em: < https://opensource.com/resources/what-*Ansible*> Acesso em: 04 de maio de 2021.

QUIN, Yinghua. **The Role of *Containers* in Business Transformation**. *DevOps*.com, 2019 Disponível em: <https://*DevOps*.com/the-role-of-*containers*-in-business-transformation/> Acesso em: 2 de abril de 2021.

REDHAT. **Integração e entrega contínuas: pipeline CI/CD**. Disponível em: <https://www.redhat.com/pt-br/topics/*DevOps*/what-is-ci-cd> Acesso em: 23 de abril de 2021.

REDGATE DOCS. ***Rollback***. 2021. Disponível em: <https://documentation.red-gate.com/db*DevOps*/concepts/*Rollback*> Acesso em: 08 de maio de 2021

RESILLE, Willian. **Afinal, o que é *DevOps*?** 2017. Disponível em: < http://agile.pub/assuntos-diversos/afinal-o-que-e-*DevOps*/> Acesso em: 20 de março de 2021

ROBETE, Diana. **How Good Is Your *Rollback* Strategy? - 5 Ingredients For A Successful *Rollback***. 2016. Disponível em: <https://www.linkedin.com/pulse/how-good-your-*Rollback*-strategy-5-ingredients-diana-robete> Acesso em: 08 de maio de 2021

ROVEDA, Ugo. **O que é *Deploy*, para que serve, vantagens e como fazer *Deploy***. Kenzie Academy, 2021. Disponível em: <https://kenzie.com.br/blog/o-que-e-*Deploy*/> Acesso em: 08 de maio de 2021

SHAHIN, M., ALI BABAR, M., & ZHU, L. (2017**). Continuous Integration, Delivery and *Deploy*ment: A Systematic Review on *App*roaches, Tools, Challenges and Practices**. IEEE Access, 5(Ci), 3909–3943. https://doi.org/10.1109/ACCESS.2017.2685629.

SKOWRONSKI, Jason. **Intro to *Deploy*ment strategies: blue-green, canary, and more**. DevDiscuss, 2018. Disponível em: <https://dev.to/mostlyjason/intro-to-*Deploy*ment-strategies-blue-green-canary-and-more-3a3> Acesso em: 08 de maio de 2021.

SRIVATSAV, Bhargav. ***Rollback* and it's importance in a *DevOps* world**. 2017. Disponível em: <https://medium.com/@bharghavyerravalli/*Rollback*-and-its-importance-in-a-*DevOps*-world-21d486738760> Acesso em: 08 de maio de 2021

STARAGILE, **What Is *Ansible*? Uses, Benefits, Architecture**, 2020. Disponível em: <https://staragile.com/blog/what-is-*Ansible*-in-*DevOps*> Acesso em: 04 de maio de 2021.

TOZZI, Christopher. **What Do *Containers* Have to Do with *DevOps*, Anyway?** Container Journal, 2017. Disponível em: <https://containerjournal.com/uncategorized/*containers*-*DevOps*-anyway/> Acesso em: 2 de abril de 2021.

VOLMAR, Philip. **A Short History Lesson in *DevOps* — And Where It's Going.** *DevOps* Zone, 2017. Disponível em: <https://dzone.com/articles/a-short-history-lesson-in-*DevOps*-and-where-its-goi-1> Acesso em: 19 de março de 2021.

APÊNDICES

Apêndice A

Implementação do arquivo de *Deploy* do *Postgres*

Postgres-Deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: Postgres-Deploy
  labels:
    name: Postgres-Deploy
    app: demo-voting-app
spec:
  replicas: 1
  Selector:
    matchLabels:
      name: Postgres-pod
      app: demo-voting-app

  template:
    metadata:
      name: Postgres-pod
      labels:
        name: Postgres-pod
        app: demo-voting-app
    spec:
      containers:
        - name: Postgres
          image: Postgres
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_USER
              value: "Postgres"
            - name: POSTGRES_PASSWORD
              value: "Postgres"
```

Apêndice B

Implementação do arquivo de *Deploy* do *Redis*

Redis-Deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: Redis-Deploy
  labels:
    name: Redis-Deploy
    app: demo-voting-app
spec:
  replicas: 1
  Selector:
    matchLabels:
      name: Redis-pod
      app: demo-voting-app

  template:
    metadata:
      name: Redis-pod
      labels:
        name: Redis-pod
        app: demo-voting-app
    spec:
      containers:
        - name: Redis
          image: Redis
          ports:
            - containerPort: 6379
```

Apêndice C

Implementação do arquivo de *Deploy* da aplicação *result*

result-app-Deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: result-app-Deploy
  labels:
    name: result-app-Deploy
    app: demo-voting-app
spec:
  replicas: 1
  Selector:
    matchLabels:
      name: result-app-pod
      app: demo-voting-app

  template:
    metadata:
      name: result-app-pod
      labels:
        name: result-app-pod
        app: demo-voting-app
    spec:
      containers:
        - name: result-app
          image: kodekloud/examplevotingapp_result:v1
          ports:
            - containerPort: 80
```

Apêndice D

Implementação do arquivo de *Deploy* da aplicação *voting*

voting-app-Deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: voting-app-Deploy
  labels:
    name: voting-app-Deploy
    app: demo-voting-app
spec:
  replicas: 1
  Selector:
    matchLabels:
      name: voting-app-pod
      app: demo-voting-app

  template:
    metadata:
      name: voting-app-pod
      labels:
        name: voting-app-pod
        app: demo-voting-app
    spec:
      containers:
        - name: voting-app
          image: kodekloud/examplevotingapp_vote:v1
          ports:
            - containerPort: 80
```

Apêndice E

Implementação do arquivo de *Deploy* da aplicação *voting*

worker-app-Deploy.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: worker-app-Deploy
  labels:
    name: worker-app-Deploy
    app: demo-voting-app
spec:
  replicas: 1
  Selector:
    matchLabels:
      name: worker-app-pod
      app: demo-voting-app

  template:
    metadata:
      name: worker-app-pod
      labels:
        name: worker-app-pod
        app: demo-voting-app
    spec:
      containers:
        - name: worker-app
          image: kodekloud/examplevotingapp_worker:v1
```

Apêndice F

Implementação do arquivo de *service* do *Postgres*

Postgres-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: db
  labels:
    name: Postgres-service
    app: demo-voting-app
spec:
  ports:
    - port: 5432
      TargetPort: 5432
  Selector:
    name: Postgres-pod
    app: demo-voting-app
```

Apêndice G

Implementação do arquivo de *service* do *Redis*

Redis-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: Redis
  labels:
    name: Redis-service
    app: demo-voting-app
spec:
  ports:
    - port: 6379
      TargetPort: 6379
  Selector:
    name: Redis-pod
    app: demo-voting-app
```

Apêndice H

Implementação do arquivo de *service* da aplicação *result*

result-app-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: result-service
  labels:
    name: result-service
    app: demo-voting-app
spec:
  Type: LoadBalancer
  ports:
    - port: 80
      TargetPort: 80
  Selector:
    name: result-app-pod
    app: demo-voting-app
```

Apêndice I

Implementação do arquivo de *service* da aplicação *voting*

voting-app-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: voting-service
  labels:
    name: voting-service
    app: demo-voting-app
spec:
  Type: LoadBalancer
  ports:
    - port: 80
      TargetPort: 80
  Selector:
    name: voting-app-pod
    app: demo-voting-app
```

Apêndice J

Implementação da *playbook* do *Ansible* para configurar o servidor com o *Kubernetes*

configurar_Kubernetes.yaml

```
---
# Configurar Kubernetes
- name: TCC DevOps | Configurar Kubernetes
  hosts:
    - "{{ host | default('tcc_DevOps') }}"
  become: yes
  roles:
    - { role: configurar_Kubernetes }

---
# Instalar MicroK8s no Linux
- name: "TCC DevOps | Instalar MicroK8s no Linux"
  shell: "snap install microk8s --classic"

# Adicionar usuario no grupo MicroK8s
- name: "TCC DevOps | Adicionar usuario no grupo MicroK8s"
  shell: "usermod -a -G microk8s $USER"

# Esperar MicroK8s iniciar
- name: "TCC DevOps | Esperar MicroK8s iniciar"
  shell: "microk8s status --wait-ready"

# Habilitar dashboard dns ingress
- name: "TCC DevOps | Habilitar dashboard dns ingress"
  shell: "export LC_ALL=C.UTF-8 && microk8s enable dashboard dns ingress"
```

Apêndice K

Implementação da *playbook* do *Ansible* para realizar o *Deploy* da aplicação no *cluster* de *Kubernetes*

Deploy_aplicacao.yaml

```
---
# Deploy da aplicacao
- name: TCC DevOps | Deploy da aplicacao
  hosts:
    - "{{ host | default('tcc_DevOps') }}"
  become: yes
  roles:
    - { role: Deploy_aplicacao }

---
# Criar repositorio tcc-DevOps
- name: "TCC DevOps | Criar repositorio tcc-DevOps"
  file:
    path: "/opt/tcc-DevOps"
    state: directory
    mode: '0775'
  become: yes

# Copiar arquivos de configuracao do Kubernetes
- name: "TCC DevOps | Copiar arquivos de configuracao do Kubernetes"
  copy:
    src: "{{ item }}"
    dest: "/opt/tcc-DevOps/"
    owner: "root"
    mode: 0600
  with_fileglob:
    - "files/*"

# Deploy da aplicacao de votacao
- name: "TCC DevOps | Deploy da aplicacao de votacao"
  shell: "cd /opt/tcc-DevOps/ && microk8s kubectl apply -f ."
  become: yes
```

Apêndice L

Implementação da *playbook* do *Ansible* para realizar o *Rollback* da aplicação no *cluster* de *Kubernetes*

Rollback_aplicacao.yaml

```
---
# Rollback da aplicacao
- name: TCC DevOps | Rollback da aplicacao
  hosts:
    - "{{ host | default('tcc_DevOps') }}"
  become: yes
  roles:
    - { role: Rollback_aplicacao }

---
# Rollback result e voting
- name: " TCC DevOps | Rollback result e voting"
  become: yes
  shell: "microk8s kubectl rollout undo Deployment.v1.apps/voting-app-Deploy &&
microk8s kubectl rollout undo Deployment.v1.apps/result-app-Deploy"
```

Apêndice M

Implementação da *pipeline* do *GitHub Actions*

main.yaml

```
name: CI-CD

on:
  push:
    branches: [ master ]

  workflow_dispatch:

jobs:
  CI-CD:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v1
      - name: Login to Dockerhub
        uses: docker/login-action@v1
        with:
          username: ${{ secrets.DOCKERHUB_USERNAME }}
          password: ${{ secrets.DOCKERHUB_TOKEN }}

      - name: Build app vote
        id: Docker_Build_vote
        uses: docker/build-push-action@v2
        with:
          context: ./vote
          push: true
          tags: tccDevOps/vote:${{ GitHub.run_number }}

      - name: Build app result
        id: Docker_Build_result
        uses: docker/build-push-action@v2
        with:
          context: ./result
          push: true
          tags: tccDevOps/result:${{ GitHub.run_number }}

      - name: Deploy app vote
        shell: bash
        run: |
```

```
        chmod 400 tcc.pem
        ssh -T -o "StrictHostKeyChecking no" -i "tcc.pem" ubuntu@ec2-15-228-127-130.sa-east-1.compute.amazonAWS.com 'sudo microk8s kubectl set image Deployment/voting-app-Deploy voting-app=tccDevOps/vote:${{ GitHub.run_number }}'

    - name: Deploy app result
      shell: bash
      run: |
        chmod 400 tcc.pem
        ssh -T -o "StrictHostKeyChecking no" -i "tcc.pem" ubuntu@ec2-15-228-127-130.sa-east-1.compute.amazonAWS.com 'sudo microk8s kubectl set image Deployment/result-app-Deploy result-app=tccDevOps/result:${{ GitHub.run_number }}'

    - name: Esperar 15 segundos
      shell: bash
      run: sleep 15

    - name: test app vote
      id: test-vote
      uses: Cypress-io/GitHub-action@v2
      with:
        spec: |
          Cypress/integration/test-vote.js
      continue-on-error: true

    - name: test app result
      id: test-result
      uses: Cypress-io/GitHub-action@v2
      with:
        spec: |
          Cypress/integration/test-result.js
      continue-on-error: true

    - name: Rollback app vote
      if: steps.test-vote.outcome != 'success'
      shell: bash
      run: ssh -T -o "StrictHostKeyChecking no" -i "tcc.pem" ubuntu@ec2-15-228-127-130.sa-east-1.compute.amazonAWS.com "sudo microk8s kubectl rollout undo Deploy voting-app-Deploy"

    - name: Rollback app result
      if: steps.test-result.outcome != 'success'
      shell: bash
      run: ssh -T -o "StrictHostKeyChecking no" -i "tcc.pem" ubuntu@ec2-15-228-127-130.sa-east-1.compute.amazonAWS.com "sudo microk8s kubectl rollout undo Deploy result-app-Deploy"
```

```
- name: Check on failures
  if: steps.test-result.outcome != 'success'|| steps.test-vote.outcome != 'success'
  run: exit 1
```

Apêndice N

Implementação do teste da aplicação *result* em *Cypress*

test-result.js

```
it('works', () => {
    cy.visit('http://15.228.127.130:30002').then(() => {
        cy.title().should('include', 'dogs')
        cy.title().should('include', 'cats')
    })
 })
```

Apêndice O

Implementação do *Teste* da aplicação *vote* em *Cypress*

Test-result.js

```
it('works', () => {
    cy.visit('http://15.228.127.130:30001').then(() => {
        cy.title().should('include', 'dogs')
        cy.title().should('include', 'cats')
    })
 })
```