Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Engenharia de Software

Modernização de software: Um estudo de caso sobre a aplicação de práticas de DevOps

Autor: Caio Vinícius Fernandes de Araújo e
Lucas Dutra Ferreira do Nascimento

Orientador: Professora Doutora Carla Silva Rocha Aguiar

Brasília, DF

2022

Caio Vinícius Fernandes de Araújo e
Lucas Dutra Ferreira do Nascimento

Modernização de software: Um estudo de caso sobre a aplicação de práticas de DevOps

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade UnB Gama – FGA

Orientador: Professora Doutora Carla Silva Rocha Aguiar

Brasília, DF
2022

Caio Vinícius Fernandes de Araújo e
Lucas Dutra Ferreira do Nascimento
Modernização de software: Um estudo de caso sobre a aplicação de práticas de DevOps/ Caio Vinícius Fernandes de Araújo e
Lucas Dutra Ferreira do Nascimento. – Brasília, DF, 2022-
61 p. : il. (algumas color.) ; 30 cm.

Orientador: Professora Doutora Carla Silva Rocha Aguiar

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade UnB Gama – FGA , 2022.

1. Modernização de Software. 2. DevOps. I. Professora Doutora Carla Silva Rocha Aguiar. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Modernização de software: Um estudo de caso sobre a aplicação de práticas de DevOps

CDU 02:141:005.6

Este trabalho é dedicado a todas as pessoas
que não desistem do objetivo final, apesar das intempéries.

Agradecimentos

Gostaríamos de agradecer a todas e todos professores que participaram na nossa formação pessoal e profissional durante o período da graduação, principalmente a professora Carla, que aceitou ser nossa orientadora nesse projeto que propomos, fazendo-o tornar realidade.

Eu, Lucas, gostaria de agradecer aos meus pais Eurenice e Abrahão, pela educação, ensinamentos e direcionamentos que foram a mim dados para que eu me tornasse a pessoa que sou hoje e pudesse chegar nessa etapa da graduação de Engenharia de Software. Agradeço em especial ao meu companheiro de trabalho Caio, pelo companheirismo, paciência, cumplicidade e amizade durante todo o tempo de desenvolvimento desse projeto de conclusão de curso e que se extendem para diversos outros momentos, também por aceitar realizar essa etapa tão importante ao meu lado. Além disso, mas não menos importante, agradecer aos meus amigos, Danilo, Gabriel, Guilherme, Talles, por serem meu apoio em momentos difíceis e comemorarem minhas as vitórias ao meu lado. Por fim, agradeço a todas pessoas que estiveram comigo ao longo dessa caminhada e que, direta ou indiretamente, fizeram parte esta história, pois sei que cada momento contribuiu para que eu pudesse me tornar uma pessoa ou profissional melhor.

"Os filósofos interpretam o mundo,
mas o que importa é transformá-lo." (Karl Marx)

Resumo

Com o avanço acelerado de novas tecnologias de desenvolvimento de software, softwares de grande porte desenvolvidos com a finalidade de serem pilares de negócios se tornam obsoletos antes das empresas estarem preparadas para substituí-los. Existem várias perspectivas que tornam um software defasado, ou em outras palavras, caracterizam um software como legado, entre esses aspectos estão a desatualização de dependências, ausência de práticas DevOps e infraestrutura ultrapassada. Com essa defasagem dos sistemas, os problemas são cada vez mais presentes e com o passar do tempo, as aplicações podem apresentar vulnerabilidades de segurança, incompatibilidade com novas tecnologias, além de dificuldades no contexto de *build* e *deploy*. Um software obsoleto nesse aspecto costuma ser mais sujeito a erros. Ademais, atualizações de código para o contexto de produção se tornam menos frequentes. Este trabalho busca, por meio de um estudo de caso, aplicar técnicas de modernização de software em uma aplicação em contexto de produção e com uma equipe de desenvolvedores ainda atuante, para, dessa forma, ter como resultado as vantagens e desvantagens das técnicas aplicadas, complexidade envolvida ao se implantar técnicas DevOps, a complexidade relacionada a realocação da infraestrutura de um projeto através da utilização de infraestrutura como código, além do software escolhido modernizado nos moldes definidos.

Palavras-chave: DevOps. Modernização de software. Software Legado. Infraestrutura como código. Infraestrutura.

Abstract

With the fast advance of new technologies, large software developed to serve as main pillars of business become obsolete before companies are ready to replace them. Several perspectives make a software outdated, or in other words, characterize a software as legacy, among these aspects are out-of-date dependencies, lack of DevOps practices, and outdated infrastructure. With this system's outdatedness, the problems are more frequent as time goes by applications may present security vulnerabilities, incompatibility with new technologies, besides difficulties in the build and deploy context. An obsolete software in this aspect is usually more prone to errors, moreover, code updates for the production context become less frequent. This project seeks, through a case study, to apply software modernization techniques in an application in a production context and with an active team of developers, so that it is possible to analyze the advantages and disadvantages of the applied techniques, the complexity involved when implementing DevOps techniques and also the complexity related to the reallocation of the infrastructure of a project through the use of infrastructure as code.

Key-words: DevOps. Software Modernization. Legacy Software. Infrastructure-as-code. Infrastructure.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

TI	Tecnologia da Informação
CI	Continuous Integration
CD	Continuous Deploy/Delivery
SO	Sistema Operacional
TCC	Trabalho de conclusão de curso
IaC	Infrastructure as Code
SQL	Structured Query Language
AWS	Amazon Web Services
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets

Sumário

Introdução

Atualmente, a tecnologia avança em ritmo acelerado, impactando diferentes áreas de aplicação, principalmente a indústria de software. A tendência são os *frameworks* (abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica), bibliotecas e ferramentas serem cada vez melhores e mais úteis, acompanhando o avanço do conhecimento.

Durante esse avanço, software de grande porte são desenvolvidos com a expectativa de suprirem as necessidades de empresas por um longo período de tempo, levando em consideração o esforço e o investimento aplicado para construção dos mesmos. Porém, como citado anteriormente, os avanços das tecnologias relacionadas a área de software são constantes, logo, o software desenvolvido com a expectativa de ser utilizado a longo prazo, tem seu ciclo de vida e eficiência rapidamente reduzidos, devido a desatualização de ferramentas, dependências, práticas, servidores, entre outros. Essas aplicações desatualizadas também são chamadas de softwares legado, ou seja, um sistema que é desenvolvido em linguagens e tecnologias que não são mais utilizadas para desenvolver novos sistemas e, tipicamente, ainda são essencias para o negócio da organização (SOMMERVILLE, 2016).

Com essa defasagem dos sistemas, os problemas são cada vez mais presentes. Com o passar do tempo, essas aplicações podem apresentar ameaças a segurança dos dados, incompatibilidade com novas tecnologias (MCCARTY, 2020), além de dificuldades no contexto de *build* e *deploy*, um software obsoleto nesse aspecto costuma ser mais sujeito a erros, ademais, atualizações de código para o contexto de produção se tornam menos frequentes (BIRCHALL, 2016).

Esse trabalho de conclusão de curso tem como objetivo realizar um estudo de caso sobre a aplicação de técnicas de modernização de software, enfatizando a integração contínua, *deployment* contínuo e infraestrutura como código em uma aplicação legado. Visando assim analisar as vantagens e adversidades da utilização desses procedimentos em um software inserido em um contexto de produção, com desenvolvedores ativos e pleno acesso à infraestrutura.

Entre as etapas a serem realizadas estão a revisão bibliográfica, com a finalidade de investigar as pricipais referências e conteúdos relacionados a área de estudo discutida nesse trabalho, escolha das técnicas de modernização de software a serem aplicadas, definição do projeto alvo que serão implementadas as técnicas determinadas, seleção das ferramentas de automação necessárias, implementação das técnicas estabelecidas e análise qualitativa do processo de implementação.

Logo, o resultado esperado desse trabalho consiste na modernização do software

legado escolhido, utilizando das técnicas levantadas e a análise de algumas métricas, entre elas podemos elencar:

- a) Análise da complexidade envolvida ao se implantar as técnicas de DevOps;
- b) Análise da complexidade de implantação da cultura DevOps dentro de uma equipe de software;
- c) Benefícios atrelados a utilização de técnicas de DevOps em um software legado em produção;
- d) Elencar vantagens e desafios encontrados ao realocar a infraestrutura utilizando IaC.

1 Referencial Teórico

1.1 DevOps

DevOps é uma abordagem de cultura, automação e design de plataforma que tem como objetivo agregar mais valor aos negócios e aumentar sua capacidade de resposta às mudanças por meio de entregas de serviços rápidas e de alta qualidade. Isso tudo é possível por meio da disponibilização iterativa e rápida de serviços de TI. Adotar o DevOps significa conectar aplicações legadas a uma infraestrutura moderna e nativa em nuvem. Um composto de Dev (desenvolvimento) e Ops (operações), o DevOps é a união de pessoas, processos e tecnologias para fornecer continuamente valor aos clientes.

O DevOps permite que atribuições anteriormente isoladas – desenvolvimento, operações de TI, engenharia da qualidade e segurança – atuem de forma coordenada e colaborativa para gerar produtos melhores e mais confiáveis. Ao adotar uma cultura de DevOps em conjunto com as práticas e ferramentas de automação DevOps , as equipes ganham a capacidade de responder melhor às necessidades dos clientes, aumentar a confiança nas aplicações que constroem e cumprir as metas empresariais mais rapidamente.

DevOps é um esforço colaborativo e multidisciplinar dentro de uma organização para automatizar a entrega contínua de novas versões de software, garantindo sua exatidão e confiabilidade (LEITE et al., 2020).

1.1.1 Integração, Entrega e Implantação Contínua

Práticas contínuas, ou seja, integração, entrega e implantação contínuas, são as práticas do setor de desenvolvimento de software que permitem que as organizações lancem novos recursos e produtos com frequência e confiabilidade. Com o crescente interesse e literatura sobre práticas contínuas, é importante revisar e sintetizar sistematicamente as abordagens, ferramentas, desafios e práticas relatadas para a adoção e implementação de práticas contínuas. A Integração Contínua (CI) é uma prática de desenvolvimento amplamente estabelecida na indústria de desenvolvimento de software, na qual os membros de uma equipe integram e mesclam o trabalho de desenvolvimento (por exemplo, código) com frequência. A CI permite que as empresas de software tenham ciclos de lançamento mais curtos e frequentes, melhorem a qualidade do software e aumentem a produtividade de suas equipes. Esta prática inclui a construção de testes de software automatizados. (CHEN, 2015)

A Entrega Contínua (CD) visa garantir que um aplicativo esteja sempre no estado pronto para produção após passar com sucesso em testes automatizados e verificações

de qualidade. A CD emprega um conjunto de práticas, por exemplo, CI e automação de implantação para entregar software automaticamente para um ambiente semelhante à produção. De acordo com (CHEN, 2015), essa prática oferece vários benefícios, como redução do risco de implantação, redução de custos e obtenção de feedback do usuário mais rapidamente.

A prática de implantação contínua vai um passo além e implanta automaticamente e continuamente o aplicativo para ambientes de produção ou de clientes. Há um debate robusto nos círculos acadêmicos e industriais sobre a definição e distinção entre implantação contínua e entrega contínua. O que diferencia a implantação contínua da entrega contínua é um ambiente de produção (ou seja, clientes reais): o objetivo da prática de implantação contínua é implantar automática e constantemente todas as alterações no ambiente de produção (SHAHIN; BABAR; ZHU, 2017).

1.1.2 Pipelines de Integração contínua (CI) e Entrega Contínua (CD)

Pipelines de CI/CD são uma série de etapas que devem ser executadas para entregar uma nova versão de software. Os pipelines de integração contínua/entrega contínua (CI/CD) são uma prática focada em melhorar a entrega de software usando uma abordagem DevOps ou engenharia de confiabilidade de site (SRE).

Estes pipelines introduzem monitoramento e automação para melhoraria do processo de desenvolvimento de aplicativos, principalmente nas fases de integração e teste, bem como durante a entrega e implantação. Embora seja possível executar manualmente cada uma das etapas de CI/CD, o verdadeiro valor dos pipelines é obtido por meio da automação.

As etapas que formam um pipeline de CI/CD são subconjuntos distintos de tarefas agrupadas no que é conhecido como estágio de pipeline. Os estágios típicos do pipeline incluem:

- Teste - O estágio em que o código é testado. A automação aqui pode economizar tempo e esforço. Aqui, podem ser executados os vários tipos de testes existentes, como teste unitátio, teste de integração, teste de fumaça, etc.
- Build - O estágio em que o aplicativo é compilado.
- Release - O estágio em que o aplicativo é entregue ao repositório.
- Deploy - Neste estágio, o código é implantado em ambiente de produção.
- Validação - As etapas para validar uma compilação são determinadas pelas necessidades de sua organização. Ferramentas de verificação de segurança de imagem, como

Clair, podem garantir a qualidade das imagens comparando-as com vulnerabilidades conhecidas.

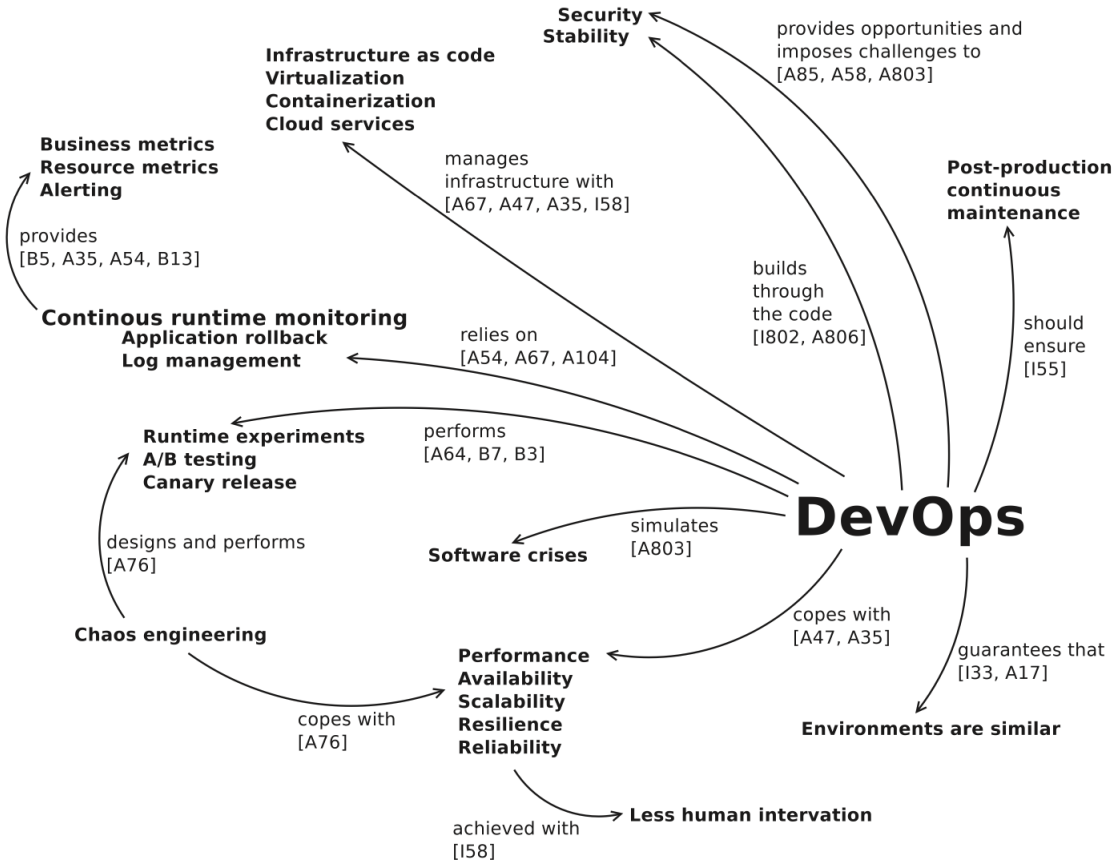

Figura 1 – Runtime category of concepts - Fonte: (LEITE et al., 2020)

1.2 Engenharia de Confiabilidade de Sites - Site Reliability Engineering (SRE)

O SRE é uma estrutura organizacional para implementacao do devops, no qual um time de plataforma, ou SRE, prove infraestrutura, ferramentas e serviços de automação para os times de produto. (LóPEZ-FERNáNDEZ et al., 2021)

A Engenharia de Confiabilidade do Site representa uma ruptura significativa com as práticas recomendadas existentes do setor para gerenciar serviços grandes e complexos. Um conjunto de princípios, um conjunto de práticas, um conjunto de incentivos e um campo de atuação dentro da disciplina maior de engenharia de software.

Em geral, uma equipe de SRE é responsável pela disponibilidade, latência, desempenho, eficiência, gerenciamento de mudanças, monitoramento, resposta a emergências e planejamento de capacidade de seu(s) serviço(s). Codificam regras de engajamento e

princípios de como as equipes de SRE interagem com seu ambiente - não apenas o ambiente de produção, mas também as equipes de desenvolvimento de produtos, as equipes de teste, os usuários e assim por diante. Essas regras e práticas de trabalho ajudam a manter o foco no trabalho de engenharia, em oposição ao trabalho de operações.

1.2.1 Runtime

Não basta entregar continuamente novas versões, mas também é necessário que cada nova versão seja escalável e confiável. Como podemos observar na Figura 1. A categoria de conceitos Runtime traz alguns dos resultados desejados, como desempenho, disponibilidade, escalabilidade, resiliência e confiabilidade. Que podem ser alcaçados com o uso de infraestrutura como código, virtualização, conteinerização, serviços em nuvem e monitoramento (LEITE et al., 2020).

1.2.2 Infraestrutura como código

Infraestrutura como código (IaC) é o processo de provisionamento de infraestrutura com arquivos de configuração. Os profissionais consideram o IaC como um pilar fundamental para implementar as práticas de DevOps , como é possível observar na Figura 1, o que os ajuda a entregar rapidamente software e serviços aos usuários finais. Organizações de tecnologia da informação como GitHub, Mozilla, Facebook, Google e Netflix, adotam o IaC e utilizam esse mecanismo para provisionar instâncias baseadas em serviços de nuvem, como Amazon Web Services (AWS), mas também é possível utilizar a infraestrutura como código para gerenciar bancos de dados e gerenciar contas de usuários em instâncias de computação locais e remotas, por exemplo (RAHMAN; MAHDAVI-HEZAVEH; WILLIAMS, 2019).

A IaC é muito utilizada juntamente com serviços de nuvem para prover a infraestrutura como serviço necessária ao ambiente de execução do software. Algumas das ferramentas mais conhecidas e utilizadas pela comunidade são o Terrafom[1], Ansible[2], Chef[3] e Puppet[4], cada um com sua especialidade. Tais ferramentas podem ser utilizadas com uma variedade de provedores de nuvem, ou até mesmo para configurar uma infraestrutura local . Outras opções, agora proprietárias, de cada provedor de serviços de nuvem são o AWS CloudFormation (Amazon Web Services)[5], Azure Resource Manager (Microsoft Azure)[6] e Google Cloud Deployment Manager (Google Cloud Provider)[7].

[1] Terraform
[2] Ansible
[3] Chef
[4] Puppet
[5] AWS CloudFormation
[6] Azure Resource Manager
[7] Google Cloud Deployment Manager

1.2.3 Containerização e Virtualização

A computação em nuvem conta com técnicas de virtualização para alcançar a elasticidade de recursos compartilhados em larga escala. As máquinas virtuais (VMs) fornecem sistemas operacionais virtualizados. Já os contêineres, em vez de virtualizar o hardware subjacente, eles virtualizam o sistema operacional (normalmente Linux ou Windows) para que cada contêiner individual contenha apenas a aplicação, suas bibliotecas e dependências.(EDUCATION, 2021) Eles consomem menos recursos e tempo, por isso foram sugeridos como uma solução para pacotes de softwares mais interoperáveis na nuvem.

Embora VMs e contêineres sejam técnicas de virtualização, elas resolvem problemas diferentes. Os contêineres são ferramentas para entregar software – ou seja, eles têm um foco de plataforma como serviço (PaaS) – de forma portátil, visando maior interoperabilidade e ainda utilizando princípios de virtualização de Sistemas Operacionas. VMs, por outro lado, são sobre alocação e gerenciamento de hardware (máquinas que podem ser ligadas/desligadas e provisionadas) — ou seja, há um foco de infraestrutura como serviço (IaaS) na virtualização de hardware. Containers podem ser usados como substitutos para VMs onde a alocação de recursos de hardware é feita através de containers por meio da componentização de cargas de trabalho e políticas de dimensionamento automático entre nuvens (PAHL, 2015).

Como podemos observar na Figura 1, o DevOps tem como um de seus pilares, gerenciar a infraestrutura com Virtualização e Conteinerização.

1.2.4 Serviços de Nuvem

Provedores de serviços de nuvem fornecem recursos escaláveis, sob demanda e virtualizados para os usuários. Os usuários podem usar um conjunto compartilhado de recursos de computação provisionados com seus esforços mínimos de gerenciamento. Além disso, os recursos são fornecidos pelo modelo pague-pelo-uso e os usuários pagam apenas pelos recursos que usam. Evitar o provisionamento excessivo e insuficiente de recursos e reduzir o custo de implantação de hardware podem ser considerados algumas motivações para as empresas levarem seus negócios para as nuvens. Vários modelos de serviço, nomeados software como serviço (SaaS), plataforma como serviço (PaaS) e infraestrutura como serviço (IaaS) são propostos para nuvens. Além disso, nuvens privadas, comunitárias, públicas e híbridas podem ser consideradas modelos de implantação para nuvens. Os usuários podem escolher um desses serviços e modelos de implantação de acordo com seus requisitos (GHAHRAMANI; ZHOU; HON, 2017).

A computação em nuvem apresenta um novo modelo para entrega de serviços de TI e normalmente envolve acesso de autoatendimento sob demanda e sobre uma rede,

que é dinamicamente escalável e elástica, utilizando conjuntos de recursos frequentemente virtualizados. Por meio desses recursos, a computação em nuvem tem o potencial de melhorar a maneira como as empresas de TI operam, oferecendo inicialização rápida, flexibilidade, escalabilidade e eficiência de custos.

Por meio da computação em nuvem, os departamentos de TI economizam no desenvolvimento de aplicativos, implantações, segurança, tempo e custos de manutenção, enquanto se beneficiam de economias de escala. "Tornar-se verde" e economizar custos são um ponto de foco fundamental para as organizações (CARROLL; MERWE; KOTZE, 2011).

1.2.5 Acordo de Nível de Serviço (SLA)

Um SLA (*service level agreement*) é um acordo entre fornecedor e cliente sobre métricas mensuráveis como tempo de atividade, reatividade, e responsabilidades. (ATLASSIAN, 2020) Um SLA normalmente envolve uma garantia a alguém que utilize o seu serviço de que a sua disponibilidade deve atingir um certo nível durante um período determinado, e se não for cumprido, então será infringido algum tipo de penalidade. (JUDKOWITZ MARK CARTERM, 2018)

Os SLAs são extremamente difíceis de medir, de relatar e de reunir. Esses acordos - geralmente escritos por pessoas que não estão relacionadas à área de técnologia - muitas vezes fazem promessas que são difíceis de medir para as equipes, nem sempre se alinham com as prioridades do negócio que estão em constante evolução, além de não levar em consideração as nuances do projeto.(ATLASSIAN, 2020)

1.2.6 Objetivo de Nível de Serviço (SLO)

Um SLO (*service level objective*) é um acordo dentro de um SLA sobre uma métrica específica como o tempo de disponibilidade ou tempo de resposta. Logo, se o SLA é o acordo formal entre o prestador de serviços e o seu cliente, os SLOs são os objetivos individuais que serão feitos a esse cliente. Os SLOs são o que estabelecem as expectativas do cliente e dizem às equipes de TI e DevOps quais os objetivos que precisam ser atingindos e aliados a quais se desejam medir.(ATLASSIAN, 2020)

1.2.7 Indicador de Nível de Serviço (SLI)

Um SLI (indicador de nível de serviço) mede a compatibilidade com um SLO. Assim, por exemplo, caso o SLA especifique que os sistemas estarão disponíveis 99,95% do tempo, o SLO definido provavelmente é 99,95% do tempo de funcionamento e o SLI é a medida real do tempo de funcionamento. Para se manter em conformidade com o SLA, o SLI terá de cumprir ou exceder as determinações feitas nesse documento.

1.3 Software Legado

A definição habiltual de software legado é um sistema que possui dependências (linguagens bibliotecas, frameworks) que não são atualizados tipicamente, mas que ainda são essenciais para o negócio da organização (SOMMERVILLE, 2016). Porém, também pode ser compreendido como uma aplicação que não é mais atualizada ou manutenida pelos desenvolvedores. Da mesma maneira, um software pode se tornar legado caso o desenvolvimento subitamente seja interrompido ou se é comprado por outra empresa e é tomado a decisão de descontinuá-lo (CHIMA, 2016).

Como consequência de um software ser desenvolvido e lançado menos ativamente, torna-se mais difícil, e sujeito a erros, atualizar o código e executar um novo *deploy* nas raras ocasiões em que é necessário fazer isso. Os passos necessários para configurar um ambiente de desenvolvimento, testar o software, executá-lo localmente, empacotá-lo em uma biblioteca ou executável e implantá-lo em um ambiente de produção podem se perder com o tempo (BIRCHALL, 2016).

Os sistemas legados também podem ser um fator de risco, podendo representar um ameaça para a segurança, pois são desatualizados e, por este fato, podem não receber mais atualizações e correções feitas para possíveis vulnerabilidades. A incompatibilidade com novas tecnologias também é um fator de atenção, por reduzir a flexibilidade e a escalabilidade do software, limitando, também, as integrações com novas funcionalidades de bibliotecas externas que poderiam vir a ser feitas (MCCARTY, 2020).

Entretanto, mesmo levando em consideração os riscos elucidados acima, muitas empresas ainda optam por manter o *software* legado como engrenagem primordial do negócio. O principal ponto a ser ressaltado é que *softwares* geralmente duram anos mesmo que defazados, porém a tecnologia evolui, sendo esse o motivo dos sistemas frequentemente se tornarem obsoletos antes das empresas estarem preparadas para substituí-los. Além disso, existem outros fatores como o sistema ser parte de uma operação crítica nas regras de negócio da empresa, falta de recursos como tempo, dinheiro e pessoal, ou simplesmente a modernização do sistema não interessa a empresa (STACKSCALE, 2021).

1.4 Modernização de software

A modernização de aplicações é a prática de atualizar softwares mais antigos para novas abordagens de computação, incluindo novas linguagens, estruturas e plataformas de infra-estrutura. Essa prática também é comumente chamada de modernização de legados ou modernização de aplicações legadas. É o equivalente a renovação de uma casa antiga para tirar proveito de melhorias na eficiência, segurança, integridade estrutural, porém no contexto de software. Ao invés de reformar um sistema existente ou substituí-lo por

completo, a modernização do legado prolonga a vida útil das aplicações ao mesmo tempo que tira proveito das inovações técnicas (VMWARE, 2022).

Grande parte da discussão em torno da modernização de aplicações hoje em dia está focada em aplicações monolíticas, instaladas e operadas localmente - tipicamente atualizadas e mantidas utilizando processos de desenvolvimento em cascata - e como essas aplicações podem ser convertidas para a arquitetura de serviços de nuvem e padrões de lançamento. As aplicações monolíticas têm duas características que tornam desejável modernizá-las: são difíceis de atualizar, e são difíceis e caras de escalar (IBM, 2019).

A modernização das aplicações permite que uma organização proteja seus investimentos e atualize seu portfólio de software para tirar proveito da infra-estrutura contemporânea, ferramentas, linguagens e outros avanços tecnológicos. Uma estratégia robusta de modernização de aplicações pode reduzir os recursos necessários para executar uma aplicação, aumentar a frequência e a confiabilidade das implantações e melhorar o tempo de atividade e a resiliência, entre outros benefícios. Como resultado, um plano de modernização de aplicações é uma característica comum da estratégia geral de transformação digital de uma empresa (VMWARE, 2022).

1.5 Técnicas de modernização de software

O código legado pode ser uma das experiências mais insatisfatórias e de alto desgaste para os desenvolvedores. Embora haja sempre muita cautela envolvida em prolongar e manter bases de código legadas, tais melhorias continuam a provar ser muito necessárias, apesar de que seja preciso muito esforço e investimento para continuar a manter tal sistema (THOUGHTWORKS, 2017).

1.5.1 Legacy in the box

Para ajudar a reduzir o desgaste, os desenvolvedores utilizam imagens virtualizadas ou imagens de contêineres com contêineres Docker para criar imagens imutáveis de sistemas legados e as suas configurações. Essa técnica, chamada *"Legacy in the box"*, contém o código legado em uma "caixa"para que os desenvolvedores possam operar localmente e remover a necessidade de reconstruir, reconfigurar ou compartilhar ambientes. Em um cenário ideal, as equipes que possuem sistemas legados geram as imagens legadas em "caixas"correspondentes através de suas *pipelines* de *build*, e os desenvolvedores podem então executar e orquestrar estas imagens em *sandbox* alocadas de forma mais confiável (THOUGHTWORKS, 2017).

Adotar a prática do *"Legacy in the box"* não se trata apenas de empacotar o código legado em um contêiner e enviá-lo. Ele também apresenta algumas outras práticas

relacionadas ao DevOps, tais como a adoção de Integração Contínua e Entrega Contínua (CI/CD) no projeto fluxo de trabalho - descrito anteriormente (ALVES; ROCHA, 2022).

Por exemplo, quando se trata de adotar a integração contínua no projeto legado, uma prática central consiste em todos os desenvolvedores fazerem mudanças diariamente diretamente na *branch* principal (ALVES; ROCHA, 2022). Quando uma equipe faz mudanças em incrementos menores e as integra na *branch* principal, mudanças menores, enviadas para a produção rapidamente, são muito mais fáceis de depurar quando algo quebra (MEYER, 2014).

Além do controle de versões, um servidor de integração contínua é uma das mais ferramentas importantes para uma equipe de desenvolvimento. Um servidor de integração contínua é imparcial, suas tarefas se resumem a dizer à equipe de desenvolvedores se as mudanças mais recentes ainda estão de acordo com os estágios que são configurados para executar (MEYER, 2014).

A última etapa de uma *build* automatizada, que tem como objetivo a disponibilização da aplicação para o usuário final, é a implantação na produção, o que requer um processo de implantação automatizado que todo desenvolvedor deve ser capaz de executar, assim como o servidor de integração contínua. Com uma *build* automatizada, todos podem implantar à homologação ou produção, a qualquer momento (MEYER, 2014).

1.5.2 Lift and Shift

"Lift and shift", é o processo de migração de uma cópia exata de uma aplicação ou *codebase* (e seu armazenamento de dados e SO) de um ambiente de hospedagem para outro, normalmente de um ambiente local para uma nuvem pública ou privada.

Por não envolver nenhuma mudança na arquitetura da aplicação e pouca ou nenhuma mudança no código, a estratégia de *"Lift and shift"* permite uma migração mais rápida, menos trabalhosa e, inicialmente, menos onerosa em comparação com outros processos. Atualmente, o *"Lift and shift"* é considerado principalmente como uma opção para a migração de *codebases* que estão prontos para a nuvem em algum grau (por exemplo, aplicações em containers, aplicações construídas em arquitetura de microserviços) ou como um primeiro passo no processo de transformação de arquitetura de uma aplicação monolítica para a nuvem.

As principais vantagens para utilizar essa abordagem giram em torno de ser uma migração rápida, econômica e minimamente disruptiva, por permitir migrar rapidamente, sem dedicar uma grande equipe à tarefa. A aplicação em execução pode permanecer no mesmo ambiente durante a migração para que não haja interrupção do serviço, e a experiência da aplicação permaneça idêntica aos usuários. Além disso, a escalabilidade sob demanda, fato inerente ao migrar para serviços de nuvem, e o potencial para um

ganho de performance significativo, por prover a oportunidade de executar aplicações em hardware atualizado e com melhor desempenho, também são outros benefícios dessa técnica (EDUCATION, 2019).

1.5.3 Outras técnicas

Além das técnicas citadas, que se encaixam em um contexto mais relacionado a *pipeline* e *runtime/deployment*, existem diversas outras abordagens que abrangem outros pontos cruciais de modernização de um software legado e, por mais que aplicadas em cenários distintos, as técnicas de modernização de software se baseiam em alguns principais pilares, como por exemplo:

- ***Rehosting***: também chamada *replatforming*, envolve a migração de componentes de uma aplicação para hardware moderno ou nuvem. O *rehosting* pode ou não exigir mudanças substanciais de código de acordo com a compatibilidade entre o aplicativo e a plataforma que irá hospedá-lo (DILMEGANI, 2021).

- ***Rearchitecting***: envolve a alteração e otimização do código existente para melhorar o desempenho de um sistema legado para permitir novas capacidades. *Rearchitecting* tem como objetivo atualizar a aplicação atual ao invés de construir uma nova aplicação (DILMEGANI, 2021).

- ***Rebuilding***: envolve reescrever o código dos componentes do aplicativo a partir do zero, preservando, ao mesmo tempo, seu escopo (DILMEGANI, 2021). Requere um esforço adicional, mas pode ser essencial se as aplicações existentes tiverem funcionalidades ou vida útil limitada (AZURE, 2022).

- ***Replacing***: Se um aplicativo não atenderá às necessidades comerciais atuais ou futuras, mesmo após o *rebuilding*, poderá ser necessário substituí-lo por uma solução pronta. Esta abordagem pode ser mais rápida do que a reconstrução e liberar valiosos recursos de desenvolvimento. Mas a substituição de aplicativos pode representar desafios, incluindo interrupções nos processos de negócios e limitações para futuras iniciativas de modernização (AZURE, 2022).

No contexto dessas técnicas estão envolvidas, ou tem como pré-requisito, diversas práticas da engenharia de software moderna, como por exemplo para a aplicação do *Rearchitecting*, serão necessários testes de integração, para assegurar que as mudanças aplicadas não impactam no funcionamento dos diversos módulos do software, testes *end-to-end*, para garantir que o comportamento da aplicação continua o mesmo em diversos fluxos de uso, entre outros.

Para técnicas mais agressivas aos sistemas legados, como o *Rebuilding*, conceitos de toda concepção de um software podem ser aplicados, desde a elicitação de requisitos, com a finalidade de clarificar o escopo do módulo a ser reescrito, até utilização de padrões de projeto para implementação do novo código, para ganho de performance e melhorias na manutenibilidade. Além disso, a construção de *pipelines* de integração contínua e containerização do software podem ser adicionados de forma mais simples, visto que o módulo poderá ser reescrito orientado a utilização dessas ferramentas.

2 Proposta

2.1 Problema

Este trabalho tem como objetivo aplicar técnicas de modernização de software, com ênfase em integração contínua, deployment contínuo e infraestrutura como código em uma aplicação legado. Com isso, visamos trazer os benefícios e desafios de tais implantações. Ao final do trabalho, as seguintes questões devem ser solucionadas:

1. Qual a complexidade envolvida ao se implantar as técnicas DevOps, como pipeline de Integração, Deployment contínuos e Infraestrutura como código em um software legado?
2. Quais foram os benefícios de se utilizar DevOps em um software legado em produção?
3. Quais foram as vantagens e desafios encontrados em realocar a infraestrutura utilizando IaC?

2.2 Metodologia

2.2.1 Revisão Bibliográfica

Inicialmente, foi realizada a revisão bibliográfica ao investigar os principais trabalhos realizados em relação à cultura DevOps e como as técnicas de Integração, Entrega e Implantação contínuas são implementadas ao prol da melhoria da entrega e cultura DevOps dentro de uma equipe.

Após a leitura e entendimento dos principais pilares do DevOps, iniciou-se o estudo voltado a Engenharia de Confiabilidade de Sites, que engloba os items de Runtime, IaC, Container e Serviços de Nuvem.

Por fim, foi realizado o estudo das técnicas de modernização de software, que definem métodos e estratégias de como a aplicação dos conceitos da cultura DevOps e Engenharia de Confiabilidade de Sites podem ser aplicados com o intuito de modernizar um software legado.

2.2.2 Escolha das técnicas de modernização

Após a revisão bibliográfica, com os conhecimentos acerca das técnicas de modernização de software em mente, selecionamos quais técnicas seriam utilzadas para modernizar

um software do ponto de vista de DevOps. Este item é explicado em tópicos futuros.

2.2.3 Escolha do projeto

Com as técnicas de modernização definidas, deu-se inicio à busca por projetos que poderiam se beneficiar da modernização e também que cumpriam certos critérios definidos, estes critérios são discutidos na seção 2.2.7.

2.2.4 Escolha das ferramentas de automação

Com a crescente adoção de práticas de Integração Contínua por parte das comunidades de *software*, surgiram diversas ferramentas que auxiliam no processo de execução dos scripts de *build* de forma automática, integrações com plataformas de hospedagem de código-fonte e muitas outras (ELAZHARY et al., 2021). De acordo com (GOLZADEH; DECAN; MENS, 2021), algumas das ferramentas mais utilizadas por projetos de código aberto, são:

			repositories			usages	
CI	**URL**	**first observed on**	**#**	**%**	**cum. %**	**%**	**cum. %**
Travis	http://travis-ci.org	Jun 10, 2011	53,401	58.2%	58.2%	44.9%	44.9%
GHA	http://github.com/features/actions	Jan 23, 2019	46,416	50.6%	90.9%	39.0%	83.9%
CircleCI	http://circleci.com	Jan 15, 2014	11,431	12.4%	98.1%	9.6%	93.5%
AppVeyor	http://www.appveyor.com	Apr 04, 2014	3,553	3.9%	98.3%	3.0%	96.5%
Azure	http://azure.microsoft.com	Sep 11, 2018	1,045	1.1%	98.7%	0.9%	97.3%
GitLab CI	http://docs.gitlab.com/ee/ci	Sep 02, 2015	1,018	1.1%	99.1%	0.9%	98.2%
Jenkins	http://www.jenkins.io	Mar 30, 2016	1,008	1.1%	99.6%	0.8%	99.0%
Others	N/A	Oct 23, 2013	1,138	1.2%	100.0%	1.0%	100.0%

Figura 2 – Uso de serviços de CI - (GOLZADEH; DECAN; MENS, 2021)

O processo de escolha da ferramenta ideal para a implementação dos pipelines da aplicação, foi executado em duas etapas:

1. Colher o conjunto das principais ferramentas utilizadas pela comunidade;
2. Selecionar aquela que traria mais facilidade na manutenção e monitoramento da execução.

Como o projeto selecionado se encontra hospedado no github, a ferramenta que melhor se encaixou nos critérios apresentados, portanto, foi o Github Actions (GHA).

2.2.5 Implementação das técnicas

Tendo em mente o propósito de modernizar uma aplicação do ponto de vista DevOps, como já discutido anteriormente e também explicitado no backlogs, as ténicas que serão implementadas ao projeto, com o propósito de modernizá-lo, serão:

1. Containerização do projeto;
2. Construção dos pipelines de CI/CD;
3. Elaboração dos scripts de IaC;
4. Aplicação do Lift and Shift da infraestrutura.

2.2.6 Análise de métricas

Para medir a o impacto da modernização no software definido, algumas métricas serão colhidas durante o processo, com o objetivo de analisar a melhoria e impacto(s) que a modernização trará ao software e à equipe de desenvolvimento. Tais métricas, inicialmente serão:

- Percepção de esforço e eficácia da execução local do ambiente;
- Percepção de esforço, eficácia e confiabilidade para o versionamento do sistema;
- Resiliência e escalabilidade do sistema.

2.2.7 O projeto legado

Com a finalização da revisão bibliográfica, foi iniciada a busca por repositórios com códigos legado, do ponto de vista de DevOps e SRE. Os critérios utilizados para definir um software legado do ponto de visto de DevOps foram:

- Containerização do software: Busca por softwares que não contassem com ambiente containerizado. Para que seja possível aplicar a técnica de Legacy in The Box (Seção 1.5.1).
- Pipeline de Integração contínua: Busca por software que não tivessem um pipeline de CI implementado.
- Pipeline de Implantação Contínua : Busca por software que não tivessem um pipeline de CD implementado.
- Acesso à infraestrutura: Busca por software em que a infraestrutura pudesse ser acessada. Com isso, será possível aplicar as técnicas de Engenharia de Confiabilidade de software.

Estes foram os itens utilizados para avaliar repositórios legados do ponto de vista de DevOps. Porém, outras características podem definir um software legado, pois, de acordo com Sommervile, a definição desse termo não se limita a isso, também pode ser visto como

um sistema que é desenvolvido em linguagens e tecnologias que não são mais utilizadas para desenvolver novos sistemas e, tipicamente, ainda são essencias para o negócio da organização do ponto de vista de código. Algumas características de software legado do ponto de vista de código podem ser vistas na tabela 1.

3 Resultados

3.1 Caracterização do software

Tendo em vista a definição de software legado do ponto de visto de DevOps, como citado anteriormente, o software escolhido para aplicar a modernização de software foi o Prêmio Profissionais da Música (PPM), disponível em ppm.art.br. Este software, trata-se de uma aplicação web, que funciona como sistema de votação para o evento de mesmo nome.

O software foi desenvolvido a partir de uma arquitetura cliente-servidor, com o cliente sendo desenvolvido em Javascript, sem o uso de nenhum framework moderno. Já aplicação servidor, também em Javascript, foi desenvolvida utilizando a biblioteca NodeJS, que simula um runtime Javascript no servidor.

A escolha deste sistema se deu pela alta pontuação de acordo com os critérios de avaliação citados acima. Tal sistema, é utilizado diariamente, com uma carga maior nos momentos de votação, obtendo milhares de requisições por dia neste período. Este é um ponto importante do ponto de vista de SRE, para trazer uma solução de infraestrutura que escale as instâncias de runtime automaticamente de acordo com a carga de uso do sistema.

O sistema conta com uma equipe de desenvolvimento de três pessoas, que fazem evoluções e manutenções recorrentes. Essa mesma equipe também exerce o suporte de tecnologia em momentos de pico de uso, escalando manualmente os servidores. O principal desafio no trabalho desta equipe, se trata justamente da observabilidade do software nos momentos de pico e momentos de votação, onde podem haver gargalos por conta dos múltiplos acessos.

Além da observabilidade ser bastante desafiadora, a equipe tem de tomar decisões baseadas em dados colhidos durante esse processo, seja escalando os servidores de backend, servidores de banco de dados ou outros serviços e recursos utilizados pelo sistema como um todo. A modernização da infraestrutura com *cloud services* utilizando as técnicas de SRE irá automatizar essas ações com base no uso e na carga dos serviços, além de trazer métricas de observabilidade mais claras e diretas para o time, também automatizadas, sem a necessidade de leitura de logs por conta da equipe.

3.2 Escolha das técnicas

Para resolver os pontos citados acima, será aplicada a técnica de Legacy in the box (Seção 1.5.1), em seguida, serão implementadas Pipelines de Integração contínua (CI) e Entrega Contínua (CD) (Seção 1.1.2), paralelamente ao o Lift and Shift (Seção 1.4.2) etapa concluída na infraestrutura do Software, utilizando IaC.

A escolha das técnicas utilizadas se baseou em cima do contexto do projeto que será aplicado, essencialmente pela ausência de conteinerização do projeto, inexistência de pipelines de CI e CD e hospedagem em uma plataforma a qual não oferece escalonamento e a liberdade do mantenedor do projeto quanto aos recursos computacionais utilizados. Circunstâncias resolvidas inteiramente pelos métodos selecionados.

A sequência das técnicas é, primordialmente, em detrimento de serem complementares e possuirem a capacidade de tanger diferentes estágios do ciclo de vida e desenvolvimento de um software. O primeiro método a ser aplicado é o Legacy in the box , devido à necessidade de um ambiente isolado, ou em outras palavras, conteinerizado, para que as mudanças efetuadas pela equipe no projeto se assemelhem ao ambiente utilizado para execução dos testes realizados nas pipelines de CI/CD, etapa que assegura qualidade e disponibilidade do código implementado. O Lift and Shift é aplicado posteriormente ao Legacy in the Box pelo fato de ser executada uma mudança na plataforma a qual hospeda a ferramenta em questão, logo, é necessário que haja um serviço em pleno funcionamento para que a modificação seja efetuada de maneira segura. Além disso, a etapa a qual constitui o desenvolvimento dos scripts de IaC do Lift and Shift pode ser realizada paralelamente a implementação das pipelines de integração e deploy contínuo, sem gerar interferência em sua etapa anterior.

Tais técnicas serão implementadas em um software legado, que no entanto é utilizado por milhares de usuários.

3.3 Arquitetura da solução

Um importante passo na solução de uma modernização de um software é o planejamento arquitetural, dessa maneira, realizamos o estudo da arquitetura legado e com isso planejamos a nova, já integrando todos os conceitos e técnicas de modernização do ponto de vista de DevOps.

De acordo com a Figura B, temos um diagrama arquitetural da solução legado, que é utilizada diariamente. Nesta solução, o cliente e o servidor são hospedados na plataforma Heroku.

O Heroku é uma plataforma de nuvem como serviço que suporta várias linguagens de programação. Como uma das primeiras plataformas em nuvem, o Heroku está em

desenvolvimento desde junho de 2007, quando suportava apenas a linguagem de programação Ruby, mas agora suporta Java, Node.js, Scala, Clojure, Python, PHP e Go.

Este provedor tem como base uma funcionalidade chamada Plataforma como Serviço (PaaS), que visa facilitar a implantação de novas aplicações de maneira rápida e prática, sem se preocupar com a saúde do servidor, atualizações de sistema operacional, nem instalação de dependências. O Heroku provê toda a infraestrutura de forma automática para facilitar a vida de seus cliente. Essa plataforma é uma boa solução para pequenos projetos e projetos iniciais, mas a partir do momento que sua aplicação precisa crescer e ter mais funcionalidades que a permitam funcionar bem e de uma maneira saudável, ele deixa de se tornar uma boa solução, pois o uso de servidores profissionais no Heroku é extremamente custoso, e ele não fornece funcionalidades como orquestração de containers, configurações de rede e rede de distribuição de conteúdos, por exemplo.

Na parte de implantação de novas versões da aplicação, foi configurado um webhook, entre o repositório do github e o Heroku, onde toda vez que há a subida de quaisquer modificações no código fonte em uma branch específica, o Heroku iniciará uma nova implantação destas modificações nos servidores.

A configuração atual conta com dois ambientes Heroku, um para o servidor (Back-End) e outro para o cliente (Front-End). Vemos aqui um desperdício de recurso, já que o frontend não necessita de nenhum processamento no lado do servidor, já que se trata de uma aplicação estática, composta por HTML, CSS e JavaScript.

Do lado do banco de dados, tal recurso é gerenciado por um provedor chamado KingHost. Que provê e gerencia um banco de dados MySQL, que é utilizado diretamente pelo servidor(Back-End).

De acordo com a figura C, onde podemos observar nossa proposta de solução, é possível visualizar onde cada método de modernização foi aplicado. Iniciamos com a aplicação Back-End, onde aplicamos o Legacy in the Box, utilizando uma imagem Docker para declarar o ambiente necessária a essa aplicação e utilizar essa imagem como base para executar a aplicação em forma de containers.

Outra modernização aplicada, foi na parte IaC, onde implementamos toda a infraestrutura de nuvem AWS que a aplicação irá utilizar com Terraform, um framework de IaC que utiliza como base a linguagem de configuração HCL.

Com os pipelines de integração, temos a aplicação de alguns conceitos utilizados em DevOps, como Implantação Contínua (CI) e Entrega Contínua (CD). Além da utilização da imagem Docker gerada para realização dos passos do pipeline.

Por fim, temos a parte de Lift and Shift, que se propõem a realizar uma modernização do software do lado de infraestrutura, criando uma infraestrutura que suporte a aplicação atual (lift) e realizando a migração dessa aplicação para a nova infraestrutura

(shift).

Tendo em vista os pontos negativos na utilização do Heroku e também como discutido no tópico de serviços de nuvem, o serviço de nuvem selecionado para realizar essa migração, foi a AWS.

Arquitetura anterior à solução disponível no apêndice B.

Arquitetura da solução disponível no apêndice C.

3.4 Métricas

Durante a implementação do projeto foram colhidas métricas que determinam o sucesso e a efetividade da solução realizada. Para captação desses indicadores utilizamos da observação de protocolos que eram seguidos anteriormente pela equipe de desenvolvimento, tirando proveito das técnicas descritas anteriormente para a execução.

No dado contexto do estudo de caso realizado a percepção de esforço é dada como a quantidade de comandos que o desenvolvedor precisa executar para o determinado contexto, aliado com a quantidade de refências e diferentes fontes de adquirir instruções necessárias para a execução da tarefa proposta.

Além disso, a percepção de eficácia é definida como a efetividade em que a tarefa é executada pelo desenvolvedor, ou seja, não necessita retrabalho ou passos extras que adicionem complexidade à tarefa definida.

Esses conceitos foram definidos a fim de trazer as métricas coletadas para um contexto mais subjetivo, em que pudéssemos nos desvenciliar das métricas númericas, que não agregariam de forma coerente em primeiro momento desse estudo de caso.

3.4.1 Percepção de esforço e eficácia da execução local do ambiente

Anteriormente à conteinerização da aplicação (Legacy in the Box), os desenvolvedores do projeto necessitavam seguir uma série de passos para configuração do ambiente, o que pode trazer um desgaste desnecessário, além de conflitos de versionamento do *framework* instalado, acarretando em erros ao implantar o sistema no ambiente de produção por conter trechos de código desatualizados ou com o comportamento diferente da versão utilizada pelo desenvolvedor.

Os passos para instalação e execução local da aplicação Back-End, levando em consideração que serão feitos em um ambiente UNIX baseado em Debian, são:

```
# Instalar o Node.js
$ curl -fsSL https://deb.nodesource.com/setup_19.x | sudo -E bash -
$ sudo apt-get install -y nodejs
```

```
# Instalar o gerenciador de pacotes Yarn
$ npm install --global yarn

# Instalar pacotes da aplicação
$ yarn install

# Executar a aplicação
$ yarn run
```

Já para instalação e execução local utilizando um contêiner Docker, é necessário apenas possuir o Docker instalado[1] localmente. Além disso, o contêiner gerencia a versão da ferramenta de desenvolvimento, padronizando esse aspecto entre os desenvolvedores.

Os passos para instalação e execução do contêiner da aplicação Back-End, levando em consideração que serão feitos em um ambiente UNIX baseado em Debian, são:

```
# Instala o Docker localmente
$ sudo apt-get update && \
  sudo apt-get install docker-ce docker-ce-cli \
  containerd.io docker-compose-plugin

# Executa a aplicação
$ docker-compose up --build
```

Logo é possível concluir que a implementação de um dos princípios da técnica Legacy in the box 1.5.1, contribuiu para a confiabilidade do software, ao padronizar a versão de execução local da aplicação com a versão implantada em produção, além de reduzir o desgaste dos contribuidores do projeto.

3.4.2 Percepção de esforço, eficácia e confiabilidade para o versionamento do sistema

A confiabilidade do software é a probabilidade de funcionamento sem falhas de um programa de computador durante um período determinado, num ambiente específico. A confiabilidade é uma visão da qualidade do software orientada para o cliente. Está relacionada com o funcionamento e não com a concepção do programa, e por isso é dinâmica e não estática. (IANNINO; MUSA, 1990)

[1] Instalação Docker

Previamente à implementação da solução proposta para este projeto, o sistema alvo da modernização tinha como provedor de infraestrutura o Heroku, conforme mencionado em 3.3. Aliado as limitações da plataforma, a maneira com que o sistema era implantado em produção se limitava a utilização de um *webhook* para monitoramento do código disponibilizado na *branch* padrão do projeto e, a partir da nova versão detectada, o código era empacotado e implantado em produção.

Esse tipo de configuração reduz a eficácia relacionada ao versionamento do sistema, pois além de não manter rastros de versões estáveis do software, também agrega um esforço adicional para o *rollback*, reversão de alterações feitas no sistema para o estado anterior da aplicação.

Tomando como exemplo, caso seja necessário recorrer ao *rollback* do sistema para uma versão anterior por um erro detectado em produção no serviço de Back-End - parte do software que não pode ser acessada por um usuário - sem alteração no banco de dados, os seguintes passos seriam necessários:

1. Realização de exclusão do último *commit* incorporado a branch padrão (*git reset –hard HEAD^*);
2. Forçar a reescrita da ordem de *commits* da *branch* padrão (*git push –force origin master*).

Os comandos da ferramenta *git* descritos anteriormente são comandos agressivos, pois os mesmos editam a ordem das alterações realizadas e enviam para o repositório remoto de forma obrigatória, ignorando as possíveis restrições impostas, a árvore de *commits* local. Tal alteração pode excluir permanentemente mudanças chaves realizadas no sistema e, por estar automaticamente ligado a um webhook de implantação em produção, reduzir a confiabilidade do software entregue ao usuário. Os mesmos passos descritos anteriormente também se aplicam para o *rollback* do serviço de Front-End, camada de interface com o usuário.

Agora, caso seja necessário a realização de um *rollback* do sistema para uma versão anterior no serviço de Back-End que envolvam alterações no banco de dados, serão necessários os seguintes passos:

1. Realização de exclusão do último *commit* incorporado a branch padrão (*git reset —hard HEAD^*);
2. Forçar a reescrita da ordem de *commits* da *branch* padrão (*git push –force origin master*);
3. Acesso ao console de produção da plataforma que hospeada o banco de dados;

4. Reversão manual da alteração feita via comandos SQL.

Neste caso, as mudanças seriam ainda mais agressivas pois seria necessário a alteração do banco de dados de produção via linha de comando, estando ainda mais sujeito a falhas e, por consequência, acarretando na redução da confiabilidade do software entregue ao usuário.

Com a implementação da solução de modernização do software, o risco relacionado ao *rollback* de versões da aplicação é praticamente nulo. Isso se deve ao *pipeline* de integração contínua construído, que em um de seus passos, gera uma versão da imagem do sistema para envio ao Amazon ECR. Logo, para realização da reversão das alterações realizadas no ambiente de produção, serão necessários os seguintes passos:

1. Acesso a interface de admin da Amazon Web Sevices;
2. Deploy da versão selecionada, gerada automaticamente pela integração contínua, no serviço ECR.

Ao adicionar a complexidade da reversão de alterações também no banco de dados, o protocolo a ser seguido é:

1. Acesso a interface de admin da Amazon Web Sevices;
2. Deploy da versão selecionada, gerada automaticamente pela integração contínua, no serviço ECR;
3. Implantação do último snapshot - uma visão do banco de dados em um determinado momento - prévio à mudança desejada.

Levando em consideração os procedimentos levantados anteriormente, a modernização da aplicação levou ao aumento da eficácia do versionamento do sistema e esforço para a realização do mesmo. Ademais, por consequência, a melhoria na confiabilidade do software também foi um fator de melhoria.

3.4.3 Resiliência e escalabilidade do sistema

É levado em consideração os conceitos de resiliência de software, como a capacidade de uma solução de absorver o impacto de um problema em uma ou mais partes de um sistema, continuando ao mesmo tempo a fornecer um nível de serviço aceitável ao negócio.(NASSAR, 2020) E escalabilidade, como a medida da capacidade de um sistema para aumentar ou diminuir o desempenho e o custo em resposta a mudanças nas demandas da aplicação e de processamento do sistema.(CYBERLINKASP, 2021)

A plataforma utilizada para hospedar a infraestrutura do projeto alvo deste estudo de caso tem como base de serviço o que se chama de *Dynos* - containers Linux isolados e virtualizados que são concebidos para executar código com base num comando determinado pelo operador. A aplicação pode ser dimensionado para qualquer número desejado de *dynos*, com base nas necessidades de recursos. As capacidades de gestão de containers do Heroku possibilitam uma forma simplificada de escalar e gerir o número, tamanho e tipo de dynos que a aplicação possa necessitar em qualquer ocasião (HEROKU, 2022) - oferece essa solução de acordo com o plano escolhido pelo operador, ou seja, apenas o escalonamento vertical (adição de maior capacidade de recursos de hardware) de recursos é oferecido.

A limitação da plataforma se dá principalmente pela escolha de configuração feita pela equipe que gere o projeto alvo da modernização, que para o serviço de Front-End foi delegado somente um *Basic Dyno* e, para o serviço de Back-End dois *Standard Professional Dynos*. Esses *Dynos* podem ser visualizados na tabela 2.

Entretanto, na solução implementada, o escalonamento vertical e horizontal é automático, baseado em métricas de uso de CPU e memória coletadas pelo serviço *AWS CloudWatch*. Logo, ao adicionar a implementação feita neste estudo na tabela anterior o panorama seria o retratado na tabela presente na tabela 3.

Levando em consideração esse cenário, tanto a resiliência do software quanto a escalabilidade foram melhoradas pela adequação automática de recursos para o sistema.

4 Conclusão

A modernização de software envolve diversos desafios por conta da necessidade de contato com a equipe atual de mantenedores do projeto, acesso a painel de monitoramento e chaves necessárias para conexão com banco de dados. Fora os aspectos que envolvem modernizar um software que não segue boas práticas também se torna um desafio, pela ausência de padronização no que tange versionamento de software, containerização do projeto, implementação de testes unitários e monitoramento de métricas do sistema.

É possível concluir que a relação entre o esforço necessário para a modernização de uma aplicação e a adoção de boas práticas de software são inversamente proporcionais, pois quanto mais despadronizado um projeto, maior será o esforço necessário para atualizá-lo para as últimas técnologias adotadas pelo mercado, que, muitas vezes, tem como pré-requisito o uso de boas práticas.

Durante a realização do processo de modernização de sofware, ficou claro as nuances entre as diferentes provedoras de serviço em nuvem e como essa escolha pode impactar no rumo em que o sistema irá tomar, no que diz respeito a escalabilidade a um médio/longo prazo. Outrossim, é evidente a necessidade da containerização de softwares que serão distribuídos para usuários finais, essa prática além de facilitar o processo de desenvolvimento da aplicação, contribui na mitigação de riscos relacionados a confiabilidade do software.

Ao final do trabalho, foi possível concluir que um dos passos mais desafiadores do projeto englobava a aquisição de métricas númericas, dado que a infraestrutura do sistema anteriormente não possuía nenhum tipo de monitoramento configurado. Dessa forma, o trabalho contou apenas com métricas subjetivas, mais relacionadas a eficácia, esforço e conceitos do campo de estudo do tema.

Em síntese foi possível observar a diferença que a aplicação de técnicas de modernização de software trazem para um sistema, afetando não só o esforço realizado para determinadas tarefas pelos desenvolvedores quanto o desempenho da aplicação em produção, contando com uma infraestrutura muito mais consistente e que se adapta as necessidades.

4.1 Lições aprendidas

A motivação inicial para o desenvolvimento desse estudo foi o desgaste latente ligado a sistemas legados, que não se adequam à realidade atual das convenções de engenharia de software. Levando em consideração esse contexto, decidimos nos aprofundar quanto a aplicação de práticas de DevOps no âmbito da modernização de software.

Softwares legados são sistemas que possuem dependências (linguagens bibliotecas,

frameworks) que não são atualizados tipicamente, mas que ainda são essenciais para o negócio da organização, e é nesse contexto que focamos nosso estudo, em suma, softwares que ainda são necessários porém que geram desgastes de manutenção e desenvolvimento.

Adotamos para este estudo um viés que engloba, principalmente, a parte de containerização do software e melhorias de infraestrutura, o que trouxe ganhos, no sentido de reduzirmos o escopo da modernização para a parte de "entrega"da aplicação, porém também agregou mais desgastes, pois ao se trabalhar com infraestrutura também adicionamos a váriavel monetária ao contexto, além das várias permissões necessárias para se acessar a infraestrutura legado.

Iniciamos nosso estudo pela conteinerização do sistema, que de certa forma foi bastante simples e direta, o software não possuía muitas dependências e foi desenvolvido em uma linguagem e framework que possuem uma vasta comunidade, então esse fator ajudou muito na agilidade da entrega dessa melhoria. Nos seguintes passos fomos evoluíndo gradualmente em direção a infraestrutura, passando pelas *pipelines* de CI e CD, que foram desenvolvidas com o GitHub Actions, que utilizavam das imagens construídas no passo da conteinerização.

Passando para a etapa que envolve a infraestrutura, o conhecimento prévio das ferramentas e do provedor cloud que foi utilizado foi um ponto muito forte da nossa equipe. Já possuíamos prática em grandes projeto de desenvolvimento de IaC, principalmente direcionado para AWS.

No que tange os desafios enfrentados para a aplicação da técnica Lift and Shift (processo de migração de uma cópia exata de uma aplicação de um ambiente de hospedagem para outro), o principal notado por nós foi a utilização inicial do free-tier das ferramentas disponíveis, o que pode ser evoluído caso o stakeholder do projeto deseje por adotar a modernização proposta nesse projeto.

Para evoluções desse projeto e estudos posteriores que tenham este como base, recomendamos que antes da aplicação das técnicas de modernização de software, agreguem ferramentas de monitoramento no software legado, para que dessa forma o desempenho e a efetividade da solução implementada possua suporte numérico palpável. Além disso, a definição de SLAs, SLOs e SLIs podem agregar a futuras evoluções objetivos mais claros de implementação e juntar a eles a visão dos interessados (que não participam do time de técnologia) no projeto.

Apêndices

APÊNDICE A – Técnicas de modernização

Técnica	Características
Ausência de testes	Software que não possui nenhum tipo de teste automatizado.
Dependências desatualizadas	Software que utiliza dependência que não são mais mantidas, ou versões desatualizadas de dependências.
Tecnologia ultrapassada	Software que foi desenvolvido utilizando uma tecnologia que não é mais mantida e/ou atualizada.

Tabela 1 – *Técnicas de modernização - Autoria própria*

APÊNDICE B – Arquitetura anterior à solução

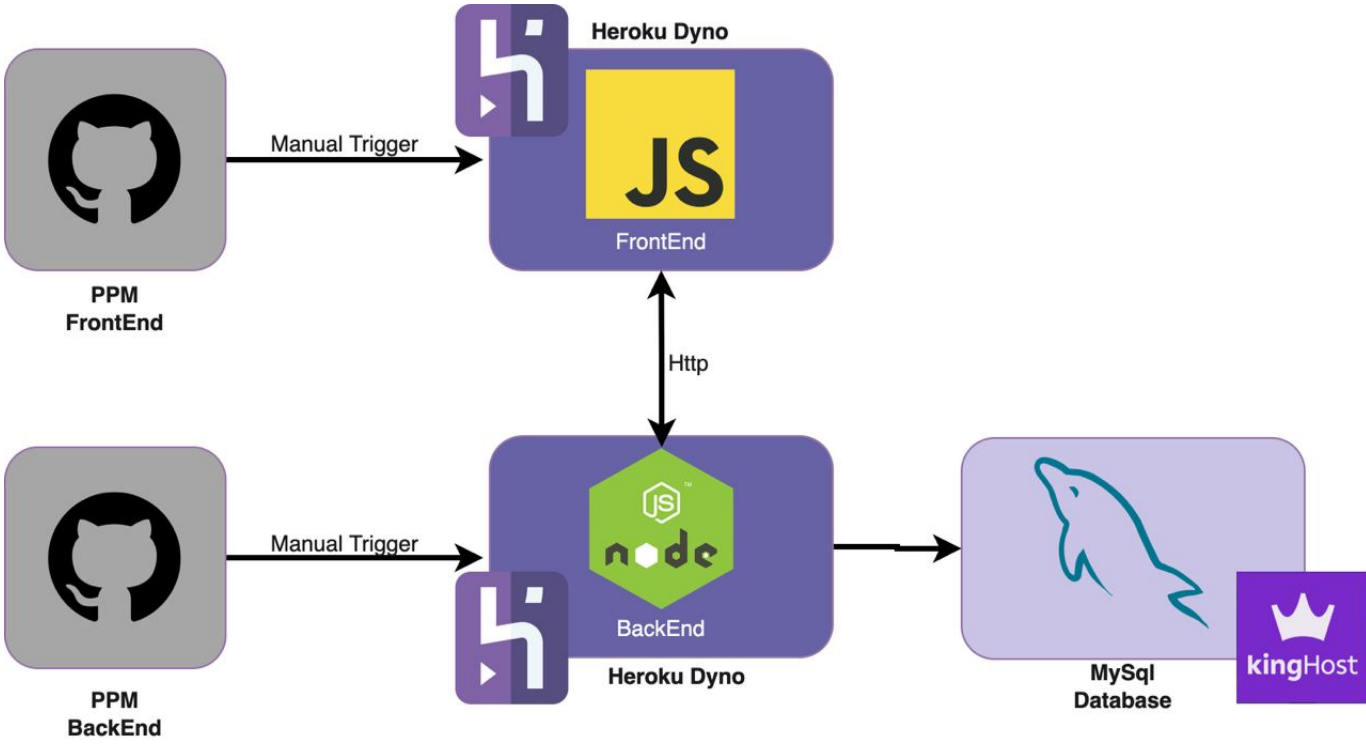

Figura 3 – Arquitetura anterior à solução (Autoria própria)

APÊNDICE C – Arquitetura da solução

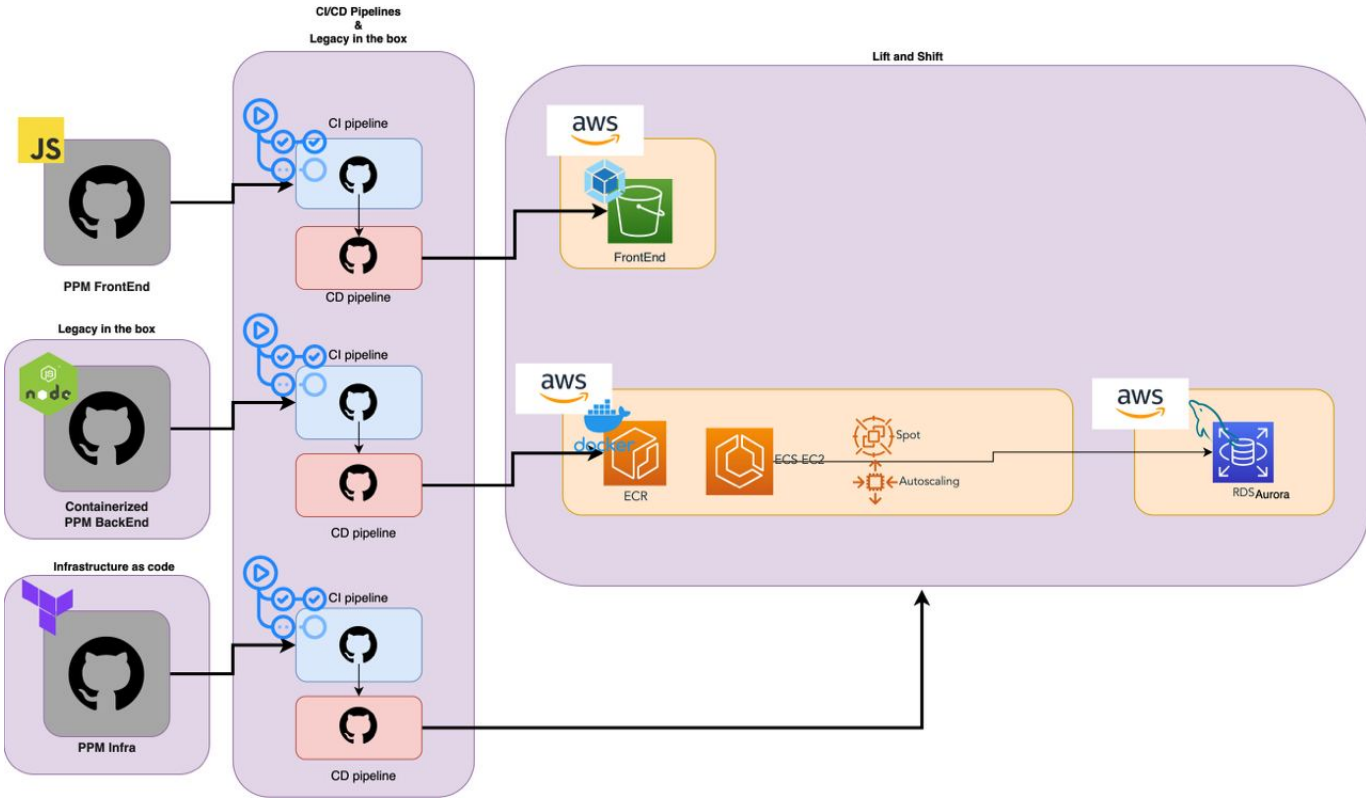

Figura 4 – Arquitetura da solução (Autoria própria)

APÊNDICE D – Recursos por Dyno

Recursos / Solução	**Basic Dynos**	**Standard Professional Dynos**
RAM	512MB	512MB
Número de tipos de processo	10	Ilimitado
Escalonamento horizontal	Não possui	Possui
Escalonamento automático	Não possui	Não possui

Tabela 2 – Recursos por Dyno - Autoria própria

APÊNDICE E – Comparação de soluções

Recursos / Solução	**Basic Dynos**	**Standard Professional Dynos**	**Solução do estudo**
RAM	512MB	512MB	Automaticamente configurada
Número de tipos de processo	10	Ilimitado	Ilimitado
Escalonamento horizontal	Não possui	Possui	Possui de acordo com a demanda
Escalonamento automático	Não possui	Não possui	Possui

Tabela 3 – Comparação de soluções - Autoria própria

Referências

ALVES Álax; ROCHA, C. Assuring the evolvability of legacy systems in devops transformation/adoption: Insights of an experience report. A ser publicado, 2022. Citado na página 29.

ATLASSIAN. *SLA vs. SLO vs. SLI: What's the difference?* 2020. <https://www.atlassian.com/incident-management/kpis/sla-vs-slo-vs-sli#:~:text=An%20SLI%20(service%20level%20indicator,actual%20measurement%20of%20your%20uptime.>, acessado em novembro de 2022. Citado na página 26.

AZURE. *What is application modernization?* 2022. <https://azure.microsoft.com/pt-br/overview/legacy-application-modernization/>, acessado em março de 2022. Citado na página 30.

BIRCHALL, C. *Re-engineering legacy software*. [S.l.]: Simon and Schuster, 2016. Citado 2 vezes nas páginas 19 e 27.

CARROLL, M.; MERWE, A. van der; KOTZE, P. Secure cloud computing: Benefits, risks and controls. In: *2011 Information Security for South Africa*. IEEE, 2011. Disponível em: <https://doi.org/10.1109/issa.2011.6027519>. Citado na página 26.

CHEN, L. Continuous delivery: Huge benefits, but challenges too. *IEEE Software*, Institute of Electrical and Electronics Engineers (IEEE), v. 32, n. 2, mar. 2015. Disponível em: <https://doi.org/10.1109/ms.2015.27>. Citado 2 vezes nas páginas 21 e 22.

CHIMA, R. *Legacy Software: How To Tell If Your Software Needs Replacing*. 2016. <https://www.bbconsult.co.uk/blog/legacy-software>, acessado em fevereiro de 2022. Citado na página 27.

CYBERLINKASP. *What is Software Scalability and Why is it Important?* 2021. <https://www.cyberlinkasp.com/insights/what-is-software-scalability-and-why-is-it-important/#:~:text=Software%20scalability%20is%20the%20ability,scale%20back%20operations%20as%20needed.>, acessado em novembro de 2022. Citado na página 43.

DILMEGANI, C. *What is Legacy App Modernization? Benefits Techniques*. 2021. <https://research.aimultiple.com/application-modernization/>, acessado em março de 2022. Citado na página 30.

EDUCATION, I. C. *Lift and Shift: An Essential Guide*. 2019. <https://www.ibm.com/cloud/learn/lift-and-shift>, acessado em março de 2022. Citado na página 30.

EDUCATION, I. C. *Containers vs. Virtual Machines (VMs): What's the Difference?* 2021. <https://www.ibm.com/cloud/blog/containers-vs-vms>, acessado em Abril de 2022. Citado na página 25.

ELAZHARY, O. et al. Uncovering the benefits and challenges of continuous integration practices. *IEEE Transactions on Software Engineering*, Institute of Electrical and Electronics Engineers (IEEE), p. 1–1, 2021. ISSN 2326-3881. Disponível em: <http://dx.doi.org/10.1109/TSE.2021.3064953>. Citado na página 34.

GHAHRAMANI, M. H.; ZHOU, M.; HON, C. T. Toward cloud computing QoS architecture: analysis of cloud systems and cloud services. *IEEE/CAA Journal of Automatica Sinica*, Institute of Electrical and Electronics Engineers (IEEE), v. 4, n. 1, p. 6–18, jan. 2017. Disponível em: <https://doi.org/10.1109/jas.2017.7510313>. Citado na página 25.

GOLZADEH, M.; DECAN, A.; MENS, T. *On the rise and fall of CI services in GitHub*. Zenodo, 2021. Disponível em: <https://zenodo.org/record/5815352>. Citado 2 vezes nas páginas 11 e 34.

HEROKU. *Heroku Dynos*. 2022. <https://www.heroku.com/dynos>, acessado em novembro de 2022. Citado na página 44.

IANNINO, A.; MUSA, J. D. Software reliability. In: YOVITS, M. C. (Ed.). Elsevier, 1990, (Advances in Computers, v. 30). p. 85–170. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0065245808602995>. Citado na página 41.

IBM. *Application Modernization*. 2019. <https://www.ibm.com/cloud/learn/application-modernization>, acessado em março de 2022. Citado na página 28.

JUDKOWITZ MARK CARTERM, G. C. J. *SRE fundamentals: SLIs, SLAs and SLOs*. 2018. <https://cloud.google.com/blog/products/devops-sre/sre-fundamentals-slis-slas-and-slos>, acessado em novembro de 2022. Citado na página 26.

LEITE, L. et al. A survey of DevOps concepts and challenges. *ACM Computing Surveys*, Association for Computing Machinery (ACM), v. 52, n. 6, p. 1–35, nov. 2020. Disponível em: <https://doi.org/10.1145/3359981>. Citado 4 vezes nas páginas 11, 21, 23 e 24.

LóPEZ-FERNáNDEZ, D. et al. *DevOps Team Structures: Characterization and Implications*. arXiv, 2021. Disponível em: <https://arxiv.org/abs/2101.02361>. Citado na página 23.

MCCARTY, D. *What Are the Biggest Problems with Legacy Software?* 2020. <https://www.gavant.com/library/what-are-the-biggest-problems-with-legacy-software/>, acessado em fevereiro de 2022. Citado 2 vezes nas páginas 19 e 27.

MEYER, M. Continuous integration and its tools. *IEEE Software*, Institute of Electrical and Electronics Engineers (IEEE), v. 31, n. 3, p. 14–16, maio 2014. Disponível em: <https://doi.org/10.1109/ms.2014.58>. Citado na página 29.

NASSAR, I. S. *Software Resilience*. 2020. <https://www.ibm.com/developerworks/websphere/techjournal/1407_col_nasser/1407_col_nasser.html>, acessado em novembro de 2022. Citado na página 43.

PAHL, C. Containerization and the PaaS cloud. *IEEE Cloud Computing*, Institute of Electrical and Electronics Engineers (IEEE), v. 2, n. 3, p. 24–31, maio 2015. Disponível em: <https://doi.org/10.1109/mcc.2015.51>. Citado na página 25.

RAHMAN, A.; MAHDAVI-HEZAVEH, R.; WILLIAMS, L. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, Elsevier BV, v. 108, p. 65–77, abr. 2019. Disponível em: <https://doi.org/10.1016/j.infsof.2018.12.004>. Citado na página 24.

SHAHIN, M.; BABAR, M. A.; ZHU, L. Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, Institute of Electrical and Electronics Engineers (IEEE), v. 5, p. 3909–3943, 2017. Disponível em: <https://doi.org/10.1109/access.2017.2685629>. Citado na página 22.

SOMMERVILLE, I. *Software Engineering*. 10th. ed. USA: Pearson Education, 2016. ISBN 1-292-09613-6, 978-1-292-09613-1. Citado 2 vezes nas páginas 19 e 27.

STACKSCALE. *What is a legacy system?* 2021. <https://www.stackscale.com/blog/legacy-systems/#Why_are_legacy_systems_still_used>, acessado em fevereiro de 2022. Citado na página 27.

THOUGHTWORKS, I. *Legacy in a box*. 2017. <https://www.thoughtworks.com/pt-br/radar/techniques/legacy-in-a-box>, acessado em março de 2022. Citado na página 28.

VMWARE. *What is application modernization?* 2022. <https://www.vmware.com/topics/glossary/content/application-modernization.html>, acessado em março de 2022. Citado na página 28.