Revista do *Fórum Gerencial*, v. 1, n. 2, 2021
© Centro Universitário de Patos de Minas
https://revistas.unipam.edu.br/index.php/forumgerencial

A CULTURA *DEVOPS*

Raphael Custodio Soares[1]
Henaldo Barros Moraes[2]

RESUMO: Este artigo tem intuito de apresentar a cultura *DevOps*, abordando as três maneiras que são seus pilares fundamentais como também exemplificar as principais ferramentas que esse novo movimento oferece
PALAVRAS-CHAVE: *DevOps*; três maneiras.

ABSTRACT: This article aims to present the DevOps culture, addressing the three ways its fundamental pillars are as well as exemplifying the main tools that this new movement offers.
KEYWORDS: DevOps; three ways.

1 INTRODUÇÃO

Apesar do movimento *DevOps* está crescendo aos poucos, ele já é uma cultura bastante distribuída. *DevOps* surgiu a partir de uma necessidade: simplificar as regras de negócios através dos esforços coordenados e colaborativos. Em outras palavras, *DevOps* é responder com rapidez as mudanças do mercado de trabalho com seus esforços. Esse movimento é projetado para garantir alta qualidade de software para que chegue aos seus consumidores finais com excelência na qualidade.

A mudança cultural traz consigo confiança, aprendizado contínuo, colaboração entre equipes e alta sinergia. Para que tudo isso ocorra em um intervalo pequeno de tempo e com sucesso, foi criado os pilares do *DevOps* que são chamadas as três maneiras.

De acordo com Antonio Muniz (2020): "A Primeira Maneira possui princípios e práticas que potencializam o fluxo rápido de desenvolvimento para operações, seu foco é executar ações para acelerar o fluxo da esquerda para a direita, visando reduzir o tempo da implantação".

Segue abaixo os benefícios da primeira maneira.

- Tornar trabalho visível;
- Reduzir tamanho dos lotes e intervalos;
- Remover desperdícios e foco no cliente;
- Incorporar qualidade na origem.

DevOps é cinquenta por cento automação e cinquenta por cento colaboração. Neste primeiro pilar como mencionado os benefícios acima pode resumir todos esses atributos como em qualidade na origem que um termo essencial nesse movimento, sendo responsável pela entrega da experiência ao cliente.

De acordo com Gim Kane (2016): "A Segunda Maneira permite fluxo de *feedback* rápido e constante, do cliente para o desenvolvimento, em todos os estágios do fluxo de valor".

[1] Aluno de Sistemas de Informações do UNIPAM. E-mail: raphaelcustodio94@gmail.com.
[2] Mestre em Redes de Computadores pela UFU. E-mail: henaldobarros@gmail.com.

Exige que amplifique o feedback para evitar que problemas ocorram novamente. “Para garantir a satisfação e um ótimo feedback, segue alguns princípios básicos da segunda maneira”.

- Telemetria;
- Desenvolvimento por Hipóteses e Testes A/B;
- Programação em pares;
- Programação sobre os ombros;
- Qualidade próxima da fonte.

O Teste A/B é um dos principais conceitos que se utiliza no *DevOps* referente à segunda maneira. O mesmo vem da área de marketing. Steve Krug (2020) relata que os testes de usabilidade há muito tempo e a ideia básica: se deseja saber se o software ou website é, suficientemente, fácil de usar, assista algumas pessoas enquanto navegam pelo software observando suas dificuldades.

Um dos casos de sucesso de Steven Krug sobre o teste A/B foi usado na Amazon. O teste foi baseado se o usuário compraria no site da Amazon com apenas um clique ou no modo convencional com três cliques. A partir dos resultados do teste, a Amazon adotou, em seu site, a compra em “um clique”, tornando mais fácil a compra pelo site, pois não precisa digitar informações pessoais. Na Amazon o teste A/B é de extrema importância quando se quer lançar um produto novo ou requer alterações de software sem ter um feedback assertivo.

A Terceira maneira de acordo com Antonio Muniz (2020) é mencionada como a criação de uma cultura de alta confiança, que permite correr riscos e potencializar o aprendizado contínuo, possibilitando a adoção de uma cultura de experimentação.

Seguem algumas práticas adotadas na terceira maneira:

- Cultura organizacional: confiança, desempenho e gestão sem culpa ou medo;
- Institucionalizar a melhoria do trabalho diário;
- Transformar descobertas locais em melhorias globais;
- Criar padrões de resiliência no trabalho diário;
- Cultura de aprendizado.

2 REFERENCIAL TEÓRICO

Nesta seção são apresentados conceitos e estudos realizados sobre a cultura *DevOps* e algumas técnicas desse movimento.

2.1 ENTREGA CONTÍNUA

Brian Docker (2020) relata que a entrega contínua permitirá que as equipes de desenvolvimento de software se movam mais rapidamente e se adaptem às necessidades dos usuários com mais rapidez, reduzindo o atrito inerente associado ao lançamento de alterações de software.

2.1.1 Integração Contínua

A integração Contínua é uma das principais rotinas da cultura *DevOps*. Aécio Pires (2019) demonstra que

> Integração Contínua é uma prática que consiste em juntar o código de vários desenvolvedores de uma mesma aplicação com maior frequência, identificar problemas e corrigi-los mais rapidamente, resultando em menos bugs, automatização de testes, feedbacks mais frequentes e entrega mais rápida (PIRES, 2019).

2.3 DOCKER

Docker é um conceito que vem ganhando espaço no mercado de trabalho. Essa ferramenta tem o intuito de compartilhar recursos do kernel ao invés de duplicar como acontece nas máquinas virtuais, levando assim a um problema de performance.

Por ser uma ferramenta *open source*, qualquer pessoa pode visualizar o código e contribuir com melhorias para o *Docker*. "Isso traz maior transparência e faz com que correções de bugs e melhorias aconteçam bem mais rápido do que seria em um software proprietário com uma equipe bem menor e poucos cenários de testes" (FERNANDO; ANDRÉ, 2018).

2.4 JENKINS

Como o *DevOps* aborda bastante a parte de automação, deve-se usar o conceito de *Pipeline* e também *CI/CD* para que tudo ocorra de forma automatizada desde submeter o código até levar para produção.

Pipeline de acordo com Aécio Pires (2019) é uma execução em sequência de vários jobs, que podem ter dependência entre si. Um job que faz o *build* de uma aplicação web dependa da execução com sucesso do job anterior ou seja para passar para próxima etapa só será possível se estes jobs forem aprovados. Apenas o *pipeline* não é suficiente para consolidar a implantação e entrega contínuas. Para que tudo ocorra como previsto, a ferramenta Jenkins é peça fundamental para que haja sucesso.

O Jenkins é uma ferramenta de código-fonte aberto e desenvolvida em Java. Aécio Pires (2019) afirma que Jenkins pode ser usado para realizar o *Continuous Integration, Continuous Delivery e Continuous Deployment* em projetos de diversos tamanhos e com linguagens e tecnologias variadas, tais como: .NET, Ruby, Groovy, Grails, PHP, Python, Java e outros.

2.5 BALANCEADOR DE CARGA

Outro grande conceito que este estudo tem a oferecer é o de *open-source* ou simplesmente código livre que se baseia em uma licença, *General Public Licene* (GPL), que garante benefícios de software como.

- A liberdade de executar o programa, para qualquer propósito;

- A liberdade de estudar como o programa funciona;
- A liberdade de redistribuir cópias;
- A liberdade de aperfeiçoar o programa e liberar os seus aperfeiçoamentos.

Como se trata de um código livre, qualquer pessoa pode contribuir para que o software cresça ainda mais, além de personalizar seu próprio software. Segundo o autor Christian Leoanardo (2020), o simples fato de ser disponibilizado sob a licença GPL permitiu que outros programadores adotassem o projeto, passando a contribuir com melhorias e correções.

A partir desse ponto de vista, o projeto desenvolvido adotou a ferramenta *HAPROXY* que é disponibilizado pela licença GLP, de acordo com o site logz.io "The 5 Best Open Source Load" o *haproxy* se encontra na lista de melhores ferramentas para realizar o balanceamento de carga entre servidores.

O balanceador distribui cargas de trabalho em vários servidores virtuais aumentando a disponibilidade e a tolerância a falhas de seus aplicativos (AMAZON, 2020).

2.6 GIT

Um dos grandes requisitos para *DevOps* ou até mesmo para os desenvolvedores é saber como controlar a versão do código, com isso o *Git* se torna peça fundamental.

De acordo com o autor Christian Leoanardo (2020), o *Gıt* foi criado por Lınus Torvalds em 2005 para desenvolver o kernel Linux. Também é usado como uma ferramenta de controle de versão distribuída importante para o *DevOps*.

Ryan Hodson (2020) afirma que o *GIT* é um Sistema de Controle de Versão (VCS) criado para uma única tarefa: gerenciar alterações em seus arquivos. Permite que você acompanhe todas as alterações pelas quais um projeto de software passa, bem como de onde essas alterações vieram".

Como mencionado acima, o *Git* é uma ferramenta essencial para gerenciar grandes projetos, mas também pode abrir uma vasta gama de possibilidades para controlar o fluxo de trabalho pessoal como senhas salvas a partir de um bloco de notas.

2.7 AZURE

DevOps está cada vez mais presente no mercado, este artigo usou as ferramentas mais atualizadas até o momento, como por exemplo o conceito de nuvem ou popularmente conhecido como *cloud*.

Cloud surgiu como a transformação mais importante na computação em décadas o autor Kirshc (2020) retrata que "A nuvem está mudando a forma como as organizações planejam fornecer serviços a seus clientes para aumentar o ritmo dos negócios e oferecer serviços novos e inovadores".

Todo o projeto será baseado em nuvem. Atualmente, existem inúmeros fornecedores que oferecem este serviço. Devido ao custo e benefício e, também, à alta disponibilidade, a plataforma escolhida foi Azure.

Azure é uma plataforma de computação em nuvem pública da Microsoft que oferece muitos serviços combinados com uma estimativa, análise, armazenamento e rede. George Prestonship (2020), menciona que os usuários podem escolher entre esses serviços para ampliar e dimensionar novos programas ou executar pacotes atuais, dentro da nuvem pública.

2.8 DYNATRACE

Para conseguir medir a experiência do usuário e receber um feedback coerente, o projeto utilizou o conceito de telemetria.

De acordo com Abel Rubio (2019), que se faz uma analogia com fórmula 1, a telemetria considera todo o sistema e permite que os dados coletados pelos sensores do carro sejam enviados ao computador dos engenheiros. É um sistema imediato. Conforme as coisas acontecem no carro, a equipe vai recebendo feedback para poder analisá-las e tomar decisões posteriormente.

Esse contexto se encaixa perfeitamente no ambiente em que o carro é o software. Na medida que o cliente vai encontrando erros no software, o programa *Dynatace,* que utiliza o conceito de telemetria, coleta essas possíveis falhas de desenvolvimento e retorna para os programadores, para que se realize uma ação corretiva de bugs solucionando assim as falhas.

O *Dynatrace* tem por objetivo monitorar a aplicação em um ambiente de produção, homologação como também detalhar a infraestrutura como servidores, discos rígidos, cargas da unidade de processamentos tudo em tempo real.

2.9 TERRAFORM

O software para automação da infraestrutura usado em todo projeto para criar recursos na nuvem Azure de forma automática foi escolhido o *Terraform*, a partir dessa ferramenta é possível integrar várias nuvens ao mesmo tempo com apenas um código fonte, criando várias maquinas virtuais ou recursos individuais facilitando o desenvolvimento.

3 METODOLOGIA

O presente estudo foi desenvolvido a respeito da cultura *DevOps* voltado para o conceito das três maneiras e como se aplicam no ambiente de produção.

Foi construído um ambiente de produção abordando as três maneiras aplicando, também, as ferramentas do movimento *DevOps* para consolidar a ideia dos pilares para uma melhor análise.

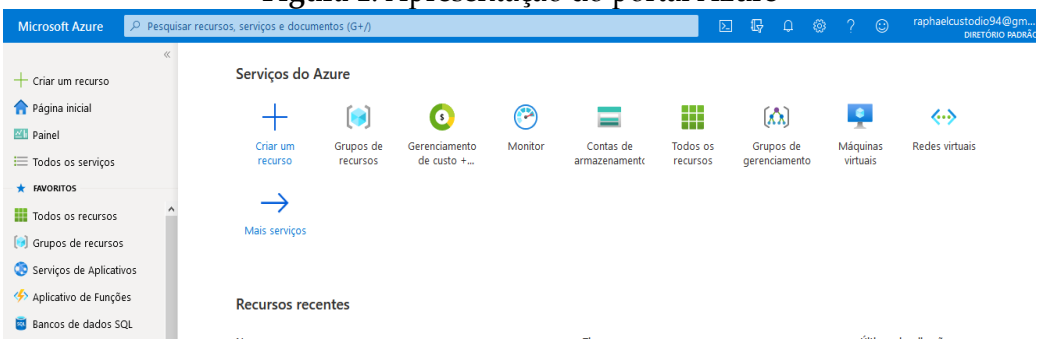
Neste sentido, foram consultados livros, artigos e projetos desenvolvidos para que o usuário tenha uma melhor disseminação do conteúdo.

Em uma última etapa, observa o estudo como um todo desde eliminar funções manuais até usar a telemetria para ter um rápido *feedback* o que acelerar a integração contínua e entrega contínua, toda aplicação foi utilizado na plataforma Azure uma das mais conceituadas no meio de arquitetura em nuvem.

4 DESENVOLVIMENTO E RESULTADOS

Nessa primeira fase, o acesso ao portal Azure é fundamental devido ser a base de todo o projeto. Para inscrever na plataforma é bastante fácil, apenas ter em mãos um cartão de crédito sendo possível desfrutar de todo conteúdo que Azure oferece.
Em casos de uma conta estudantil, ganhe um saldo de até R$200 mensais para analisar todos os serviços que a plataforma disponibiliza. A Figura 1 demonstra a página inicial como também seus principais recursos.

Figura 1: Apresentação do portal Azure



Fonte: Dados do trabalho.

Na Figura 1, é possível perceber alguns serviços em destaque na plataforma do Azure como grupos de recursos, máquinas virtuais e redes virtuais.

A criação do grupo de recursos é como se fosse a base das máquinas virtuais, sendo possível adotar um *IP Andress* estático ou dinâmico, escolha da localização do grupo de recursos o que pode impactar diretamente o custo em torno do projeto e por último ajuste de regras de firewall.

Para que todo esse conteúdo do grupo de recursos seja criado rapidamente e com alta qualidade usou a ferramenta *Terraform* que por padrão tem função de *MultiCloud,* onde a ferramenta faz integração com vários fornecedores de nuvem com apenas um código.

O *Terraform* tem o objetivo de simplificar a infraestrutura como código, ao invés de criar vários grupos de recursos para cada serviço de nuvem, o que demanda tempo e trabalho. O *Terraform* usa um método de código universal, ganhando qualidade na origem, parte essencial do *DevOps*.

A Figura 2 tem o objetivo de apresentar o código usado para criação de um grupo de recursos usando *Terraform*.

Figura 2: Código para implantação do grupo de recursos

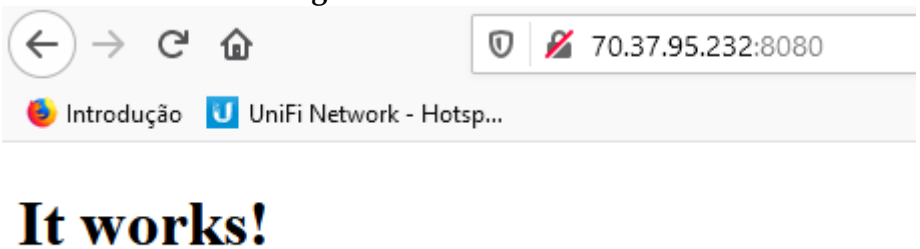
```
provider "azurerm" {
  version = "=2.20.0"
  features {}
}
# Create a resource group
resource "azurerm_resource_group" "tccgroup" {
  name     = "tccgroup"
  location = "West Europe"
}
#Create Virtual Network
resource "azurerm_virtual_network" "mynetwork"{
  name = "NetworkTCC"
  address_space = ["10.0.0.0/16"]
  location = "West Europe"
  resource_group_name = azurerm_resource_group.tccgroup.name
}
#Create Firewall Police
resource "azurerm_network_security_group" "tccngs"{
name = "tccsecuritygroup"
location = azurerm_resource_group.tccgroup.location
resource_group_name = azurerm_resource_group.tccgroup.name
security_rule {
  name = "http"
  priority = 100
  direction ="Inbound"
  access ="Allow"
  protocol = "TCP"
  source_port_range = "*"
  destination_port_ranges = [22,80,443]
  source_address_prefix = "*"
  destination_address_prefix = "*"
}
```

Fonte: Dados do trabalho.

O próximo objetivo é criar máquina virtual utilizando o sistema operacional *Ubuntu* na versão 18.0. Após definir o host e a hora de criar contêineres, um para hospedar o website com intenção de *deploy contínuo* e um segundo contêiner para testar o balanceador de carga *haproxy*.

Para instalar o site que tem como propósito geral testar CI/CD deve-se primeiro aplicar as configurações do apache. Para isso baixe a imagem httpd com o comando "Docker pull httpd". Ao executar o comando, o Docker vai baixar a imagem para a máquina virtual e em seguida execute com "Docker run –name test -p 8080:80 httpd". Caso tudo ocorra perfeitamente, vai aparecer a imagem como na Figura 3.

Figura 3: Container Online



Fonte: Dados do trabalho.

O *Github* é responsável pelo controle de versão do projeto, servido como um repositório para armazenar um site e também por controlar as alterações feitas pela equipe de desenvolvimento, o projeto usou o *Github* para controlar a estrutura de código e também para aplicar o conceito de integração continua e entrega continua junto com o *Jenkins.* A Figura 4 abaixo demostra a estrutura básica de um *commit*.

Figura 4: Commit no Github

Fonte: Dados do trabalho.

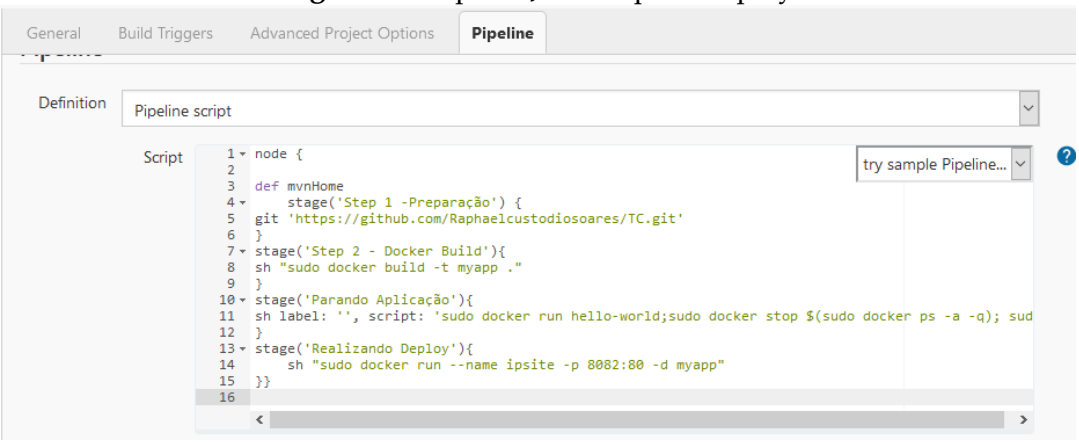
A Figura 4 representa um *commit* simples, porém deve destacar o item em vermelho e a linha em verde sendo também representadas pelos símbolos de menos ou mais.

A linha em vermelho significa que o texto foi retirado e em seguida acrescentado uma nova linha destacada com a cor verde. Pode dizer também que houve uma inclusão de código e uma exclusão. Esse *commit* demonstra apenas o básico. Hoje, os desenvolvedores utilizam para analisar cuidadosamente toda estrutura do projeto quando se trabalha em equipe, sabendo quem alterou, em qual linha e quando, em caso de bug, o que deixa o sistema inoperante é possível encontrar a falha rapidamente.

O Jenkins junto com o *Github* promove o *CI/CD.* Imagine que uma grande empresa de varejo onde suas vendas são realizadas através de um e-commerce está prestes a entrar na *Black Friday*. Antigamente para ajustar todos os preços do site, promoções, cupons de descontos, layout da página, menu personalizado entre outros até o horário da Black Friday, o site deveria ficar fora do ar para fazer todos os ajustes. Sendo assim, a perda da receita é extremamente alta por ser a promoção mais esperada do ano. Com *CI/CD* já é possível ajustar em tempo real as alterações, basta ter uma ferramenta que faça a integração entre os ajustes de homologação até o ambiente de produção.

A Figura 5 representa o código do Jenkins que busca no repositório *Github* e em seguida realizar o *Deploy* no *Docker*.

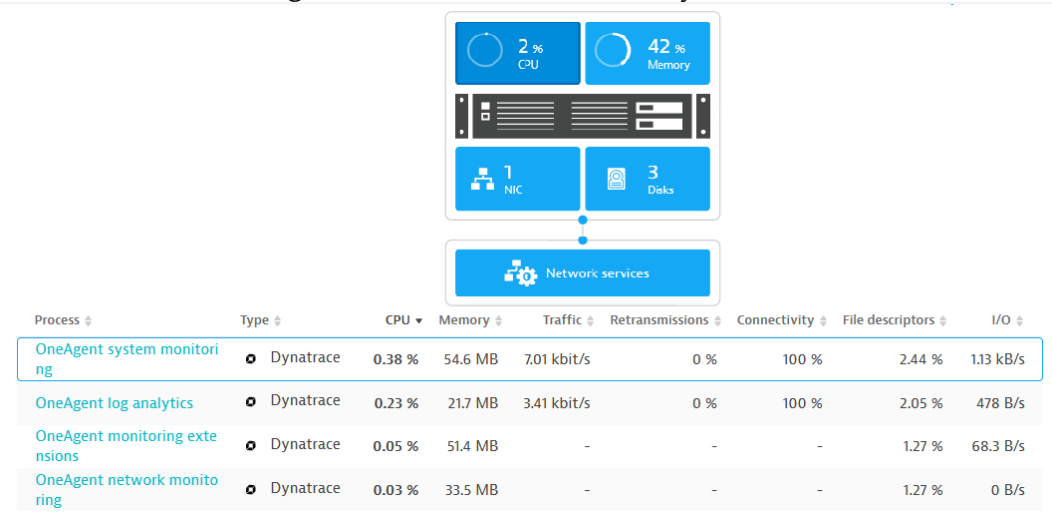
Figura 5: Script do Jenkins para Deploy



Fonte: Dados do trabalho.

Um grande fator para o *DevOps* é sempre receber feedbacks rápidos desde de clientes até mesmo da infraestrutura. *Dynatrace* foi usado para monitorar o ambiente de produção como por exemplo percentual de memória RAM, unidade de processamento e discos físicos. A Figura 6 apresenta todos os dados monitorados pela ferramenta *Dynatrace* em tempo real.

Figura 6: Monitoramento com Dynatrace



Fonte: Dados do trabalho.

5 CONCLUSÃO

O artigo demonstrado objetivou descrever a cultura *DevOps* como também a introdução das três maneiras, um passo fundamental para quem deseja seguir a profissão *DevOps*. Ao entender como funciona essa metodologia é possível estruturar um fluxo de projeto com mais rapidez e menos suscetível a falhas.

Com intuito de consolidar ainda mais o aprendizado, foi desenvolvido exemplos práticos em cada pilar do *DevOps*. As ferramentas utilizadas neste artigo são

as mais atualizadas no mercado de trabalho, o que proporciona um melhor aprendizado e uma melhor inclusão para quem deseja ingressar na carreira de *DevOps*.

Todas ferramentas mencionadas neste artigo podem se encontrar no *Gartner* que hoje é referência mundial em destaque de melhores ferramentas do mundo. A partir do Azure, foram criadas máquinas virtuais para consolidar o mecanismo de balanceamento de carga. As máquinas também foram utilizadas para demonstrar a integração da automação entre o Jenkins e *Github*, onde ao alterar no *Github* o site escrito em *Html* é totalmente modificado em segundos. Com o auxílio do balanceador de cargas, pode-se obter atualizações constantemente no site para atrair novos clientes e, também, proporcionar uma plataforma sem interrupção.

REFERÊNCIAS

AMAZON**. Elastic Load Balancing**: User Guide. Seattle 2020.

BROWN, Mike. **VMware vSphere 6.7 Data Center Design Cookbook**: over 100 practical recipes to help you design a powerful virtual infrastructure based on vSphere. 3. ed. Birmingham, 2019.

DOCKER, Brian. **KUBERNETES**: a simple guide to master kubernetes for beginners and advanced users. 2020.

FERNANDO, Jeferson; ANDRÉ, MARCUS. **Descomplicando Docker**. 2 ed. Rio De Janeiro, 2018.

LEVANTAMENTO do PageGroup mostra 38 profissões em alta em 2020. **G1**. 2020. Disponível em: https://g1.globo.com/economia/concursos-e-emprego/noticia/2020/01/14/levantamento-do-pagegroup-mostra-38-profissoes-em-alta-em-2020.ghtml. Acesso em: 17 ago. 2020.

HODSON, Ryan. **Ry's Git Tutorial**. Sugar Land. 2020.

HUMBLE, Jez; FARLEY, David. **Entrega contínua**: como entregar software de forma rápida e confiável. Poro Alegre: Bookman, 2014.

HURWITZ, Judith; KIRSCH, Daniel. **Cloud computing for dummies**. 2. ed. New Jersey. 2020.

JEE, Charlote; MACAULAY, Thomas. **Falhas e invasões prejudicam empresas de diferentes setores**. 2018. Disponível em: https://computerworld.com.br/2018/07/12/10-grandes-falhas-da-tecnologia-nos-ultimos-anos. Acesso em: 28 set. 2020.

KIM, Gene; DEBOIS, Patrick; WILLIS, John; HUMBLE, Jez.**The DevOps Handbook**: how to create world-class agility, reliability, and security in technology organizations. IT Revolution Press. Portland, 2016.

KRUG, Steve; CROCE, DANIEL. **Não me faça pensar**. São Paulo: Alta Books, 2020.

LEONARDO, Christian. **Git**: a fast and easy guide to version control. 2020

MORIMOTO, Carlos. **LINUX, Guia Prático**, 2020.

MUNIZ, Antonio; SANTOS, Rodrigo; IRIGOYEN, ANALIA; MOUTINHO, RODRIGO **Jornada DevOps**: unindo cultura ágil, lean e tecnologia para entrega de software com qualidade. 2. ed. Rio De Janeiro: Brasport, 2020.

PIRES, Aecio; MILITÃO, Janaina. **Integração contínua com Jenkins**: automatize o ciclo de desenvolvimento, testes e implantação de aplicações. São Paulo: Novatec Editora. 2019.

PRESTONSHIP, George. **Azure**: essential guide to learn microsoft azure fundamentals, cloud, security, machine learning and devops. 2020.

RANJAN, Ankit**. Using the devops three ways to do laundry**. *Online*. Disponível em: https://freshservice.com/itsm/phoenix-project-three-ways-devops-blog/. Acesso em: 15 jul. 2020.

REBACK, Gedalyah. **The 5 best open source load balancers**. Disponível em: https://logz.io/blog/best-open-source-load-balancers/. Acesso em: set. 2020.

RUBIO Abel. **Telemetry in Formula 1**. 2019.