



FACULDADE PROFESSOR MIGUEL ÂNGELO DA SILVA SANTOS – FeMASS  
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Pipeline Devops: Um guia Prático de Implementação e Desenvolvimento

POR:  
ARTHUR ESTEPHANO LEMOS DE MOURA

MACAÉ  
2023

FACULDADE PROFESSOR MIGUEL ÂNGELO DA SILVA SANTOS – FeMASS  
CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

Arthur Estephano Lemos de Moura

PIPELINE DEVOPS: UM GUIA PRÁTICO DE IMPLEMENTAÇÃO E  
DESENVOLVIMENTO

Trabalho Final apresentado ao curso de graduação em  
Sistemas de Informação, da Faculdade Professor  
Miguel Ângelo da Silva Santos (FeMASS), para  
obtenção do grau de BACHAREL em Sistemas de  
Informação.

Professor Orientador: Prof. D. Sc. Alan Carvalho Galante

MACAÉ/RJ

2023

ARTHUR ESTEPHANO LEMOS DE MOURA

PIPELINE DEVOPS: UM GUIA PRÁTICO DE IMPLEMENTAÇÃO E  
DESENVOLVIMENTO

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Sistemas de Informação, da Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS), para obtenção do grau de BACHAREL em Sistemas de Informação.

Aprovada em \_\_\_\_ de \_\_\_\_\_ de 20\_\_\_\_

BANCA EXAMINADORA

---

Prof. D. Sc. Alan Carvalho Galante  
Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS)  
1º Examinador

---

Prof. D. Sc. Isac Mendes Lacerda  
Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS)  
2º Examinador

## CÓPIA DA ATA ASSINADA APÓS A BANCA

Este espaço deverá ficar reservado para colocação da Ata, após a defesa.

## **DEDICATÓRIA**

Dedico este trabalho a todos que me apoiaram ao longo dessa jornada, em especial à minha família, aos amigos, ao orientador e à minha querida namorada. Obrigado por fazerem parte desse caminho.

## **AGRADECIMENTO**

À minha família e aos amigos, por me apoiarem tanto na decisão de mudança de curso superior quanto durante este que está em processo. À minha namorada Marcela, por estar comigo durante meu percurso acadêmico e ser minha base. Ao professor Alan Carvalho Galante, por ter sido meu orientador e ter desempenhado essa função com dedicação, amizade e companheirismo. A todos os colegas do curso de S.I. que estudaram comigo desde 2019, por compartilharem momentos inesquecíveis e se tornarem verdadeiros amigos de caminhada.

## EPÍGRAFE

" A tecnologia move o mundo."  
(Steve Jobs)

## RESUMO

Este trabalho constitui um estudo experimental voltado à análise e à implementação de Pipelines DevOps, realizando sua aplicação prática em um projeto Java no âmbito da disciplina de Desenvolvimento de Sistemas II na FeMASS. O cerne do estudo consiste na elaboração de um guia prático destinado à implementação e ao desenvolvimento dessas ferramentas. O objetivo principal é educar, informar os alunos da FeMASS e inspirá-los a compreender e dominar os princípios da Integração Contínua, Entrega Contínua e da filosofia DevOps. A abordagem metodológica escolhida é de natureza qualitativa e teve início com a observação da realidade do mercado de Tecnologia da Informação, com ênfase nas Pipelines DevOps, comparando-a com a oferta de disciplinas na FeMASS. Para a realização deste estudo, foi utilizado um repositório específico com o GitHub Actions, visando à implementação de Pipelines abrangendo testes, compilação, análise de código e *deploy* de uma aplicação Java. Nesse processo, foi utilizado o SonarCloud como ferramenta de qualidade e o Azure Container Registry e Azure Kubernetes Service como ferramentas para *deploy*. A disponibilização deste guia tem como meta primordial servir de referência para os alunos da instituição, capacitando-os a conceber e executar esses processos, o que contribuirá significativamente para a expansão de seus conhecimentos técnicos, preenchendo lacunas não abordadas durante o curso de Sistemas de Informação na FeMASS. Os resultados obtidos promoveram uma melhor compreensão da adoção de boas práticas de programação durante o ciclo de desenvolvimento, alertando para a necessidade de integração de testes ao longo do projeto, bem como para os riscos de segurança envolvidos, facilitando a compreensão e implementação desses processos. Como resultado, este trabalho produziu um guia abrangente, documentando em detalhes o processo de desenvolvimento e implementação das Pipelines, proporcionando uma valiosa fonte de consulta para os alunos da FeMASS.

Palavras-chave: Pipelines. DevOps. Integração Contínua. Entrega Contínua.



## **ABSTRACT**

This work is an experimental study aimed at analyzing and implementing DevOps Pipelines, applying them in practice in a Java project as part of the Systems Development II course at FeMASS. The core of the study consists of drawing up a practical guide for the implementation and development of these tools. The main objective is to educate, inform and inspire FeMASS students to understand and master the principles of Continuous Integration, Continuous Delivery, and the DevOps philosophy. The chosen methodological approach is qualitative in nature and began by observing the reality of the Information Technology market, with an emphasis on DevOps Pipelines, and comparing it with the courses on offer at FeMASS. To carry out this study, a specific repository with GitHub Actions was used to implement Pipelines covering testing, compilation, code analysis and deployment of a Java application. In this process, SonarCloud was used as a quality tool and Azure Container Registry and Azure Kubernetes Service as deployment tools. The primary goal of making this guide available is to serve as a reference for the institution's students, enabling them to design and execute these processes, which will contribute significantly to expanding their technical knowledge, filling in gaps not addressed during the Information Systems course at FeMASS. The results obtained promoted a better understanding of the adoption of good programming practices during the development cycle, alerting them to the need to integrate tests throughout the project, as well as the security risks involved, facilitating the understanding and implementation of these processes. As a result, this work has produced a comprehensive guide, documenting in detail the process of developing and implementing Pipelines, providing a valuable source of reference for FeMASS students.

Key words: Pipelines. DevOps. Continuous Integration. Continuous Delivery.

## LISTA DE FIGURAS

<b>Figura 1</b> – DevOps – Integrando Desenvolvimento e Operações .....	22
<b>Figura 2</b> – Simplificação da Pipeline DevOps .....	23
<b>Figura 3</b> – Erro durante pipeline. ....	24
<b>Figura 4</b> – Ciclo CI/CD .....	25
<b>Figura 5</b> – CI/CD como extensão. ....	26
<b>Figura 6</b> – Desenvolvimento com SonarQube. ....	27
<b>Figura 7</b> – Aplicações em containers versus aplicações em VMs.....	29
<b>Figura 8</b> – Exemplo de Dockerfile para uma aplicação java.....	30
<b>Figura 9</b> - Fluxograma da solução.....	34
<b>Figura 10</b> – Tela de assinatura do GitHub Student. ....	35
<b>Figura 11</b> – Área logada do GitHub Student.....	35
<b>Figura 12</b> – Conexão com o Microsoft Azure.....	36
<b>Figura 13</b> – Área logada Microsoft Azure. ....	36
<b>Figura 14</b> – Criação cluster Kubernetes.....	37
<b>Figura 15</b> – Utilização SonarLint. ....	38
<b>Figura 16</b> – Pipeline Azure java maven. ....	38
<b>Figura 17</b> – Pipeline Azure erro por assinatura.....	39
<b>Figura 18</b> – Pipeline GitHub Actions CI.....	40
<b>Figura 19</b> – Pipeline GitHub Actions <i>build</i> completa.....	41
<b>Figura 20</b> – Pipeline GitHub Actions testes realizados.....	41
<b>Figura 21</b> – Email SonarSource. ....	42
<b>Figura 22</b> – Pipeline SonarCloud. ....	43
<b>Figura 23</b> – Secrets Repositório. ....	44
<b>Figura 24</b> – Quality Gate SonarCloud.....	45
<b>Figura 25</b> – Análise Security Hotspots.....	45
<b>Figura 26</b> – Análise Testes de Cobertura. ....	46
<b>Figura 27</b> – Análise linhas duplicadas.....	47
<b>Figura 28</b> – Análise manutenção do código. ....	47
<b>Figura 29</b> – Análise clean code. ....	48
<b>Figura 30</b> – Fluxo pipeline CD.....	48
<b>Figura 31</b> – Informações básicas pipeline CD. ....	49
<b>Figura 32</b> – buildImage job parte 1. ....	50

<b>Figura 33</b> – buildImage job parte 2. ....	50
<b>Figura 34</b> – Certificados e segredos Azure .....	51
<b>Figura 35</b> – Dockerfile da solução. ....	51
<b>Figura 36</b> – <i>Deploy</i> job.....	52
<b>Figura 37</b> – Arquivo manifesto. ....	53
<b>Figura 38</b> – <i>Build</i> e push da imagem para o ACR.....	54
<b>Figura 39</b> – <i>Deploy</i> da aplicação. ....	54
<b>Figura 40</b> – Guia de implementação e Desenvolvimento de Pipelines DevOps .....	55
<b>Figura 41</b> – Arquivos gerados durante o presente trabalho.....	56
<b>Figura 42</b> – Recurso novo. ....	57
<b>Figura 43</b> – Abertura de Pull Request para develop.....	58
<b>Figura 44</b> – Pipelines de CI e análise de código no sonar.....	58
<b>Figura 45</b> – Análise SonarCloud Pull Request.....	59
<b>Figura 46</b> – Sucesso pipelines CD e análise de código. ....	59
<b>Figura 47</b> – Análise da <i>branch</i> develop após pull request. ....	60
<b>Figura 48</b> – <i>Deployment</i> feito no AKS.....	60
<b>Figura 49</b> – Consulta do IP externo do serviço. ....	60
<b>Figura 50</b> – Consulta de logs do pod.....	61

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>16</b>
<b>2</b>	<b>REFERENCIAL TEÓRICO .....</b>	<b>22</b>
2.1	DEVOPS.....	22
2.2	PIPELINES.....	23
2.3	INTEGRAÇÃO CONTÍNUA E ENTREGA CONTÍNUA (CI/CD).....	24
2.3.1	<i>Integração Contínua (CI)</i> .....	25
2.3.2	<i>Entrega contínua (CD)</i> .....	25
2.4	TESTES AUTOMATIZADOS.....	26
2.4.1	<i>Lint</i> .....	27
2.4.2	<i>SonarQube</i> .....	27
2.4.3	<i>SonarCloud</i> .....	27
2.4.3.1	<i>QualityGate</i> .....	28
2.5	DOCKER .....	28
2.5.1	DOCKERFILE .....	29
2.6	KUBERNETS.....	30
2.7	GITHUB ACTIONS.....	30
<b>3</b>	<b>PLANEJAMENTO ESTRUTURAL DA SOLUÇÃO DE DEVOPS.....</b>	<b>32</b>
3.1	FLUXOGRAMA DA SOLUÇÃO.....	33
<b>4</b>	<b>IMPLEMENTAÇÃO DA PIPELINE DEVOPS .....</b>	<b>35</b>
4.1	IMPLEMENTAÇÃO DO CI A PARTIR DA AZURE PIPELINES .....	35
4.2	IMPLEMENTAÇÃO DO CI A PARTIR DO GITHUB ACTIONS .....	39
4.3	IMPLEMENTAÇÃO DA ANÁLISE DE CÓDIGO COM SONARQUBE .....	42
4.4	IMPLEMENTAÇÃO DA ANÁLISE DE CÓDIGO COM SONARCLOUD.....	42
4.5	IMPLEMENTAÇÃO DO CD A PARTIR DO GITHUB ACTIONS .....	48
4.6	CRIAÇÃO DO GUIA DE IMPLEMENTAÇÃO E DESENVOLVIMENTO DE PIPELINES DEVOPS.....	55
<b>5</b>	<b>UTILIZAÇÃO DA PIPELINE DEVOPS .....</b>	<b>57</b>
	<b>CONSIDERAÇÕES FINAIS.....</b>	<b>62</b>
	<b>REFERÊNCIAS .....</b>	<b>64</b>
	<b>ANEXO A – EMENTA DISCIPLINA DE DESENVOLVIMENTO DE</b>	
	<b>SISTEMAS II FEMASS.....</b>	<b>65</b>
	<b>ANEXO B – GUIA DE IMPLEMENTAÇÃO E DESENVOLVIMENTO DE</b>	
	<b>PIPELINES DEVOPS.....</b>	<b>66</b>
	<b>ANEXO C – ANÁLISE SONARCLOUD .....</b>	<b>67</b>

<b>ANEXO D – DECLARAÇÃO DE CORREÇÃO GRAMATICAL.....</b>	<b>68</b>
---	-----------

## 1 INTRODUÇÃO

A comunicação cliente-servidor é definida como a relação entre dois programas que se comunicam entre si e, desde os primórdios da computação, costumava encontrar problemas no procedimento de hospedar aplicações. No início, o conceito utilizado era o de tempo compartilhado, a partir do qual os usuários compartilhavam os recursos computacionais, gerando o aumento da velocidade das aplicações, porém ele gerava diversos paradigmas e levou a grandes mudanças tecnológicas.

Com a evolução tecnológica, obteve-se o processo de virtualização que mitigou os problemas então existentes, mas os procedimentos de implementação e desenvolvimento criavam outros problemas que aumentavam a complexidade do desenvolvimento.

Atualmente, através de uma pipeline DevOps, é possível realizar tais procedimentos com auxílio da tecnologia em Cloud (Nuvem), endereçando muitos dos problemas supracitados, tornando-se a nuvem então uma solução cada vez mais presente nos projetos de desenvolvimento.

O termo DevOps é uma mistura das palavras desenvolvimento e operações e, segundo a Microsoft (2023), “é a união de pessoas, processos e tecnologias para fornecer continuamente valor aos clientes.”<sup>1</sup>. Em suma, a ideia é unir equipes, que antes trabalhavam em silos separados, em uma grande equipe cujo objetivo é realizar entregas coordenadas de maneira colaborativa, visando ao aumento da qualidade e da confiabilidade do produto entregue.

Para Bass, Weber & Zhu (2015), a pipeline DevOps é um conjunto de processos e práticas que as equipes utilizam para acelerar e facilitar as atividades realizadas, sendo seu principal objetivo automatizar a entrega de software, fazendo com que sua qualidade seja garantida por diversos testes, porém feitos de maneira rápida. Todavia hoje tais processos ainda não são tão explorados no contexto do ensino superior e é importante aproximar o mundo acadêmico ao mundo corporativo. É visível que, na grade do curso de Sistemas de Informação da Faculdade Professor Miguel Ângelo da Silva Santos (FeMASS), não existe uma disciplina que trate desse assunto.

Portanto, a proposta do presente trabalho é criar um guia (manual) para que os alunos da FeMASS consigam desenvolver suas demandas de forma célere com o uso das pipelines integrando ferramentas. Tais demandas podem ser provenientes tanto de trabalhos científicos e

---

<sup>1</sup> Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-devops>. Acesso em 13 mar. 2023.

acadêmicos (como artigos ou projetos), tanto quanto de trabalhos corporativos, a fim de trazer mais agilidade aos processos tidos como manuais.

É ainda de interesse deste trabalho documentar a pipeline desenvolvida e implementada, visando prover qualquer pessoa a utilizá-lo com todos os insumos necessários para desenvolverem uma pipeline DevOps.

O objetivo geral proposto é o de criar e implementar um guia de desenvolvimento, implementação e boas práticas de uma pipeline DevOps para os alunos da FeMASS.

Os Objetivos Específicos são os seguintes:

- desenvolver uma pipeline DevOps;
- documentar o código gerado;
- implementar a pipeline gerada em um escopo de projeto de sistemas;
- criar um guia contemplando os tópicos supracitados;
- prover os alunos da FeMASS com as informações necessárias para implementarem com autonomia uma pipeline no seu trabalho ou nas disciplinas ministradas na faculdade.

Trazer para o ambiente acadêmico da FeMASS tópicos que estão em alta no mercado de Tecnologia e Informação (TI), como os objetivos deste presente estudo, podem impactar diretamente no futuro acadêmico e profissional dos alunos dessa faculdade.

Com a ausência de uma ou mais disciplinas que lecionem diretamente sobre o tema “DevOps e ou pipelines”, o autor deste trabalho, que desenvolve e documenta pipelines DevOps profissionalmente há mais de um ano, tem a intenção de trazer tal tópico para seus pares da FeMASS. Visa também não somente informar e instruir os alunos dessa instituição de ensino, em especial os da graduação em Sistemas de Informação, em como desenvolver e implementar uma pipeline DevOps, mas também como relatar os passos em um guia.

Segundo Bass, Weber & Zhu (2015), DevOps é uma tendência e uma realidade no mercado de TI. Isto é evidente na pesquisa da Atlassian (2020)<sup>2</sup> com mais de 500 profissionais de DevOps, em que 50% das empresas afirmaram praticar o DevOps há mais de três anos. Entre os entrevistados, 99% afirmaram que o DevOps tem um impacto positivo nas suas organizações.

Com a necessidade cada vez maior do uso de softwares sempre atualizados e novas ferramentas, o DevOps demonstra cada vez mais ser uma poderosa ferramenta para atingir tal objetivo.

---

2 Disponível em: <https://www.atlassian.com/whitepapers/devops-survey-2020>. Acesso: 13 mar 2023

Enquanto, na maioria das empresas, as atualizações ocorrem uma vez a cada nove meses, em multinacionais de TI de larga escala, como Netflix, Amazon e Google são liberados por dia milhares de atualizações de seus softwares, como é visto na Tabela 1, que compara a frequência de *deploy* entre diversas empresas. Essa frequência só é possível porque elas possuem seus processos de DevOps bem construídos, como é evidenciado na Tabela 1, criando ciclos cada vez mais curtos de entregas.

**Tabela 1** – Atualizações diárias em empresas de TI<sup>3</sup>

<b>Empresa</b>	<b>Frequência de deploy</b>	<b>Prazo de entrega de deploy</b>	<b>Confiabilidade</b>	<b>Receptividade dos clientes</b>
Amazon	23.000 por dia	Minutos	Alta	Alta
Google	5.500 por dia	Minutos	Alta	Alta
Netflix	500 por dia	Minutos	Alta	Alta
Facebook	1 por dia	Horas	Alta	Alta
Twitter	3 por semana	Horas	Alta	Alta
Empresa comum	Uma a cada 9 vezes	Meses ou trimestres	Baixa/Média	Baixa/Média

Fonte: <https://www.oreilly.com/library/view/the-phoenix-project/9781457191350/42-resourceWhy.xhtml>. Acesso em: 02 mai 2023.

Um dos fatores primordiais para o sucesso do DevOps são as ferramentas certas e, a pipeline DevOps surge, nesse contexto, como uma ferramenta extremamente utilizada, visto que integra o conceito de Integração Contínua e da Entrega Contínua (CI/CD). Esses conceitos permitem que os desenvolvedores possam atualizar e entregar seus softwares sem terem que manualmente verificar a integridade dele.

A capacidade de automatizar processos que antes eram manuais no mercado é altamente desempenhada, encorajada e procurada por recrutadores que visam trazer mais agilidade para seus processos e entregar produtos com cada vez mais qualidade para seus clientes, além de minimizar o tempo gasto por desenvolvedores com atividades que outrora seriam repetitivas.

A pipeline DevOps também é crucial para detecção de erros nos códigos desenvolvidos, impedindo a continuidade do processo com um código falho, inibindo assim que atualizações falhas sejam entregues ao cliente ou mesmo que todo o software seja inutilizado por conta de um erro não esperado.

---

<sup>3</sup> Tradução livre.



A possibilidade de detectar erros existe a partir da automação de testes como teste unitário, teste de integridade, *smoke test*, entre outros, os quais serão apresentados, desenvolvidos e implementados neste trabalho e estarão presentes no guia.

Sua hipótese central é a de que, com documentação suficiente, tanto no desenvolvimento quanto na implementação da referida ferramenta, em um contexto de desenvolvimento de software, os alunos da FeMASS possam se instruir e utilizar os artefatos gerados no seu dia a dia acadêmico e profissional, como um guia básico para conseguirem impulsionar suas carreiras, sejam elas acadêmicas ou profissionais.

Este trabalho objetiva instruir, informar e instigar os alunos da FeMASS sobre o que é uma pipeline DevOps, introduzindo o próprio conceito de DevOps e utilizando da pipeline como o objeto de estudo central. Propondo, portanto, desenvolver, documentar e implementar uma pipeline em um contexto de projeto de software e, propondo um guia para atingir os objetivos supracitados.

Para atingir esse objetivo, o autor utilizará um percurso metodológico qualitativo, que inclui a observação da realidade do mercado de TI, em especial das pipelines DevOps, versus a realidade das disciplinas ministradas na instituição onde realiza o presente trabalho.

De acordo com Gil (2008, p.16), “nos experimentos, o cientista toma providências para que alguma coisa ocorra, a fim de observar o que se segue, ao passo que, no estudo por observação, apenas observa algo que acontece ou já aconteceu”. Visando atingir esse objetivo também é proposto um estudo de caso de cunho experimental com uma etapa propositiva, tendo em vista estudar e experimentar o desenvolvimento e a implementação de uma pipeline em um projeto de software, seja ele proveniente de uma das disciplinas ministradas na faculdade ou realizado de maneira a integrar este trabalho unicamente.

O estudo de caso de acordo com Gil (2010, p. 37) “consiste no estudo profundo e exaustivo de um ou mais objetos, de maneira que permita seu amplo e detalhado conhecimento”.

O método experimental, por sua vez, tem sua essência, de acordo com Gil (2008, p. 16), submeter os objetos de estudo à influência de certas variáveis, em condições controladas e conhecidas pelo investigador, para observar os resultados que a variável produz no objeto.

A etapa propositiva se dá a partir da criação e a disponibilização de um guia que ilustre como desenvolver e implementar uma pipeline DevOps. Ele contará com o código gerado no desenvolvimento da pipeline documentado, suas boas práticas de desenvolvimento, desenvolvimento de testes como teste unitário, *smoke test*, entre outros, suas etapas de

implementação, e como acessar as ferramentas utilizadas tanto neste trabalho quanto no guia de maneira gratuita.

A primeira ferramenta selecionada para compor o contexto de desenvolvimento tanto da pipeline como de um software será o Git, um sistema de controle de versão utilizado de maneira abrangente por profissionais da área de TI, visto que ele permite o versionamento (controle da versão) de um software específico ou da própria pipeline, habilitando a existência de diferentes ambientes para se desenvolver e disponibilizar softwares. Sua escolha se dá também por ser um software disponibilizado de graça a terceiros.

Algumas das ferramentas empregadas só podem ser utilizadas por meio de pagamento de assinatura ou de taxas variáveis a depender do uso, mas foi realizada uma assinatura sem custos do GitHub Education pelo autor, mediante o seu comprovante de matrícula na faculdade, habilitando-o a usar somente algumas das ferramentas que serão expostas neste trabalho, mas também outras que não são disponibilizadas sem custos.

O serviço de computação em nuvem da Microsoft, a Azure, foi selecionado por ter ferramentas essenciais para o objetivo deste trabalho. Uma conta foi criada nessa nuvem, via assinatura do GitHub Education.

O subserviço central utilizado será o Azure DevOps, o qual contém ferramentas essenciais para a utilização da cultura DevOps e para o objetivo do trabalho. No serviço, foi utilizado, em primeiro nível, o Azure Boards, que permite o planejamento de sprints (conjunto de atividades que devem ser desenvolvidas em um espaço pré-determinado de tempo) e de reportar atualizações e ou *bugs* (falhas) no código mantido.

A ferramenta Azure Repos representa o repositório Git privado e hospedado na nuvem para o trabalho. Este repositório poderia ser de qualquer outro serviço, como o próprio GitHub, porém a integração com as pipelines será um diferencial para o desenvolvimento ágil do trabalho.

As pipelines serão utilizadas pela ferramenta do Azure Pipelines, que permite a integração com o Azure Repos e o Azure Boards, habilitando a criação de pipelines hospedadas em nuvem para inúmeros Sistemas Operacionais, como Windows, Linux e macOS.

Esta ferramenta é a responsável pela etapa experimental do estudo de caso. Ela permite a utilização de qualquer linguagem de programação, não sendo um requisito uma linguagem específica. Ela também será responsável por deter todos os testes realizados no código gerado e por garantir que ele possa seguir seu caminho até a disponibilização, garantindo assim a mitigação de erros e, reduzindo o esforço por etapas manuais para a disponibilização de atualizações.

A linguagem de programação utilizada é Java, linguagem de programação Orientada a Objetos e muito operada tanto no mercado de TI quanto nas disciplinas oferecidas na FeMASS. Sua escolha é feita pela experiência do autor de se servir constantemente dela.

A última ferramenta selecionada será o repositório GitHub, um serviço de repositórios como o Azure Repos, porém a utilização dessa ferramenta se dá a partir da etapa propositiva deste trabalho. Sua utilidade será a de persistir todo o código gerado e de expor o guia, a fim de manter o produto do desenvolvimento em um serviço que não depende do autor ou de terceiros possuírem uma assinatura válida na Azure e ou no GitHub Education, também sendo este um repositório público, diferentemente do Azure Repos que se mantém privado.

É válido ressaltar que o Azure Repos continua essencial para a utilização das capacidades totais das demais ferramentas escolhidas e, mesmo sem a assinatura desse serviço, o guia e o código gerado podem servir como base de estudo e prática para o desenvolvimento de pipelines em outras ferramentas.

O primeiro capítulo descreve os principais pontos abordados no presente trabalho, contextualizando com os objetivos propostos e introduzindo o leitor no assunto proposto e na metodologia utilizada.

No segundo capítulo, é apresentada a revisão bibliográfica, expondo todos os conceitos pertinentes para o entendimento do tema proposto e contextualizando o leitor a respeito dos conceitos técnicos usados, além das particularidades dos tipos de testes automatizados entre outros elementos cruciais para implementação de uma pipeline.

O terceiro capítulo é o responsável pela explicação do trabalho desenvolvido, descrevendo e analisando o estudo de caso, demonstrando a diferença entre as abordagens para implementação de uma pipeline e o caminho tomado no contexto do presente trabalho, a fim de definir as fronteiras das estratégias de teste e a arquitetura de DevOps montada.

O quarto capítulo é responsável pela explicação sobre a implementação do trabalho proposto, deixando explícito como a pipeline foi implementada e desenvolvida e em qual contexto está foi inserida. Também serão apresentados os artefatos gerados.

O quinto capítulo se refere à validação do trabalho proposto, mostrando como ele funciona e como é possível validar um projeto desse tipo. Os testes gerados estão dispostos nesse capítulo a fim de demonstrar as particularidades de cada um.

No sexto capítulo, há uma conclusão deste trabalho que leva em consideração toda a implementação, estudo e validação gerados anteriormente, a fim de concluir sua hipótese inicial.

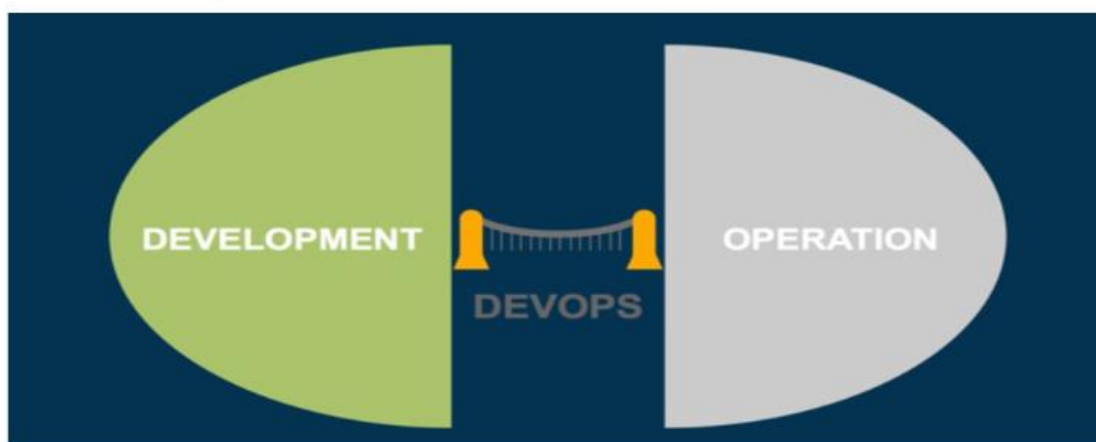
## 2 Referencial Teórico

Visando compreender todo o contexto necessário para o desenvolvimento de uma pipeline DevOps, é necessário entender conceitos específicos que lhe são relacionados, como o DevOps propriamente, CI/CD e suas diferenças, testes automatizados, ferramentas de validação de testes e as soluções utilizadas, a fim de contribuir com a pipeline.

### 2.1 DevOps.

Segundo Pereira Neto (2022a, p. 21), “DevOps é o conjunto de processos e práticas que envolve pessoas, tecnologias e conhecimento, buscando o alinhamento da TI com a estratégia organizacional.”, significando que a prática do DevOps e de suas ferramentas deve não só envolver a área ou os profissionais de TI como também as pessoas que ali estão inseridas, geralmente na área de operações, a fim de que a estratégia organizacional esteja devidamente alinhada com as definições encontradas na TI.

**Figura 1**– DevOps – Integrando Desenvolvimento e Operações



Fonte: Neto (2022a, p. 21)

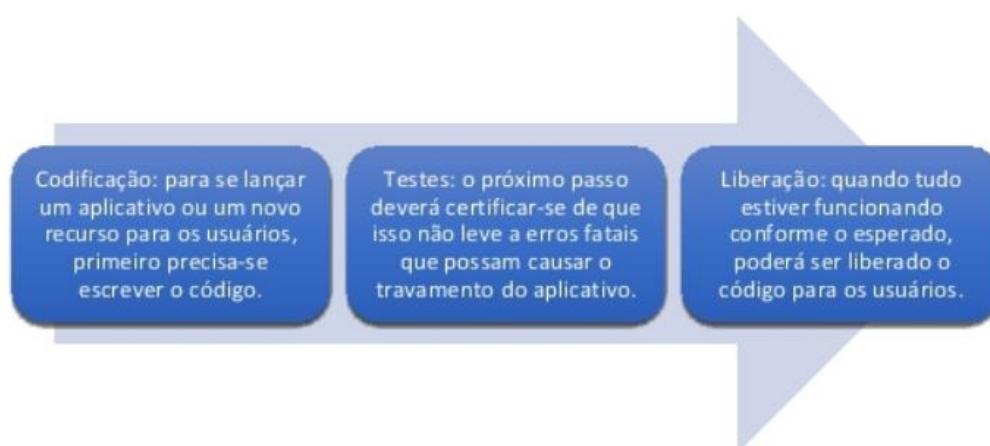
Segundo Pereira Neto (2022a), o DevOps exige a colaboração entre equipes de desenvolvimento e operações, visando trazer para o ambiente produtivo um código de software que pode ser atualizado de maneira mais rápida, mais repetitiva e automatizada. Ainda de acordo com Pereira Neto, o DevOps aumenta a eficiência, a eficácia e a efetividade de uma organização para desenvolver e operar um software.

## 2.2 Pipelines.

“Uma Pipeline DevOps é um conjunto de processos e práticas que as equipes de desenvolvimento (Dev) e operações (Ops) implementam para criar, testar e implementar um software com mais rapidez e facilidade.”<sup>4</sup> (Bass, Weber, & Zhu, 2015, p. 213).

O grande objetivo de uma pipeline DevOps é automatizar processos, principalmente a entrega do código para o ambiente de produção e seus testes, garantindo que o código gerado passou por todas as etapas necessárias e está com sua qualidade garantida.

**Figura 2–** Simplificação da Pipeline DevOps



Fonte: Neto (2022b, p. 8)

Não existe um padrão universal de pipelines nem uma especificação universal, portanto cada pipeline é individual perante o seu contexto de implementação, não somente por endereçar problemas distintos por diversas vezes, mas também por geralmente ser construída a partir de algumas tecnologias específicas, podendo se moldar de diferentes maneiras. A depender da necessidade, no caso do presente trabalho, tal pipeline se encontra no contexto pós-dev, que significa após o desenvolvimento de uma parte nova do código.








Ainda assim, mesmo que cada pipeline seja única, sua estruturação costuma ser muito homogênea entre os projetos e as organizações que a utilizam. Ela, em si, é um arquivo (.YAML) que contém diversas atividades (*tasks*) a serem realizadas antes de alguns processos como *build* (construção), *package* (empacotamento), entre outros. Cada atividade pode ser colocada em diferentes etapas (*steps*) e cada etapa possui seu próprio sucesso ou falha, incorrendo que, se uma etapa falhar (sequencial ou paralela), a pipeline como um todo é

---

4 Tradução livre

interrompida e fornece a quem está utilizando o que ocorreu para uma tomada de decisão, como a Figura 3.

**Figura 3** – Erro durante pipeline.

>		Install npm and yarn tools	8s
∨		build code and publish artifact	4s
		Initialize job	<1s
		Checkout ConsoleApp2@main to s	3s
		build code	
		this task will fail	
		Post-job: Checkout ConsoleApp2@...	

Fonte: Azure DevOps best practices – jobs and stages<sup>5</sup>

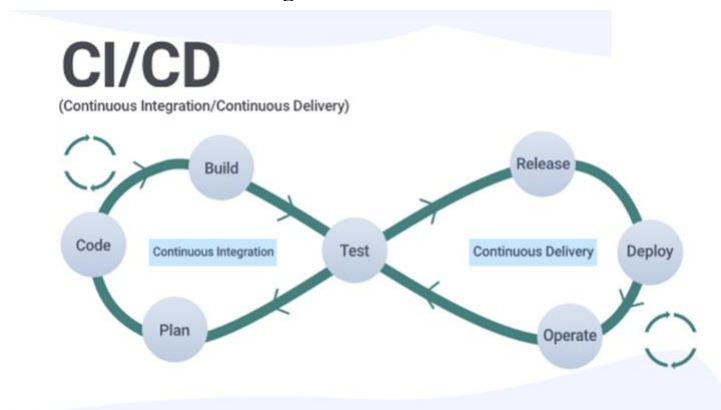
### 2.3 Integração contínua e Entrega Contínua (CI/CD).

Segundo Pereira Neto (2022b, p. 11), “Para garantir que o código passe de um estágio para o próximo e assim sucessivamente, é necessário implementar vários processos e práticas de DevOps. Os mais importantes são a Integração Contínua (Continuous Integration – CI) e a Entrega Contínua (Continuous Delivery – CD)”, significando então que a aplicação do CI/CD é de extrema importância no contexto não somente das pipelines, mas do DevOps como um todo. Ainda de acordo com Pereira Neto (2022b) CI/CD são basilares para o DevOps, contudo é necessário refletir que CI e CD são processos distintos, que possuem suas próprias características e suas individualidades.

5

Disponível em: <https://blog.geralexgr.com/azure/azure-devops-best-practices-jobs-and-stages>. Acesso em 11 jun. 2023.

Figura 4– Ciclo CI/CD



Fonte: Neto (2022b, p. 14)

### 2.3.1 Integração Contínua (CI)

Segundo Pereira Neto (2022b, p. 14), “A integração contínua é um processo que envolve a integração de pequenos pedaços de código de vários desenvolvedores em um repositório de código compartilhado com a maior frequência possível.”. Esse processo se dá no ato de mesclar (merge) diferentes trechos de código de diferentes pessoas em uma estrutura única compartilhada, que é constantemente testada.

Em suma, a integração contínua é a prática de fazer *commits* (enviar trechos novos de códigos) constantemente em um código fonte comum, que se encontra em um repositório remoto, com a finalidade de realizar testes automáticos, visando detectar erros e acelerar o processo de correção deles ainda em ambiente de desenvolvimento.

### 2.3.2 Entrega contínua (CD)

O CD pode ser visto como uma extensão do CI, no sentido de serem processos diferentes e praticamente sequenciais, uma vez que o processo de CD tem como finalidade a aceleração do processo de *deploy* (disponibilizar o software atualizado) a partir do incentivo aos desenvolvedores a enviarem pequenos trechos de código por vez, possibilitando a solução de erros e problemas no código de maneira mais ágil do que se fosse enviado um grande trecho de código.

**Figura 5** – CI/CD como extensão.



Fonte: Neto (2022b, p. 42)

Para Pereira Neto (2022b, p. 41), o principal objetivo da entrega contínua é garantir o mínimo esforço na implementação de novos códigos, todavia o autor do presente trabalho discorda que esse seja o objetivo principal, este autor que o principal objetivo da implementação do processo de CD é realizar atualizações. Pois observa que com o menor risco possível, trazendo a agilidade e a celeridade para todo o processo é garantindo tanto a qualidade do produto, mesmo que ele seja entregue em pequenas etapas, quanto a qualidade do ambiente de desenvolvimento do time de profissionais como um todo. Isso possibilita a redução de *burnouts* (ou esgotamento profissional), uma vez que cada profissional tem pequenas atividades para desenvolver, ao longo do seu dia, e, não, uma atividade gigante com um prazo de entrega irreal.

## 2.4 Testes automatizados

Desde os anos noventa, quando foi iniciado o entendimento de que o desenvolvimento de software deveria vir acompanhado dos testes, o desenvolvimento orientado a testes vem sendo utilizado de maneira rotineira por desenvolvedores, tornando a garantia de qualidade o objetivo central de muitas empresas de tecnologia.

E o que são testes? Para Pereira Neto (2022c, p. 41), é uma investigação realizada para fornecer informações sobre a qualidade do software. Em suma, os testes são validações daquele software gerado que buscam garantir que ele, além de funcional, está apto a responder aos objetivos de negócio para o qual foi desenvolvido.

Para a discussão deste trabalho, a automação dos testes visando testar o código de maneira contínua é um dos pilares de toda pipeline. Em qualquer processo de DevOps, as alterações, em um código, passam do estágio de desenvolvimento para o teste e consequentemente para implementação. Para atingir tal estágio é necessário haver testes unitários, *smoke tests* (testes de fumaça) testes de cobertura, entre outros.

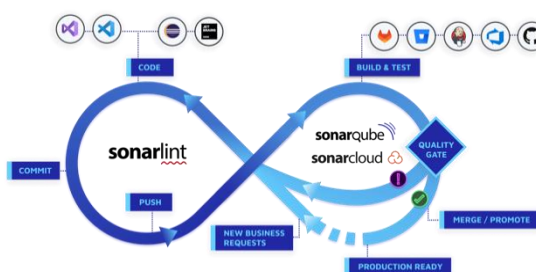


### 2.4.1 Linter

*Linter* é uma ferramenta que analisa o código fonte e encontra erros de sintaxe, *bugs*, problemas estruturais, entre outros. O *linter* faz parte do teste de White-box, que tem o propósito de, a partir de regras e expectativas pré-definidas, analisar a estrutura interna de um componente específico ou de um sistema todo. Enquanto um teste de unidade verifica como um componente deveria se comportar, o *linter* checa como ele deveria estar estruturado.

Nesse contexto, será utilizado, em um primeiro momento, o SonarLinter, uma ferramenta de revisão automática de código que irá ajudar a escrita de um código limpo. Ele foi utilizado tanto para servir como *linter*, quanto para a análise de código.

**Figura 6** – Desenvolvimento com SonarQube.



Fonte: SonarQube Documentation<sup>6</sup>

### 2.4.2 SonarQube

O Sonarqube é um software que realiza a checagem da qualidade do código. Diferentemente do SonarLinter, este é para um projeto todo e, não, para somente um arquivo de cada vez. Ele garante que o código está de acordo com as boas práticas, sem *bugs* e sem maiores problemas de segurança.

### 2.4.3 SonarCloud

O SonarCloud é um software que, tal qual o SonarQube, realiza checagem da qualidade do código, porém não é implementado diretamente na máquina de quem utiliza ou em uma aplicação específica e, sim, de maneira online utilizando o próprio site do SonarCloud.

<sup>6</sup> Disponível em: <https://docs.sonarqube.org/latest/>. Acesso em 11 jun. 2023.

### 2.4.3.1 QualityGate

O Qualitygate representa como a análise do SonarCloud irá se comportar. De acordo com a documentação do SonarQube<sup>7</sup>, ele reforça a política de qualidade de um projeto garantindo que ele está pronto para ser publicado. Essa ferramenta utiliza para tal política algumas condições que podem ser, por exemplo:

- Cobertura de código acima de 80% em todo novo código gerado;
- Ausência de *bugs*;
- Ausência de erros.

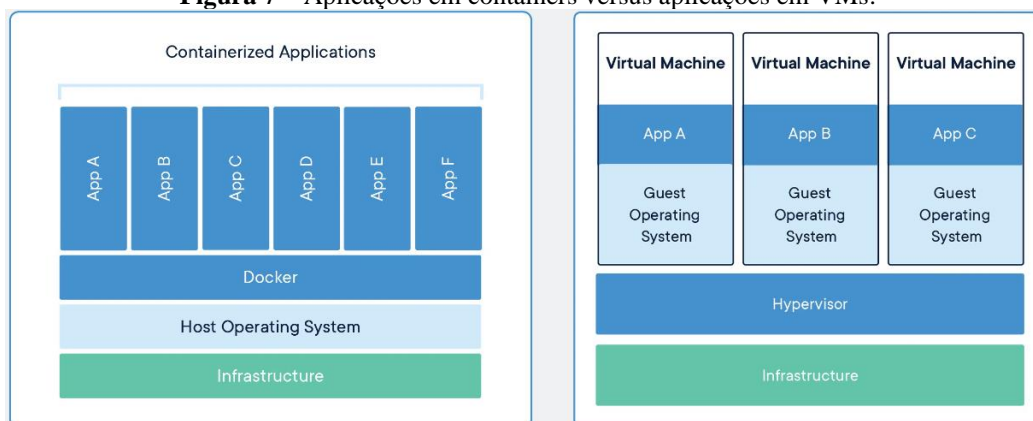
É válido ressaltar que a versão gratuita do SonarCloud não permite customizar tais condições, utilizando somente as definições padrão, porém tal análise, incorporada na rotina de desenvolvimento de software, pode trazer muitos benefícios e garantir a inexistência de *bugs* ou erros.

## 2.5 Docker

Docker é uma ferramenta de DevOps que executa aplicações distribuídas em sistemas múltiplos, permitindo tanto o processo de *build* quanto o de *ship*. Também possibilita escalonar a aplicação em containers distintos, que são uma abstração que permite empacotar tanto o código gerado quanto as dependências desse código juntos. É possível ter múltiplos containers rodando diferentes aplicações, ou diferentes instâncias de uma mesma aplicação, dentro de uma mesma máquina (física ou virtual), cada um deles usando, de maneira isolada, processos diferentes em seus espaços de usuário. Containers também consomem menos espaço que VMs (Máquinas Virtuais), podendo também cuidar de mais aplicações, necessitando de menos VMs e máquinas físicas.

---

<sup>7</sup> Disponível em: <https://docs.sonarsource.com/sonarqube/latest/user-guide/quality-gates/>. Acesso em 25 jun. 2023.

**Figura 7** – Aplicações em containers versus aplicações em VMs.

Fonte: Use containers to *Build*, Share and Run your applications<sup>8</sup>

### 2.5.1 Dockerfile

Para um Docker poder construir um container, de fato, ele precisa de um arquivo específico, conhecido como Dockerfile. Esse arquivo pode gerar a *build* de imagens, arquivos usados para executar código em um container Docker, a partir de instruções de um Dockerfile, um documento de texto que contém todos os comandos que um usuário pode usar utilizando linha de comando para resultar uma imagem.

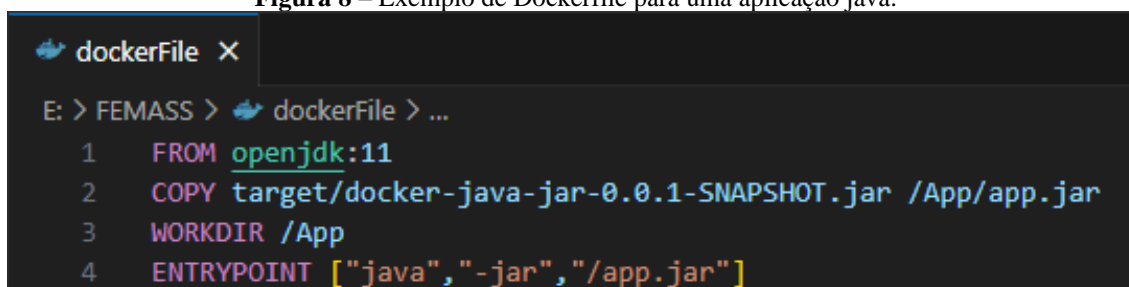
No contexto do presente trabalho, será utilizado o DockerFile para levar ao Docker uma aplicação java. Para isso são necessários alguns passos no arquivo, que são os seguintes:

- de onde é a imagem utilizada (FROM): nesse caso, é da openjdk, utilizando o java 11, porém poderia ser uma versão mais atual como 17 ou outra, bastando checar no repositório de imagens oficial do Docker;
- definir de onde será copiado o arquivo (COPY): nesse caso, será copiada somente a aplicação (target/docker-java-jar-0.0.1-SNAPSHOT.jar), que se encontra na pasta “target” para a pasta “App” e renomeando o arquivo para “app.jar”;
- definir o diretório de trabalho (WORKDIR): nesse caso, o diretório utilizado será o /App, criado no passo anterior;
- definir que o container deve rodar como um executável (ENTRYPOINT): nesse caso, são dados os comandos “java -jar” que definem que uma aplicação java de extensão jar será executada e, após tais comandos, é exposta a aplicação (“/app.jar”), uma vez que já estamos no diretório do passo anterior;

<sup>8</sup> Disponível em: <https://www.docker.com/resources/what-container/>. Acesso em 11 jun. 2023.

É válido ressaltar que a Figura 8, a seguir, não passa de um exemplo fictício, porém poderia já ser empregado em uma aplicação real. O ideal para tal arquivo seria não só consumir a aplicação, mas garantir que o projeto pode de fato ser compilado, testado e então empacotado para gerar uma aplicação.

**Figura 8** – Exemplo de Dockerfile para uma aplicação java.



```
dockerFile X
E: > FEMASS > dockerFile > ...
1 FROM openjdk:11
2 COPY target/docker-java-jar-0.0.1-SNAPSHOT.jar /App/app.jar
3 WORKDIR /App
4 ENTRYPOINT ["java", "-jar", "/app.jar"]
```

Fonte: Elaboração própria.

## 2.6 Kubernetes

Segundo o site do Docker (2023), “Kubernetes é um sistema de orquestração *open source* (código aberto) para automação, gerenciamento, posicionamento, dimensionamento e roteamento de containers”<sup>9</sup>. Porém por que usar tal ferramenta? Devemos usá-la porque deles provêm três principais bases que são primordiais no contexto de uma pipeline DevOps:

- disponibilidade: o Cluster de Kubernetes tem uma alta tolerância a falhas, permitindo escalar operações para valores altos.
- escalonamento automático: os Kubernetes podem escalar os recursos para mais ou para menos a depender do tráfego de rede e da velocidade do servidor de maneira automática.
- amplo ecossistema: possuem um vasto ecossistema para interface de rede dos containers e para a interface de armazenamento, além de ferramentas de monitoramento.

## 2.7 GitHub Actions

De acordo com a documentação do GitHub (2023, “GitHub Actions é uma plataforma de integração contínua e entrega contínua (CI/CD) que permite automatizar a sua compilação,

---

<sup>9</sup> Disponível em: <https://www.docker.com/products/kubernetes/>. Acesso em 12 jun. 2023. Tradução livre.

testar e pipeline de implantação”<sup>10</sup>. Essa ferramenta será escolhida devido a ser gratuita e a possuir uma integração suficiente com as demais ferramentas utilizadas.

Alguns dos conceitos utilizados do GitActions e do GitHub são:

- *branches*: ramificações do código principal que permitem trabalhos concorrentes que eventualmente chegam ao código principal.
- Pull Request (PR): pedido de atualização em uma *branch*.
- Push: atualização de uma *branch*.

---

<sup>10</sup> Disponível em: <https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions/>. Acesso em 25 out. 2023.

### 3 Planejamento estrutural da solução de DevOps

O presente trabalho de conclusão de curso utilizará as etapas pós-desenvolvimento (pós-dev) como seu contexto central. Todo o código novo gerado como parte da estratégia de DevOps (relativo ao momento A da Figura 9) passará ainda na máquina dos desenvolvedores pelo SonarLint, visando garantir a idoneidade do código a partir dos princípios de boas práticas de desenvolvimento e Orientação a Objetos (OO).

Uma vez que o código passe pelo Linter supracitado sem erros, ele deve passar pela limpeza, pela instalação das dependências, pelo *build* e pela geração do pacote (.jar), a fim de garantir também a qualidade em um ambiente de desenvolvedor em primeiro momento. A pipeline entra em cena nessa etapa como uma validação de algumas dessas etapas, mantendo os padrões de qualidade estabelecidos previamente no escopo do projeto.

Quando o desenvolvedor realizar os trâmites iniciais para atualizar o código em ambiente remoto (no repositório), ele chegará na etapa de abrir uma PR (relativo ao momento B da Figura 9), o código será mantido em espera para atualizar o próximo ambiente remoto.

Enquanto o processo de atualização está esperando, a pipeline começa seus trabalhos no seu ambiente inicial (dev) e aplica os testes unitários, o teste de cobertura, o *smoke test*, entre outros, visando não haver erros (relativo ao momento C da Figura 9). Caso haja erros, a pipeline tirará essa PR do estágio de espera e ela ficará como não aprovada, visto que a pipeline não cumpriu uma etapa específica e o desenvolvedor responsável, ou outro qualquer, é notificado para corrigir os problemas encontrados.

Havendo integridade na conclusão das etapas supracitadas e não ocorrendo outros erros durante esses procedimentos, a pipeline libera a PR para o ambiente de garantia de qualidade (qa). As etapas necessárias para ela sair do ambiente dev e ir para o ambiente de qa são respectivamente as relacionadas abaixo:

1. não haver erros nos testes unitários;
2. haver, ao menos, 80% de cobertura de teste;
3. não haver erros nos smokes tests.

No ambiente de QA, o código está sujeito à garantia da geração de uma imagem de container, utilizando a ferramenta Docker na pipeline DevOps (relativo ao momento D da Figura 9). O arquivo de geração da imagem está pré-preparado, dentro do ambiente atual, para ser invocado pela pipeline. O arquivo passa primeiro por um estágio inicial que garante que o código gera um pacote com a extensão .jar, e novamente passando por testes impostos pela

geração de imagem (caso haja erro na geração da imagem, o código retorna outra vez para o time de desenvolvimento a fim de corrigi-lo.

Uma vez que a imagem seja gerada sem erros, a pipeline agora parte para o estágio de garantia da idoneidade dela, utilizando o SonarQube para análise da imagem. Outras ferramentas de análise de imagens podem ser utilizadas como AquaSec, entre outras.

Finalizada a checagem da imagem, o código então é empurrado para o ambiente de produção onde a imagem gerada é registrada no Container Registry (relativo ao momento E da Figura 9) e utilizada a partir de um arquivo de manifesto (.yaml) para o Kubernetes (relativo ao momento F da Figura 9), a fim de usar essa imagem em um *deploy* final. Se não houver erros no processo, então a atualização é liberada para uso de terceiros.

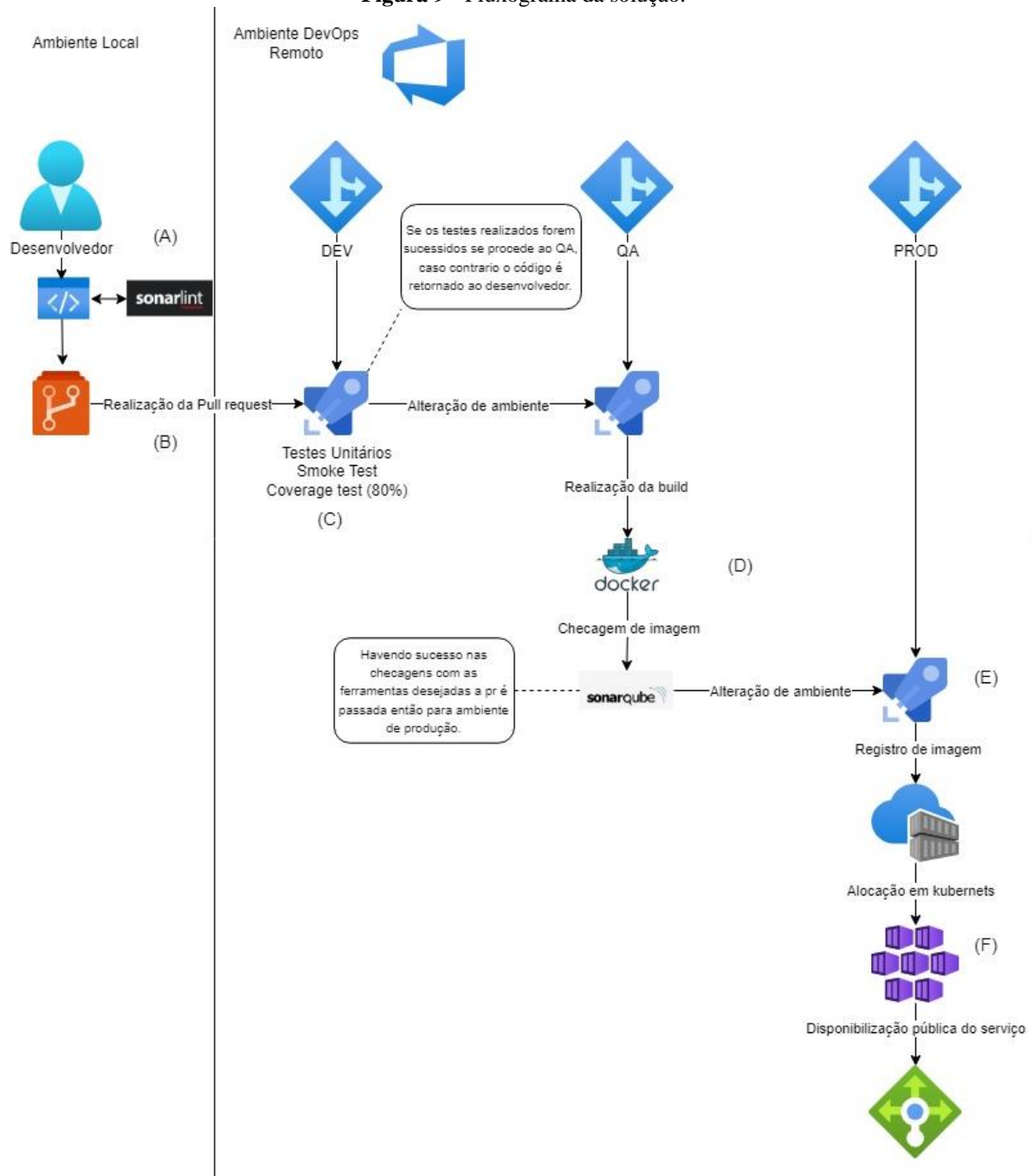
O plano de fundo deste trabalho é o código java gerado para o *backend* do projeto realizado na disciplina Desenvolvimento de Sistemas II, ministrada na presente faculdade. A ementa da disciplina segue disponível no Anexo A e tal código estará também disponível para consulta no repositório final no GitHub.

### 3.1 Fluxograma da solução

A presente solução possui um fluxograma próprio. Ele é montado em um momento pré-desenvolvimento (*pré-dev*), ou seja, antes de começar qualquer desenvolvimento, a fim de que, ao ser criado, o repositório esteja de acordo com as estratégias definidas.

O fluxograma da Figura 9 representa a solução desejada quando o desenvolvedor realiza uma tarefa e, com auxílio do SonarLint, é garantida sua idoneidade, podendo então realizar uma *PR* da *branch* local para a de dev. Uma vez a *PR* feita, a pipeline realiza os testes necessários e, caso tudo ocorra conforme esperado, segue para a *branch* de (*Quality Assurance* – garantia da qualidade) QA, realizando também os testes. Essa atualização é enviada para *branch* de produção (*main*) e pode ser finalizada e ser disponibilizada aos consumidores via kubernetes.

**Figura 9 - Fluxograma da solução.**



Fonte: Elaboração própria.

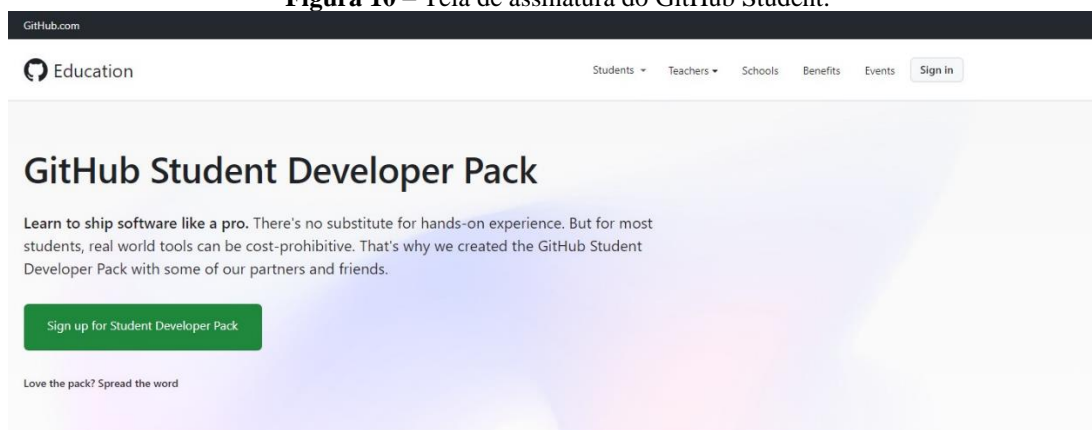


## 4 Implementação da Pipeline DevOps

### 4.1 Implementação do CI a partir da Azure Pipelines

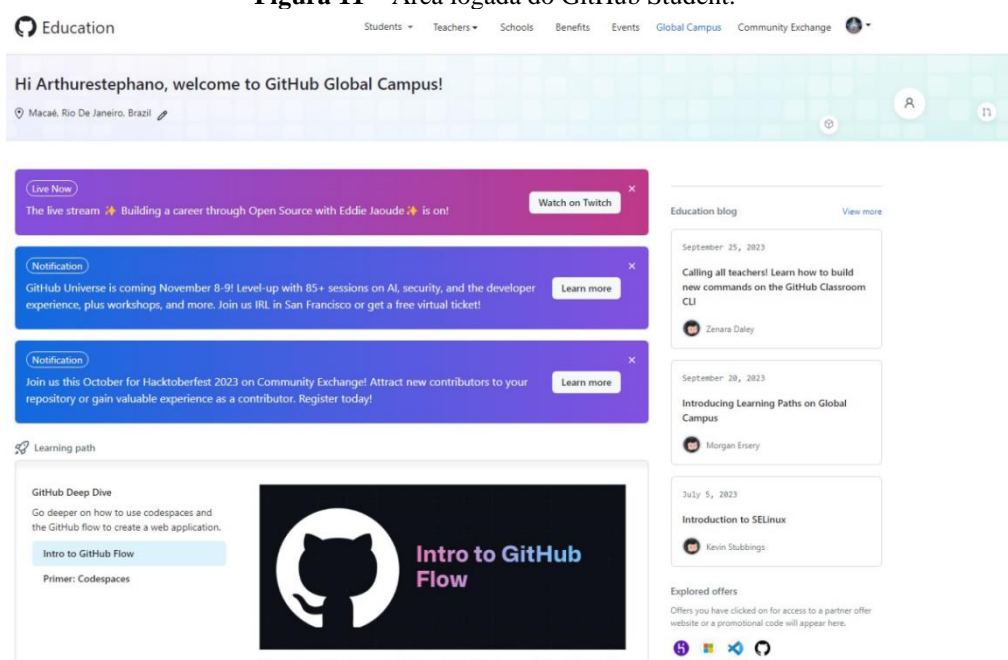
O presente trabalho começa com a assinatura do GitHub Student, programa de desenvolvimento para estudantes, professores e instituições de ensino possuírem acesso a ferramentas que normalmente são pagas. A assinatura é feita de uma maneira simples, enviando o horário do presente período ou o comprovante de matrícula em caso de estudante, como visto nas figuras 10 e 11.

**Figura 10** – Tela de assinatura do GitHub Student.



Fonte: GitHub Education<sup>11</sup>

**Figura 11** – Área logada do GitHub Student.

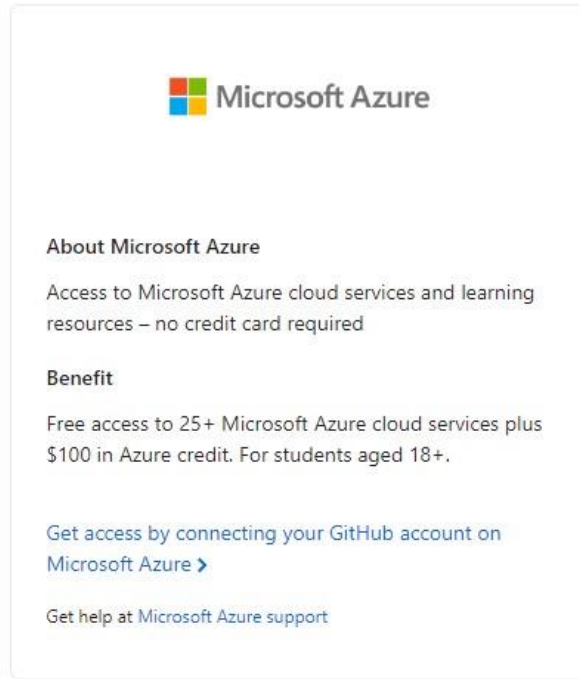


Fonte: Ambiente GitHub Education. Acesso em 13 out. 2023

<sup>11</sup> Disponível em: <https://education.github.com/pack?sort=company/>. Acesso em 13 out. 2023.

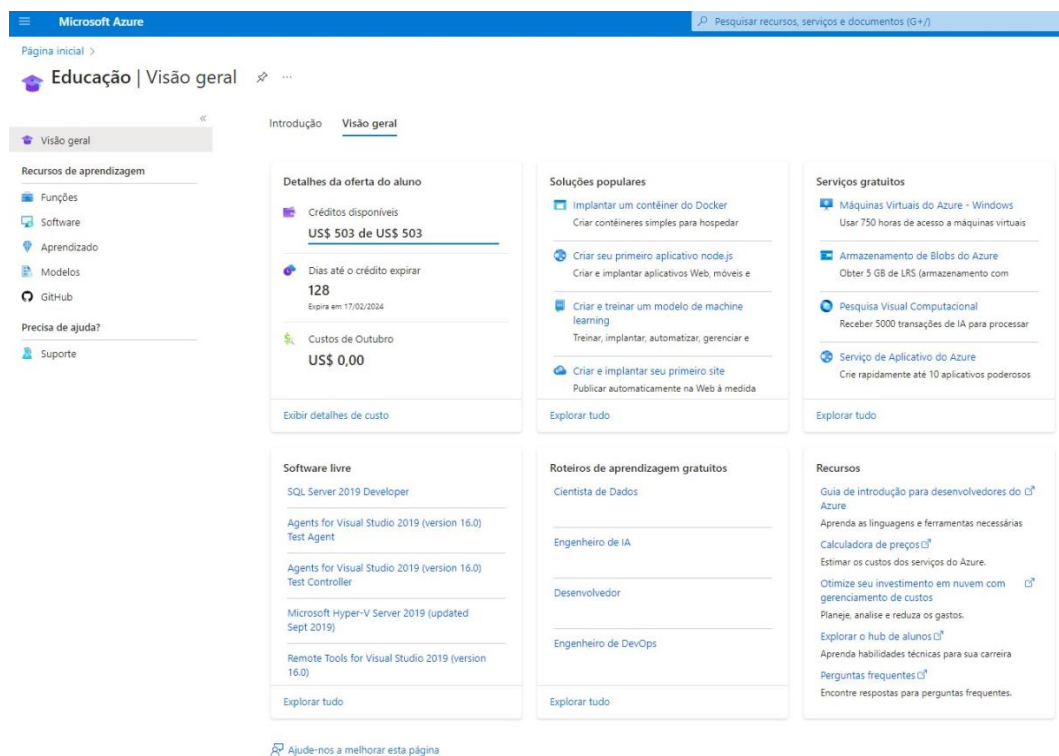
Uma vez no ambiente foi realizada a conexão com o Azure a partir da conta do GitHub, esta garante U\$100,00 em créditos na Azure para certos serviços como visto nas figuras 12 e 13.

**Figura 12** – Conexão com o Microsoft Azure.



Fonte: Ambiente GitHub Education. Acesso em 13 out. 2023

**Figura 13** – Área logada Microsoft Azure.



Ajude-nos a melhorar esta página

Fonte: Ambiente Azure. Acesso em 13 out. 2023

Uma vez no ambiente da Azure, procedeu-se para a criação de um cluster de Kubernetes, referente ao serviço de Kubernetes da Azure (AKS), que será o responsável pelo *deploy* do software utilizado na pipeline. Durante a criação do mesmo foi realizada também a criação da *virtual network* (vnet) e do serviço de registro de container (ACR), como se vê na Figura 14.

**Figura 14** – Criação cluster Kubernetes.

Microsoft Azure

Página inicial > Serviços do Kubernetes >

### Criar cluster do Kubernetes

Validação aprovada

Básico Pools de nós Rede Integrações Avançado Marcas Revisar + criar

**Básico**

Assinatura	Azure for Students
Grupo de recursos	arthur-femass
Região	East US
Nome do cluster do Kubernetes	aks-arthur-femass
Versão do Kubernetes	1.26.6
A atualização automática	Patch

**Pools de nós**

Pools de nós	1
Habilitar nós virtuais	Desabilitado

**Acesso**

Identidade do recurso	Identidade gerenciada atribuída pelo sistema
Contas locais	Habilitado
Autenticação e Autorização	Contas locais com Kubernetes RBAC
Tipo de criptografia	(Padrão) Criptografia em repouso com uma chave de criptografia gerenciada pela plataforma

**Rede**

Cluster particular	Desabilitado
Intervalos de IP autorizados	Desabilitado
Configuração da rede	Kubenet
Rede virtual	(Novo) arthur-femass-vnet
Sub-rede de cluster	(novo) default
Prefixo do nome DNS	aks-arthur-femass-dns

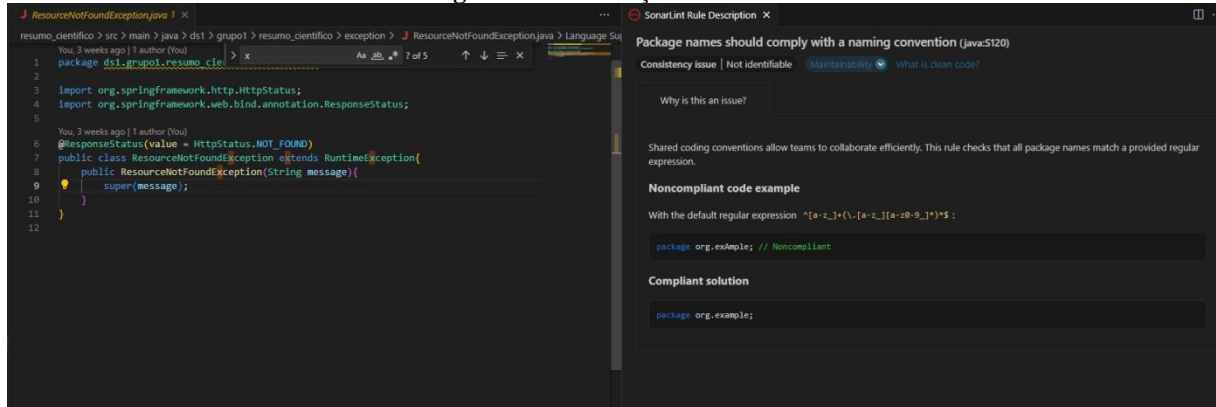
< Anterior Avançar > Criar

[Baixar um modelo para automação](#)

Fonte: Ambiente Azure. Acesso em 13 out. 2023

Com o kluster devidamente criado, foi desenvolvido e implementado as demais ferramentas necessárias para o escopo do trabalho, conforme apresentado no fluxograma da Figura 9. A próxima ferramenta empregada foi o SonarLint, serviço de Linter do SonarQube, trazendo boas práticas e análise de sintaxe e do código já gerado, conforme se observa na Figura 15.

Figura 15 – Utilização SonarLint.



Fonte: Elaboração própria.

O primeiro passo na criação de qualquer fluxo de CI/CD é propriamente o CI, uma vez que o seu código utiliza maven e Java. Foi criada uma pipeline para realização do processo de “package”, onde é criado o pacote .jar referente ao projeto java utilizado, também nessa pipeline são realizados os testes necessários, vide Figura 16. Ela foi feita via Azure pipelines, recurso do Azure DevOps.

Figura 16 – Pipeline Azure java maven.

← arthurEstephano.ds1\_femass\_grupoA

```
main ▼ arthurEstephano/ds1_femass_grupoA / azure-pipelines.yml *
```

```

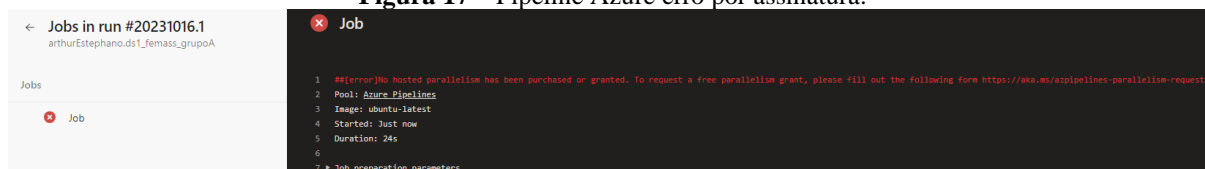
1 trigger:
2   - develop
3
4 pr:
5   - main
6   - qa
7
8
9 pool:
10  - vmImage: ubuntu-latest
11
12 steps:
13   Settings
14   - task: Maven@3
15     inputs:
16       mavenPomFile: 'pom.xml'
17       mavenOptions: '-Xmx3072m'
18       javaHomeOption: 'JDKVersion'
19       jdkVersionOption: '1.7'
20       jdkArchitectureOption: 'x64'
21       publishJUnitResults: true
22       testResultsFiles: '**/surefire-reports/TEST-*.xml'
23       goals: 'package'

```

Fonte: Elaboração própria.

Depois de realizada a operação, os resultados não saíram como o esperado, já que a licença de estudante do GitHub não permite utilização do DevOps para assinaturas que contêm o serviço de pipeline. Isso impossibilitou, portanto, a utilização desse recurso neste trabalho (vide Figura 17).

**Figura 17** – Pipeline Azure erro por assinatura.



Fonte: Ambiente Azure. Acesso em 13 out. 2023

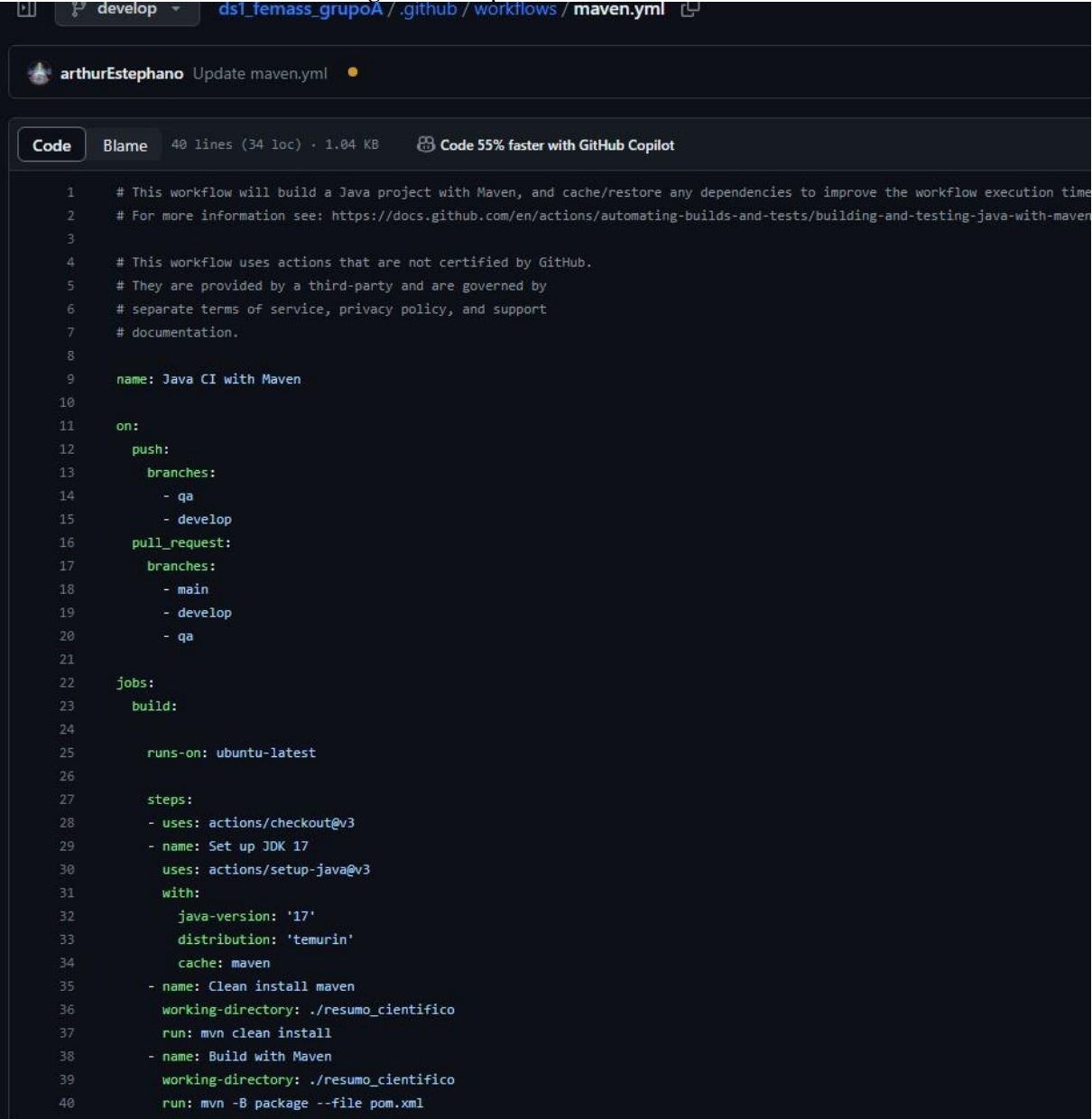
## 4.2 Implementação do CI a partir do GitHub Actions

A partir desse problema, a solução migrou para o serviço do próprio GitHub, ou GitHub Actions que permite a criação de fluxos de CI/CD tal qual o Azure Pipelines, porém utilizando os próprios recursos do GitHub.

Assim, foram reescritos os códigos gerados para pipeline, pois essa ferramenta possui a sintaxe ligeiramente diferente da Azure Pipelines. Foi criado um fluxo de CI para o projeto java utilizando o maven.

Esse fluxo acontece todas as vezes em que ocorrer uma PR nas *branches* main, qa e develop, visando manter o planejamento feito para solução. O mesmo fluxo, a partir de uma máquina Ubuntu, realiza um *checkout*, usa o java 17 com maven, realizando o processo de instalação e limpeza do código e, posteriormente, a realização dos testes e o processo de package, como pode ser visto na Figura 18.

Figura 18 – Pipeline GitHub Actions CI.



```

1  # This workflow will build a Java project with Maven, and cache/restore any dependencies to improve the workflow execution time
2  # For more information see: https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-maven
3
4  # This workflow uses actions that are not certified by GitHub.
5  # They are provided by a third-party and are governed by
6  # separate terms of service, privacy policy, and support
7  # documentation.
8
9  name: Java CI with Maven
10
11 on:
12   push:
13     branches:
14       - qa
15       - develop
16   pull_request:
17     branches:
18       - main
19       - develop
20       - qa
21
22 jobs:
23   build:
24
25     runs-on: ubuntu-latest
26
27     steps:
28       - uses: actions/checkout@v3
29       - name: Set up JDK 17
30         uses: actions/setup-java@v3
31         with:
32           java-version: '17'
33           distribution: 'temurin'
34           cache: maven
35       - name: Clean install maven
36         working-directory: ./resumo_cientifico
37         run: mvn clean install
38       - name: Build with Maven
39         working-directory: ./resumo_cientifico
40         run: mvn -B package --file pom.xml

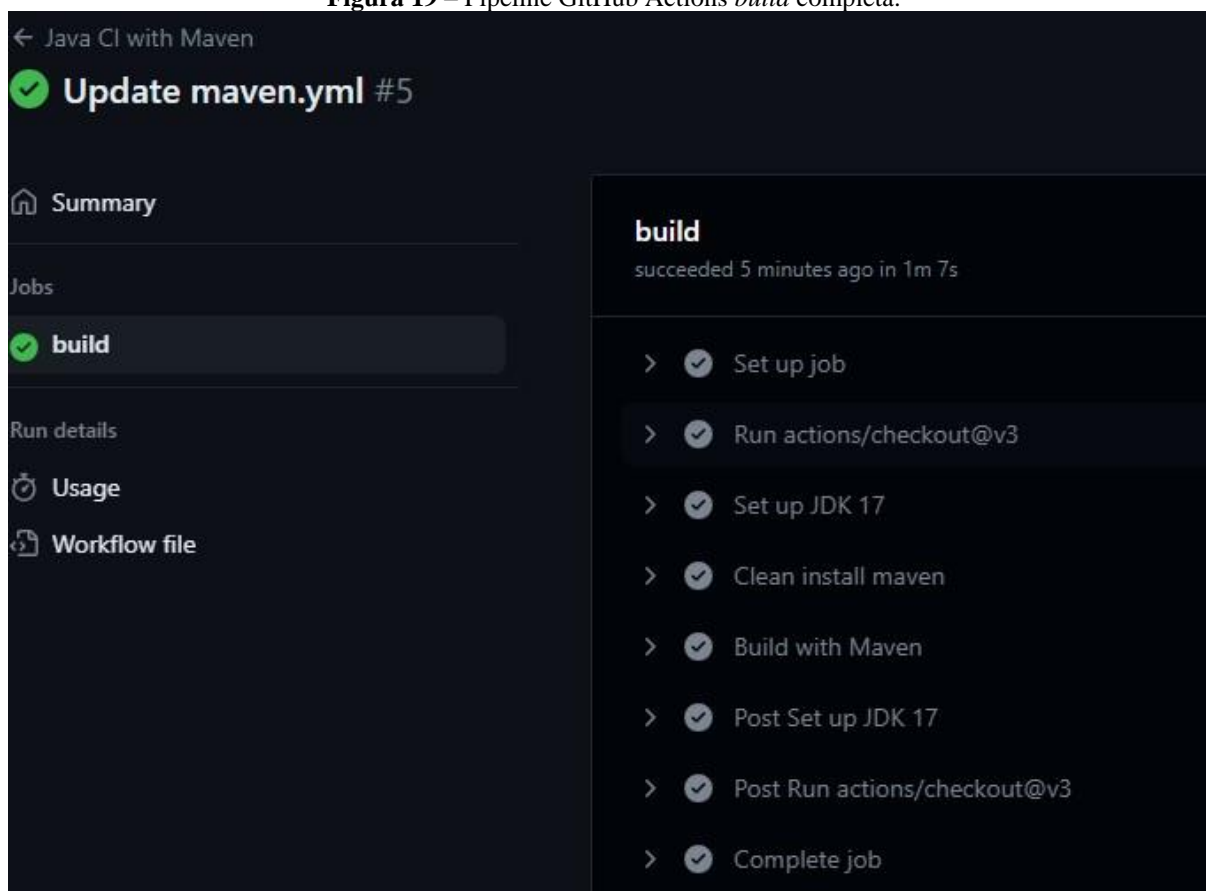
```

Fonte: Elaboração própria.

A pipeline mostrada acima foi executada após algumas tentativas de maneira correta, como visto na Figura 19, realizando então a limpeza, a instalação, a testagem e o empacotamento do projeto java em questão. É válido ressaltar que o nosso projeto só possui um teste de fato gerado (Figura 20), não permitindo a realização de testes de coberturas ou outras análises mais profundas conforme fora planejado nesse trabalho.

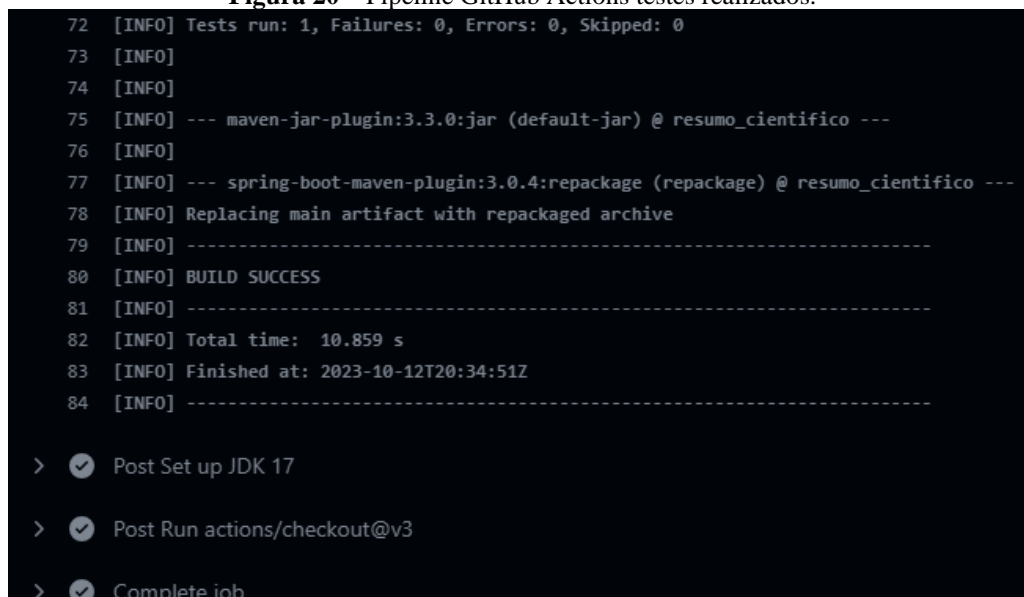
Essa primeira etapa (CI) começa pelo *checkout*, passando pela instalação e pela limpeza do código, pelo processo de *build* configurado pelos testes e empacotamento, e pelos passos pós-utilização do java e do *checkout*.

**Figura 19** – Pipeline GitHub Actions *build* completa.



Fonte: Elaboração própria.

**Figura 20** – Pipeline GitHub Actions testes realizados.

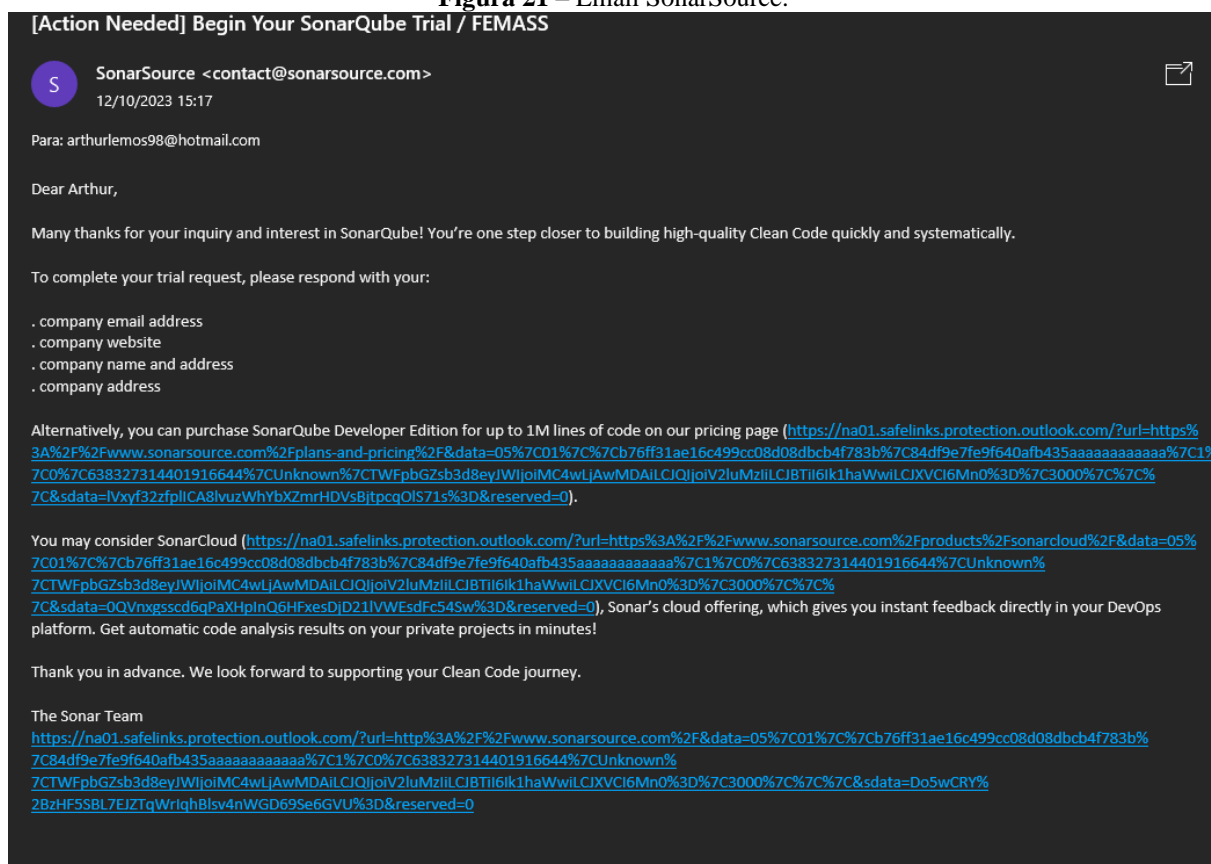


Fonte: Elaboração própria.

### 4.3 Implementação da Análise de código com SonarQube

Houve uma tentativa de integrar o SonarQube com a pipeline da Figura 18, porém tal ferramenta não permite utilização completa gratuitamente, nem mesmo para estudantes, permitindo só desenvolvedores de certas empresas a utilizarem-na e, consequentemente, impossibilitando a utilização neste trabalho de conclusão de curso. Foram feitas tentativas de contato com a empresa detentora do SonarQube, sem êxito, como visto na Figura 21.

**Figura 21** – Email SonarSource.



Fonte: E-mail pessoal do autor. Acessado em 13 out. 2023

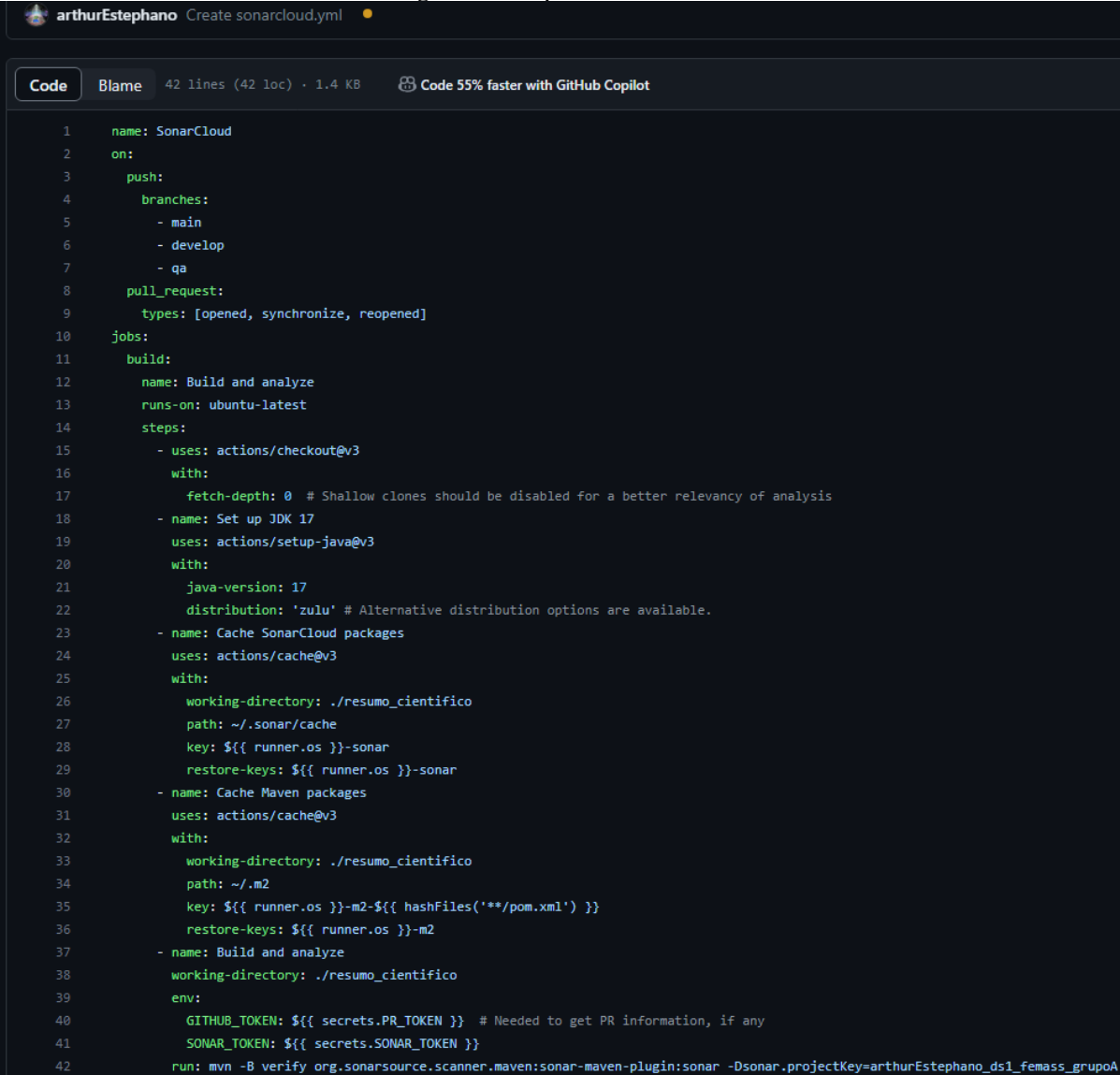
### 4.4 Implementação da Análise de código com SonarCloud

Para manutenção do presente projeto, foi necessário seguir com a utilização do SonarCloud, ferramenta em cloud da mesma empresa que realiza uma análise tão boa quanto a do SonarQube, porém não permite tantas configurações quanto esta ferramenta, como por exemplo, o percentual de testes de cobertura. Foi adicionado ao arquivo pom do projeto e criada então uma pipeline, visando separar os processos de análise (feitos de maneira assíncrona) dos processos de *build* e test.



Foi criada então uma outra pipeline no GitHub Actions, vide Figura 22, visando à realização da análise do código, que possui, como gatilho, tanto pushes diretos nas *branches* main, qa e dev quanto a abertura de pull requests (PRs). Foram adicionados aos secrets do repositório (Figura 23), tanto um token do github (PR\_TOKEN) visando permitir que o SonarCloud analisasse as PRs quanto um token do próprio SonarCloud, a fim de realizar a integração da pipeline com o projeto criado na ferramenta.

**Figura 22 – Pipeline SonarCloud.**



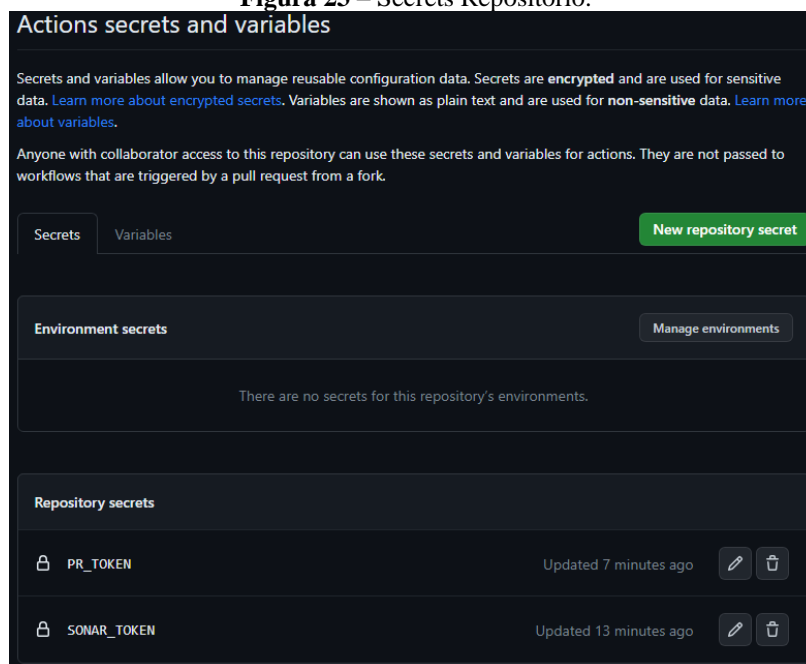
```

1  name: SonarCloud
2  on:
3    push:
4      branches:
5        - main
6        - develop
7        - qa
8    pull_request:
9      types: [opened, synchronize, reopened]
10 jobs:
11   build:
12     name: Build and analyze
13     runs-on: ubuntu-latest
14     steps:
15       - uses: actions/checkout@v3
16         with:
17           fetch-depth: 0 # Shallow clones should be disabled for a better relevancy of analysis
18       - name: Set up JDK 17
19         uses: actions/setup-java@v3
20         with:
21           java-version: 17
22           distribution: 'zulu' # Alternative distribution options are available.
23       - name: Cache SonarCloud packages
24         uses: actions/cache@v3
25         with:
26           working-directory: ./resumo_cientifico
27           path: ~/.sonar/cache
28           key: ${{ runner.os }}-sonar
29           restore-keys: ${{ runner.os }}-sonar
30       - name: Cache Maven packages
31         uses: actions/cache@v3
32         with:
33           working-directory: ./resumo_cientifico
34           path: ~/.m2
35           key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
36           restore-keys: ${{ runner.os }}-m2
37       - name: Build and analyze
38         working-directory: ./resumo_cientifico
39         env:
40           GITHUB_TOKEN: ${{ secrets.PR_TOKEN }} # Needed to get PR information, if any
41           SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
42         run: mvn -B verify org.sonarsource.scanner.maven:sonar-maven-plugin:sonar -Dsonar.projectKey=arthurEstephano_ds1_femass_grupoA

```

Fonte: Elaboração própria.

**Figura 23** – Secrets Repositório.



Fonte: Elaboração própria.

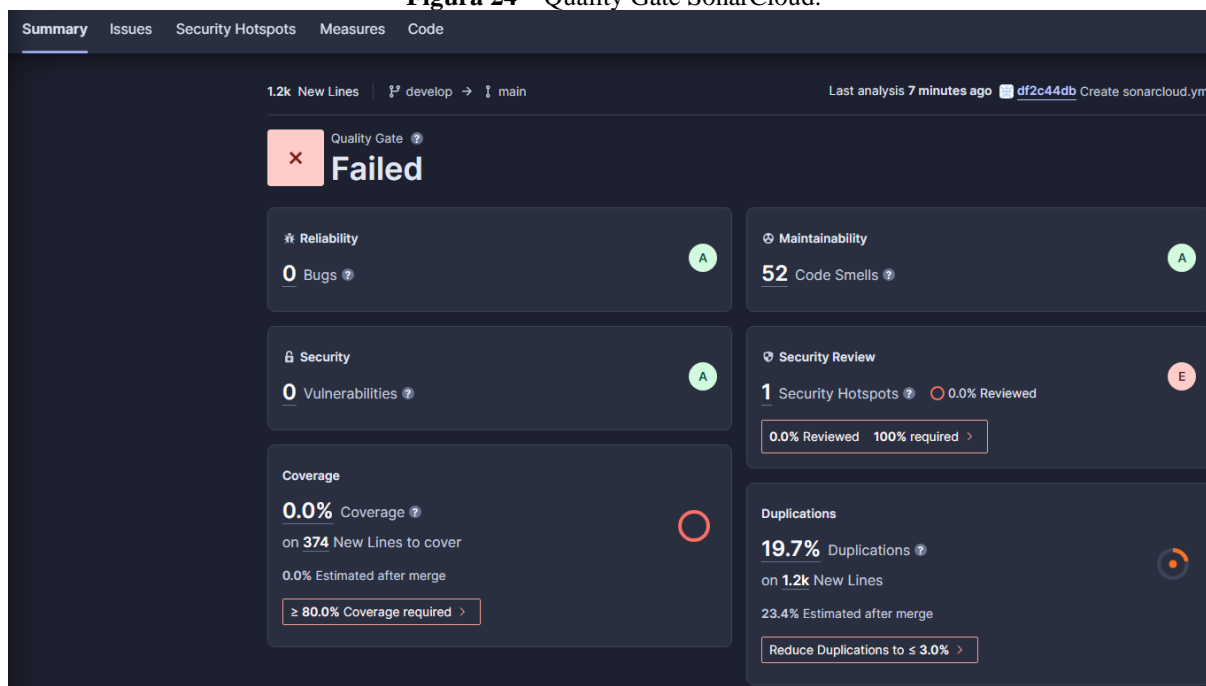
Após gerada a análise, é possível notar, como visto na Figura 24, que o projeto java utilizado para o presente trabalho não possui *bugs* e nem vulnerabilidades. Ele possui também uma boa taxa de manutenção, mas não oferece cobertura testada. Contém uma alta taxa de duplicações e revisões de segurança,

Caso tal análise fosse realizada em um contexto de um time ágil para entrega de um software para uma empresa ou pessoa física, tais problemáticas deveriam ser ajustadas antes da entrega em ambiente de produção. Todavia, como tal projeto é realizado para uma disciplina da faculdade, não há tanto empecilho na não utilização da qualidade mais alta na entrega final, pois não há possibilidade de customização das métricas de qualidade via SonarCloud.

Se tais pilares fossem abordados em disciplinas anteriores, os alunos poderiam estar cientes dos problemas que a ausência de qualidade no código pode gerar e de fato realizar entregas da mais alta excelência.

Tais pilares são parte de um contexto conhecido como *quality gate*, que deve garantir que o código gerado possua menos de 3% de duplicações em código (o que não necessariamente pode ser um problema), ausência de *bugs* e falhas de segurança, revisões de vulnerabilidade em todo o código e ao menos 80% de cobertura de código gerada. Esses pilares podem parecer triviais, porém garantem que o software entregue estará livre de riscos e escrito da melhor maneira possível, a fim de o produto ter a maior qualidade possível empregada.

Figura 24 – Quality Gate SonarCloud.

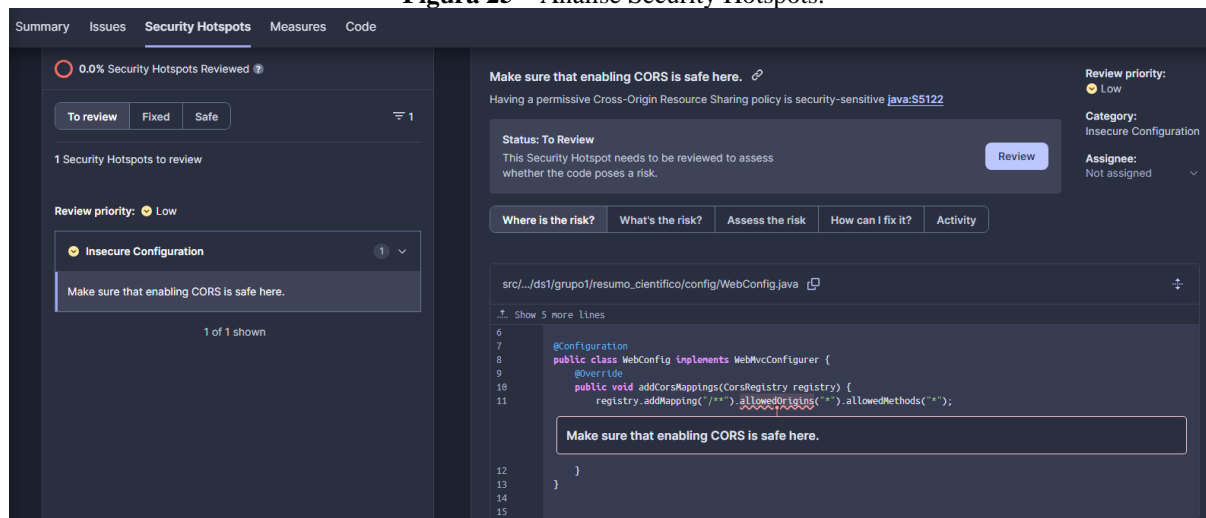


Fonte: Anexo C. Consultado: 16 out. 2023

Foi feita análise ponto a ponto dos pontos levantados pela ferramenta, começando pelos problemas de segurança: é visto pela ferramenta que o projeto possui uma configuração que pode gerar insegurança para o software entregue porque um hacker poderia, de maneira não autenticada, por exemplo, acesso remotamente ao servidor e explorá-lo como bem entender.

É possível resolver tal problema de modo simples e sem um esforço grande por parte do desenvolvedor responsável, como visto na Figura 25. Essa revisão seria de prioridade baixa, porém, uma vez implementada em ambiente produtivo, ela pode gerar ataques inesperados.

Figura 25 – Análise Security Hotspots.



Fonte: Anexo C. Consultado: 16 out. 2023

No tocante aos testes de cobertura, não há tanto a ser coberto, porém o fato de não haver testes é um indicativo que o projeto java não está devidamente testado, possuindo só 1 teste básico como visto anteriormente e deixando cerca de 374 linhas de código descobertas, como visto na Figura 26.

**Figura 26** – Análise Testes de Cobertura.

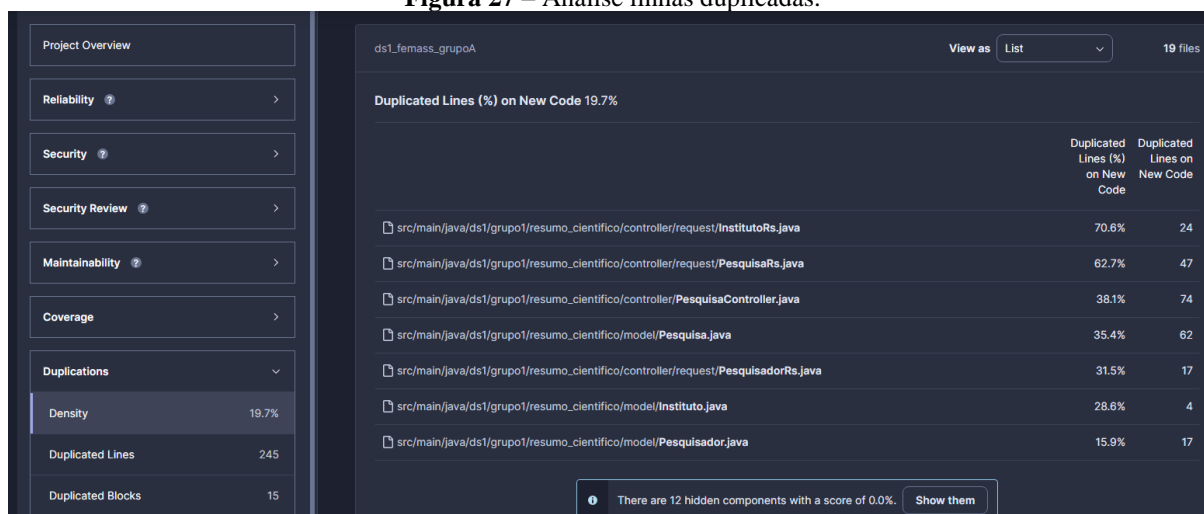
	Coverage on New Code	Uncovered Lines on New Code	Uncovered Conditions on New Code
src/main/java/ds1/grupo1/resumo_cientifico/controller/InstitutoController.java	0.0%	27	0
src/main/java/ds1/grupo1/resumo_cientifico/controller/request/InstitutoRs.java	0.0%	9	0
src/main/java/ds1/grupo1/resumo_cientifico/model/Pesquisa.java	0.0%	88	0
src/main/java/ds1/grupo1/resumo_cientifico/controller/PesquisaController.java	0.0%	109	0
src/main/java/ds1/grupo1/resumo_cientifico/model/Pesquisador.java	0.0%	42	0
src/main/java/ds1/grupo1/resumo_cientifico/controller/PesquisadorController.java	0.0%	52	0
src/main/java/ds1/grupo1/resumo_cientifico/controller/request/PesquisadorRs.java	0.0%	15	0
src/main/java/ds1/grupo1/resumo_cientifico/controller/request/PesquisaRs.java	0.0%	28	0
src/main/java/ds1/grupo1/resumo_cientifico/exception/ResourceNotFoundException.java	0.0%	2	0
src/main/java/ds1/grupo1/resumo_cientifico/config/WebConfig.java	0.0%	2	0

10 of 10 shown

Fonte: Anexo C. Consultado: 16 out. 2023

Em relação a duplicação do código, não necessariamente é um problema, visto que podem ser recursos reutilizados em diferentes lugares, porém ao utilizar java é possível abstrair classes e funções para não gerar duplicações e diminuir as linhas de código e consequentemente o tamanho do código, de acordo com a ferramenta utilizada cerca de 245 linhas e 15 blocos, em torno de 19,7% do código como um todo, como visto na Figura 27. O presente autor entende a partir de sua experiência que o ideal é reduzir a duplicação e tentar implementar funções ou classes, principalmente para os blocos duplicados.

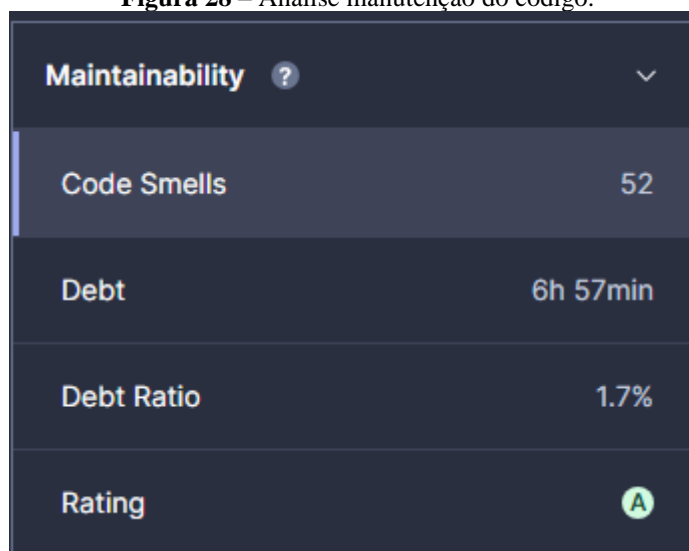
**Figura 27** – Análise linhas duplicadas.



Fonte: Anexo C. Consultado: 16 out. 2023

No tocante à manutenção do código, o maior problema encontrado pela ferramenta é justamente a ausência de testes de cobertura e alguns outros pequenos problemas. De acordo com esse recurso, tais alterações levariam em torno de 6 horas e 57 minutos para um desenvolvedor endereçar, como se observa na Figura 28, todavia a própria ferramenta não vê necessidade na correção imediata de todos esses pequenos problemas de manutenção, recomendando, portanto, o foco nas vulnerabilidades de segurança, criação dos testes de cobertura e análise, e possivelmente abstração de linhas duplicadas de código.

**Figura 28** – Análise manutenção do código.

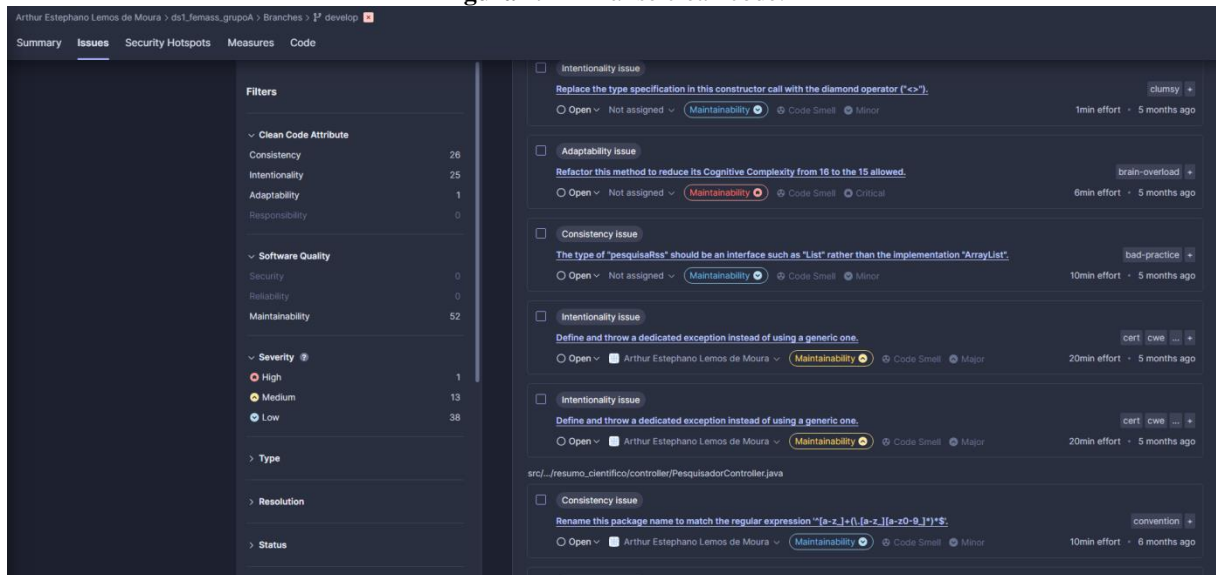


Fonte: Anexo C. Consultado: 16 out. 2023

A ferramenta ainda gera um relatório de boas práticas a partir do manifesto do *clean code* (código limpo), mostrando onde há problemas, qual seu grau de dificuldade, sua gravidade e quem é o responsável (ou deve ser) por resolver tais problemas (vide Figura 29). É válido

reforçar que alguns desses apontamentos são somente práticas não recomendadas, ao invés de problemas ou *bugs* de fato.

**Figura 29** – Análise clean code.



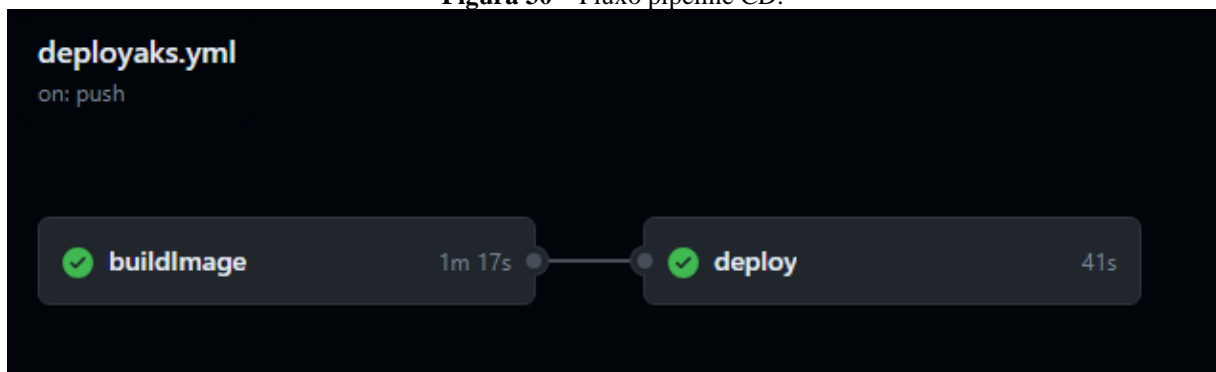
Fonte: Anexo C. Consultado: 16 out. 2023

É necessário frisar que o objetivo do presente projeto não é corrigir tais problemas no código e, sim, mostrar que o emprego das ferramentas certas, na utilização de pipelines, pode aumentar a qualidade do software gerado, enquanto emprega de maneira ágil todos os ritos necessários para sua disponibilização.

#### 4.5 Implementação do CD a partir do GitHub Actions

A etapa de implementação da pipeline de CD foi dividida em duas partes, como se verifica na Figura 30: uma relativa ao *build* do pacote (.jar) e da imagem de container que contém tal pacote, outra relativa ao processo de *deploy* no serviço de Kubernetes da Azure, o AKS.

**Figura 30** – Fluxo pipeline CD.



Fonte: Elaboração própria.

A pipeline como um todo possui partes relativamente parecidas com a de CI, visto que a limpeza e a instalação são feitas a fim de gerar o pacote (.jar). O gatilho para a utilização dessa pipeline é a atualização (push) de alguma das *branches* principais, isto é, qa, dev e main, sendo que ela pode ser acionada de maneira manual via APIs do GitHub. Além disso, ela utiliza algumas variáveis de ambientes, apontados na Figura 31:

- JAVA\_VERSION – versão do Java referente ao projeto.
- DISTRIBUTION – distribuidor do Java referente ao projeto.
- AZURE\_CONTAINER\_REGISTRY – nome do ACR relativo tanto ao registro e *build* de imagem de container quanto ao *deploy*.
- CONTAINER\_NAME – nome do container.
- RESOURCE\_GROUP – nome do grupo de recursos associados ao ACR e AKS.
- CLUSTER\_NAME – nome do cluster do AKS.
- DEPLOYMENT\_MANIFEST\_PATH – caminho para o arquivo manifesto utilizado para o *deploy* no AKS.

Figura 31 – Informações básicas pipeline CD.

```

1  name: Build and deploy JAR app to Azure Kubernetes service
2
3  env:
4    JAVA_VERSION: '17'
5    DISTRIBUTION: zulu
6    AZURE_CONTAINER_REGISTRY: "acr femass"
7    CONTAINER_NAME: "resumo_cientifico"
8    RESOURCE_GROUP: "arthur-femass"
9    CLUSTER_NAME: "acr-arthur-femass"
10   DEPLOYMENT_MANIFEST_PATH: "./resumo_cientifico/manifest.yml"
11
12  on:
13    push:
14      branches:
15        - main
16        - qa
17        - develop
18    workflow_dispatch:

```

Fonte: Elaboração própria.

A primeira etapa (job) dessa pipeline possui alguns passos, entre eles o *checkout* do git, a configuração do java e o processo de *build*, que já foram utilizados durante a pipeline de CI. Além desses passos para essa primeira etapa, também estão presentes os passos de login na Azure e *build* e registro de imagem de um container como visto nas figuras 32 e 33.

Figura 32 – buildImage job parte 1.

```

20 permissions:
21 | contents: read
22
23 jobs:
24 | buildImage:
25 |   permissions:
26 |     contents: read
27 |     id-token: write
28 |   runs-on: ubuntu-latest
29 |   steps:
30 |     - uses: actions/checkout@v3
31 |
32 |     - name: Set up Java version
33 |       uses: actions/setup-java@v3.0.0
34 |       with:
35 |         java-version: ${{ env.JAVA_VERSION }}
36 |         distribution: ${{ env.DISTRIBUTION }}
37 |         cache: 'maven'
38 |
39 |     - name: Build with Maven
40 |       working-directory: ./resumo_cientifico
41 |       run: mvn clean install
42

```

Fonte: Elaboração própria.

Figura 33 – buildImage job parte 2.

```

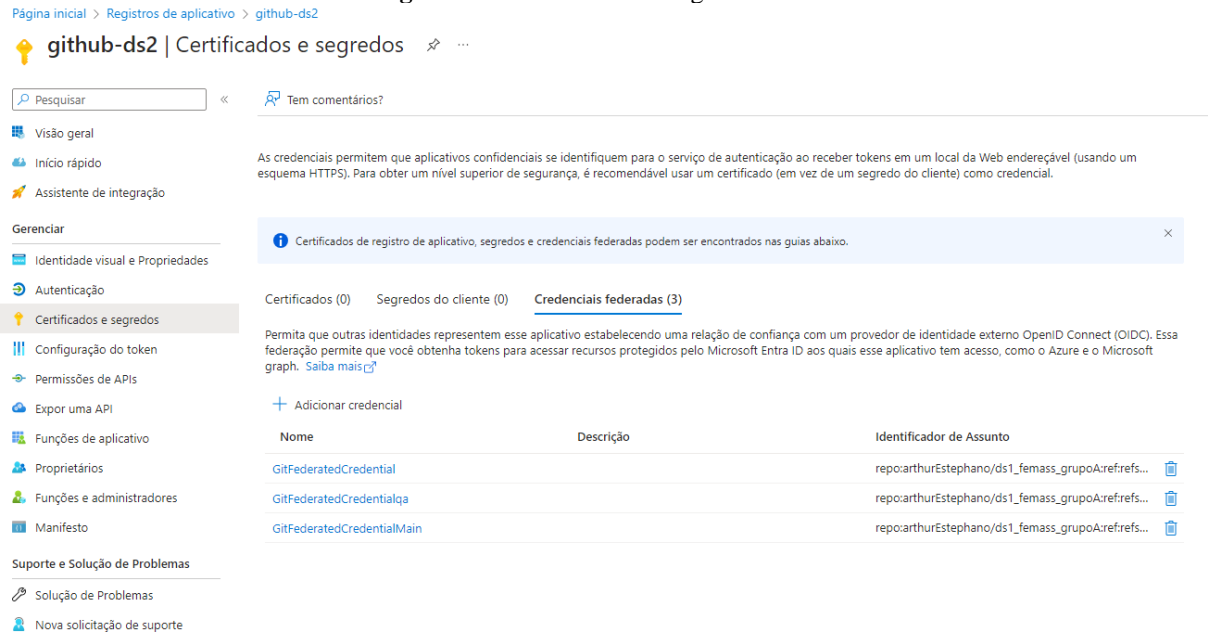
43 | - name: Azure login
44 |   uses: azure/login@v1.4.6
45 |   with:
46 |     client-id: ${{ secrets.AZURE_CLIENT_ID }}
47 |     tenant-id: ${{ secrets.AZURE_TENANT_ID }}
48 |     subscription-id: ${{ secrets.AZURE_SUBSCRIPTION_ID }}
49 |
50 | - name: Build and push image to ACR
51 |   working-directory: ./resumo_cientifico
52 |   run: |
53 |     az acr build --image ${{ env.AZURE_CONTAINER_REGISTRY }}.azurecr.io/${{ env.CONTAINER_NAME }}:${{ github.sha }} --registry ${{ env.AZURE_CONTAINER_REGISTRY }} -g ${{ env.RESOURCE_GROUP }} .

```

Fonte: Elaboração própria.

A utilização de secrets relativos ao login da Azure tem seu registro de maneira idêntica aos secrets utilizados na pipeline do SonarCloud para análise de código, porém, nesse caso, há necessidade da criação de um registro de aplicativo na interface da azure com certificados e segredos específicos para o github, além de funcionalidades e permissões de colaborador e/ou administrador (Figura 34).



**Figura 34 – Certificados e segredos Azure**

Fonte: Ambiente Azure. Acesso em 16 out. 2023

A *build* e registro de imagem utiliza comandos próprios da interface da Azure, no caso o az acr *build*, que determina o processo de *build* de uma imagem a partir do seu nome, que, com as variáveis, foi modificado para “acrfemass.azurecr.io/resumo\_cientifico”. O conteúdo, após os dois pontos, é referente à versão da imagem e foi utilizado o “github.sha”, que representa uma chave única para o evento que começou esse fluxo de pipelines.

A *build* só é possível por meio da utilização do arquivo Dockerfile (Figura 35), que é salvo na raiz do projeto java. Nesse caso o arquivo em questão utiliza uma imagem java (eclipse-temurin:17-jdk-alpine) para poder usar o comando de início da aplicação Java - o arquivo, utilizando a imagem, cria um container e copia o pacote (.jar) da pasta target para dentro dele com o nome “app.jar”, expondo em seguida a porta 8080 que é o padrão de aplicações e definindo os comandos iniciais do container, “java -jar /app.jar”, inicializando então a aplicação.

**Figura 35 – Dockerfile da solução.**

```
resumo_cientifico > Dockerfile > ...
...
1 FROM eclipse-temurin:17-jdk-alpine
2 COPY /target/resumo_cientifico-0.0.1-SNAPSHOT.jar app.jar
3 EXPOSE 8080
4 ENTRYPOINT [ "java", "-jar", "/app.jar" ]
```

Fonte: Elaboração própria.

O segundo job relacionado é o de *deploy*, que possui como pré-requisito o sucesso no job anterior. Ele também realiza um *checkout* e login na Azure, podendo então seguir com os passos de login não interativo com o Kubernetes, utilização do contexto de Kubernetes relacionados a conta Azure, login no ACR e finalizando com o próprio *deploy*, como visto na Figura 36.

Figura 36 – Deploy job.

```

55  deploy:
56    permissions:
57      actions: read
58      contents: read
59      id-token: write
60    runs-on: ubuntu-latest
61    needs: [buildImage]
62    steps:
63      - uses: actions/checkout@v3
64
65      - name: Azure login
66        uses: azure/login@v1.4.6
67        with:
68          client-id: ${ secrets.AZURE_CLIENT_ID }
69          tenant-id: ${ secrets.AZURE_TENANT_ID }
70          subscription-id: ${ secrets.AZURE_SUBSCRIPTION_ID }
71
72      - name: Set up kubelogin for non-interactive login
73        uses: azure/use-kubelogin@v1
74        with:
75          kubelogin-version: 'v0.0.25'
76
77      - name: Get K8s context
78        uses: azure/aks-set-context@v3
79        with:
80          resource-group: ${ env.RESOURCE_GROUP }
81          cluster-name: ${ env.CLUSTER_NAME }
82          admin: 'false'
83          use-kubelogin: 'true'
84
85      - name: ACR login
86        run: az acr login --name ${ env.AZURE_CONTAINER_REGISTRY }
87
88      - name: Deploys application
89        uses: Azure/k8s-deploy@v4
90        with:
91          action: deploy
92          manifests: ${ env.DEPLOYMENT_MANIFEST_PATH }
93          images: |
94            ${ env.AZURE_CONTAINER_REGISTRY }.azurecr.io/${ env.CONTAINER_NAME }:${ github.sha }

```

Fonte: Elaboração própria.

O passo de *deploy* utiliza, além do nome do container elaborado de maneira dinâmica pelas variáveis de ambiente e do github, uma propriedade chamada “manifests”, que são arquivos de manifesto em formato yml, e devem estar na raiz do projeto Java. São utilizados para configuração do *deploy* de um container em um serviço Kubernetes, utilizando somente um arquivo manifesto (Figura 37).

Figura 37 – Arquivo manifesto.

```

resumo_cientifico > ! manifest.yml > ...
io.k8s.api.core.v1.Service (v1@service.json) | io.k8s.api.apps.v1.Deployment (v1@deployment.json)
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: resumo-cientifico
5  spec:
6    replicas: 1
7    selector:
8      matchLabels:
9        app: resumo-cientifico
10   template:
11     metadata:
12       labels:
13         app: resumo-cientifico
14     spec:
15       nodeSelector:
16         "kubernetes.io/os": linux
17       containers:
18         - name: resumo-cientifico
19           image: acrfemass.azurecr.io/resumo_cientifico:latest
20           ports:
21             - containerPort: 8080
22   ---
23   apiVersion: v1
24   kind: Service
25   metadata:
26     name: resumo-cientifico
27   spec:
28     type: LoadBalancer
29     ports:
30       - port: 80
31         targetPort: 8080
32     selector:
33       app: resumo-cientifico

```

Fonte: Elaboração própria.

O arquivo em questão determina que versões da API do Kubernetes serão utilizadas e qual tipo. Nesse caso, foi utilizado um arquivo de aplicativos (apps/v1) do tipo *deployment* relacionado ao container criado por último, visto pela anotação “latest” na linha 19. Foi definido somente uma replica (pod) para o contexto do projeto utilizado a partir de um sistema operacional Linux. Além dele, foi criado um serviço relacionado ao *deployment* do projeto para servir de *LoadBalancer* e expor de maneira pública o IP relacionado ao projeto java.

O sucesso dos passos de *build* e registro do container e, de *deployment* desse para o AKS pode ser visto pelos logs do GitHub Action relativo às figuras 38 e 39, respectivamente.

**Figura 38** – Build e push da imagem para o ACR.

```

✓ Build and push image to ACR
74 c1835254adf7: Preparing
75 6c2ec132e8a1: Preparing
76 665a22563584: Preparing
77 924695645b42: Preparing
78 f314066c6934: Preparing
79 cc2447e1835a: Preparing
80 cc2447e1835a: Waiting
81 6c2ec132e8a1: Pushed
82 665a22563584: Pushed
83 c1835254adf7: Pushed
84 cc2447e1835a: Pushed
85 f314066c6934: Pushed
86 924695645b42: Pushed
87 6d017c511ea771e63803f5d8c20ddfbd27bb57f2: digest: sha256:80a6b93565958b488aaac2b07ad776402ee8e062832ec5b2ec78749755c7f438 size: 1579
88 2023/10/31 02:25:09 Successfully pushed image: acrfemass.azurecr.io/resumo_cientifico:6d017c511ea771e63803f5d8c20ddfbd27bb57f2
89 2023/10/31 02:25:09 Step ID: build marked as successful (elapsed time in seconds: 7.814381)
90 2023/10/31 02:25:09 Populating digests for step ID: build...
91 2023/10/31 02:25:09 Successfully populated digests for step ID: build
92 2023/10/31 02:25:09 Step ID: push marked as successful (elapsed time in seconds: 12.884455)
93 2023/10/31 02:25:09 The following dependencies were found:
94 2023/10/31 02:25:09
95 - image:
96   registry: acrfemass.azurecr.io
97   repository: resumo_cientifico
98   tag: 6d017c511ea771e63803f5d8c20ddfbd27bb57f2
99   digest: sha256:80a6b93565958b488aaac2b07ad776402ee8e062832ec5b2ec78749755c7f438
100 runtime-dependency:
101   registry: registry.hub.docker.com
102   repository: library/eclipse-temurin
103   tag: 17-jdk-alpine
104   digest: sha256:0f65d052c8ba399992199061c48ad3456d2ef33214a02c983d14fccb52b79b26c
105   git: {}
106
107
108 Run ID: caw was successful after 25s

```

Fonte: Elaboração própria.

**Figura 39** – Deploy da aplicação.

```

✓ Deploys application
1 ▶ Run Azure/k8s-deploy@v4
30 ▼ Deploying manifests
31 /usr/bin/kubectl apply -f /tmp/manifest.yml --namespace default
32 deployment.apps/resumo-cientifico configured
33 service/resumo-cientifico unchanged
34 ▼ Checking manifest stability
35 /usr/bin/kubectl rollout status Deployment/resumo-cientifico --namespace default
36 Waiting for deployment "resumo-cientifico" rollout to finish: 1 old replicas are pending termination...
37 Waiting for deployment "resumo-cientifico" rollout to finish: 1 old replicas are pending termination...
38 Waiting for deployment "resumo-cientifico" rollout to finish: 1 old replicas are pending termination...
39 deployment "resumo-cientifico" successfully rolled out
40 ▶ Printing ingresses
41 ▶ Annotating resources

```

Fonte: Elaboração própria.

## 4.6 Criação do Guia de Implementação e Desenvolvimento de Pipelines DevOps

Todos os arquivos gerados para realização deste trabalho estão públicos para consulta no repositório GitHub de seu autor e está presente no anexo B. Os arquivos mantidos estão como foram utilizados durante todo percurso acadêmico, porém no README do repositório há o guia de implementação e desenvolvimento que aborda cada uma das pipelines geradas, suas particularidades e seu detalhamento. O repositório também contém uma versão em pdf do trabalho (figuras 40 e 41).

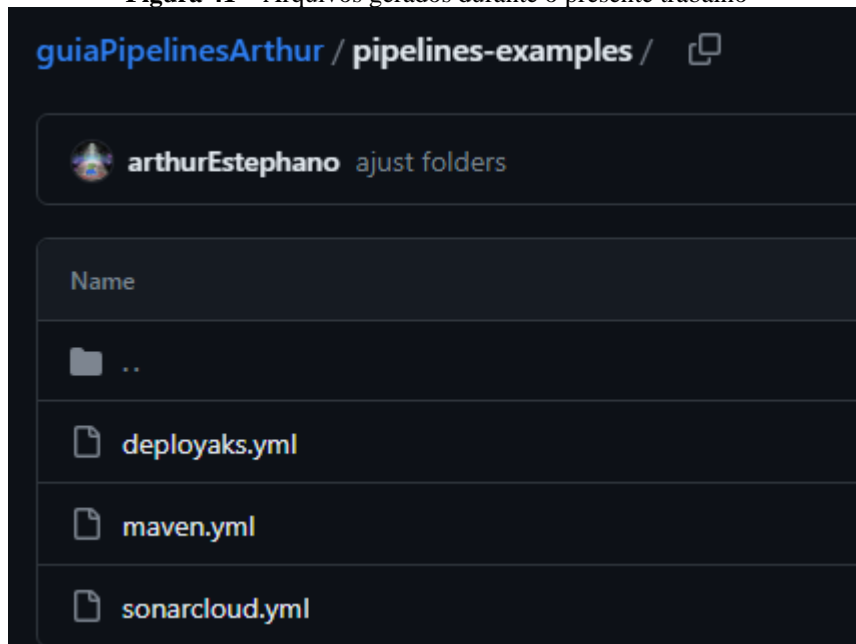
O presente guia será mantido pelo autor para consulta e melhorias pelos alunos da FeMASS, visando sempre fomentar o estudo e a adoção de novas tecnologias que possam melhorar a qualidade de um projeto de software gerado.

**Figura 40** – Guia de implementação e Desenvolvimento de Pipelines DevOps

The screenshot displays the GitHub interface for the repository 'guiaPipelinesArthur'. The main content area shows the README file, which is titled 'Guia para implementação e desenvolvimento de Pipelines DevOps'. The README text explains that the guide and repository were created to assist students of FeMASS in learning and using DevOps pipelines. It mentions that justifications for pipeline usage are provided in a PDF file within the repository. Below the introduction, there is a 'Sumário de Conteúdos' (Table of Contents) listing various topics such as 'Criando uma pipeline no GitHub Actions', 'Desenvolvimento de uma pipeline para teste e build de um projeto java', and 'Desenvolvendo uma pipeline para deploy de um projeto java em um cluster AKS'. The repository's sidebar on the right shows the 'About' section, which states the repository's purpose, and sections for 'Releases' and 'Packages', both of which are currently empty.

Fonte: Anexo B. Elaborado em 31 out. 2023

**Figura 41** – Arquivos gerados durante o presente trabalho



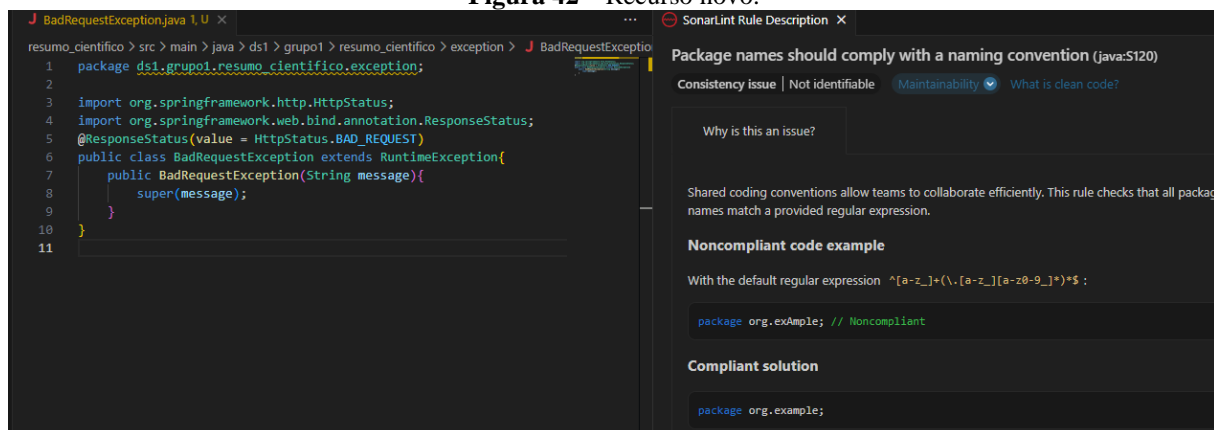
Fonte: Anexo B. Elaborado em 31 out. 2023

## 5 Utilização da Pipeline DevOps

Visando demonstrar a utilização das Pipelines DevOps em um fluxo completo o presente autor serviu de ator Desenvolvedor, este ator realiza alterações no código fonte a fim de implementar novos recursos no software.

Foi criada a partir da *branch* de desenvolvimento (develop) uma nova *branch*, chamada de “feature/dev-new”, esta conterá todo o trabalho do novo recurso do desenvolvedor. Neste caso o recurso novo é uma classe java que realiza a tratativa de exceção no caso de uma requisição errônea para o microsserviço associado a aplicação java. Ao desenvolvedor finalizar tal classe é notado pelo SonarLint uma quebra de regra como visto na Figura 42, este é referente ao nome do pacote do projeto não estar de acordo com as convenções de código, porém neste caso não há necessidade da correção imediata.

Figura 42 – Recurso novo.

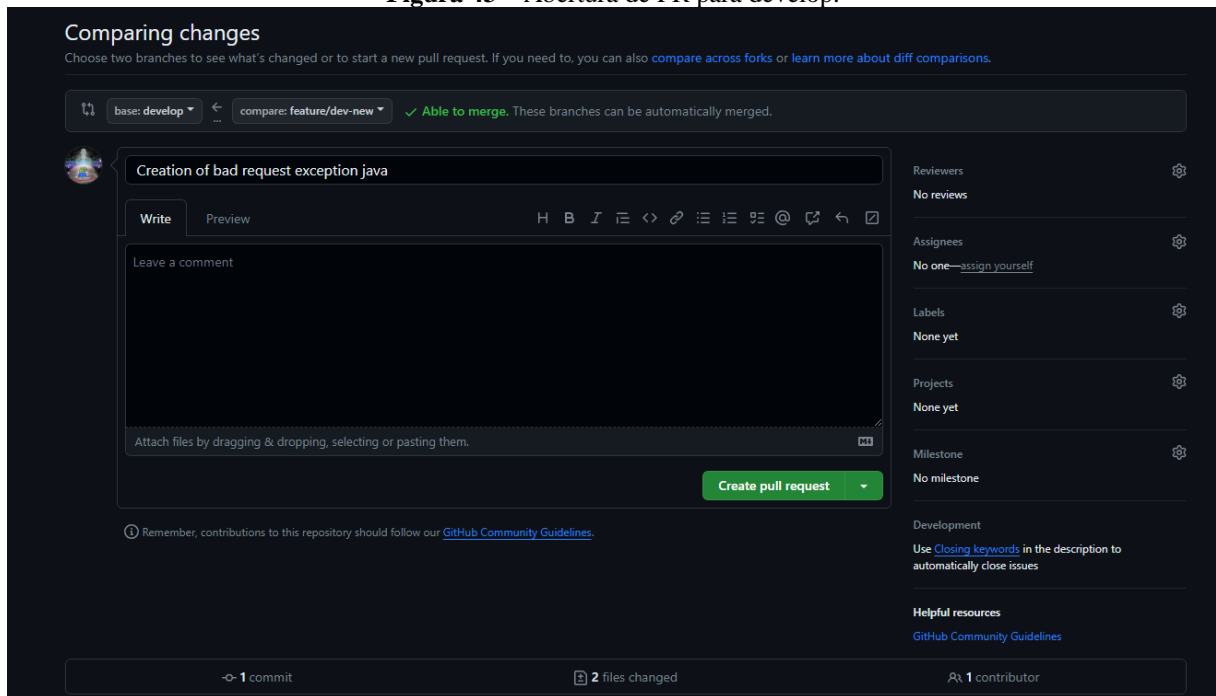


Fonte: Elaboração própria.

Uma vez o desenvolvedor estando satisfeito com a solução gerada e não vendo empecilhos maiores nos avisos do SonarLint (relativo ao momento A da Figura 9), ele prosseguirá com a publicação da *branch* criada para o repositório online e para a criação da PR para *branch* develop, como visto na Figura 43.

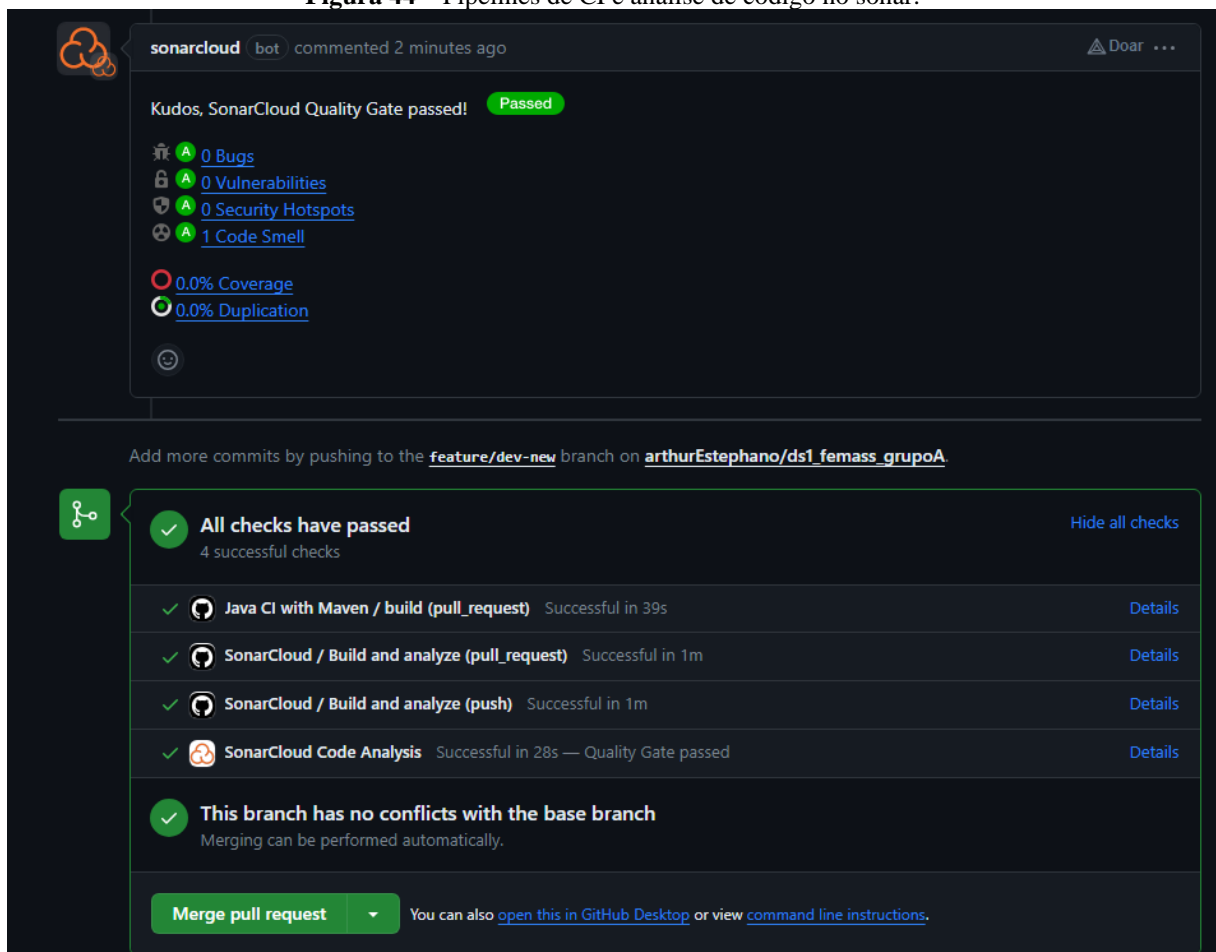
Com a PR criada (relativo ao momento B da Figura 9), as pipelines começam seus trabalhos: são chamadas para a rotina as pipelines de CI (relativo ao momento C da Figura 9) e análise de código, sendo a de análise com duas validações, uma para a atualização da *branch* nova criada e outra para a PR, verificando se há novos problemas como *bugs* e deixando explícito na própria conversa referente à PR, como se observa na Figura 44.

Figura 43 – Abertura de PR para develop.



Fonte: Elaboração própria.

Figura 44 – Pipelines de CI e análise de código no sonar.

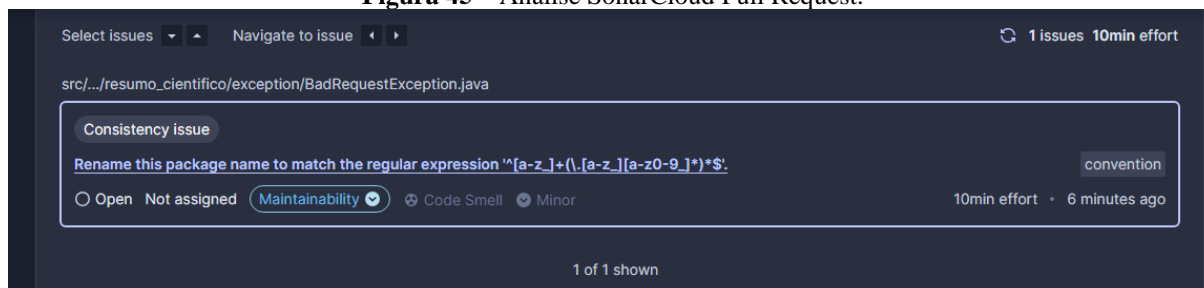


Fonte: Elaboração própria.



O único *code smell* apontado pela análise de código é o mesmo apontado pelo SonarLint e pode ser visto ao clicar em cima do seu texto ou abrindo diretamente a página web relativa ao projeto do SonarCloud como visto na Figura 45.

**Figura 45** – Análise SonarCloud Pull Request.



Fonte: Anexo C. Consultado: 16 out. 2023

Não havendo maiores problemas, o código foi aceito para *branch* develop, que engatilha as pipelines de CD e análise de código, as quais não tiveram erros, mostrando assim que o código pode seguir seu fluxo (Figura 46).

**Figura 46** – Sucesso pipelines CD e análise de código.

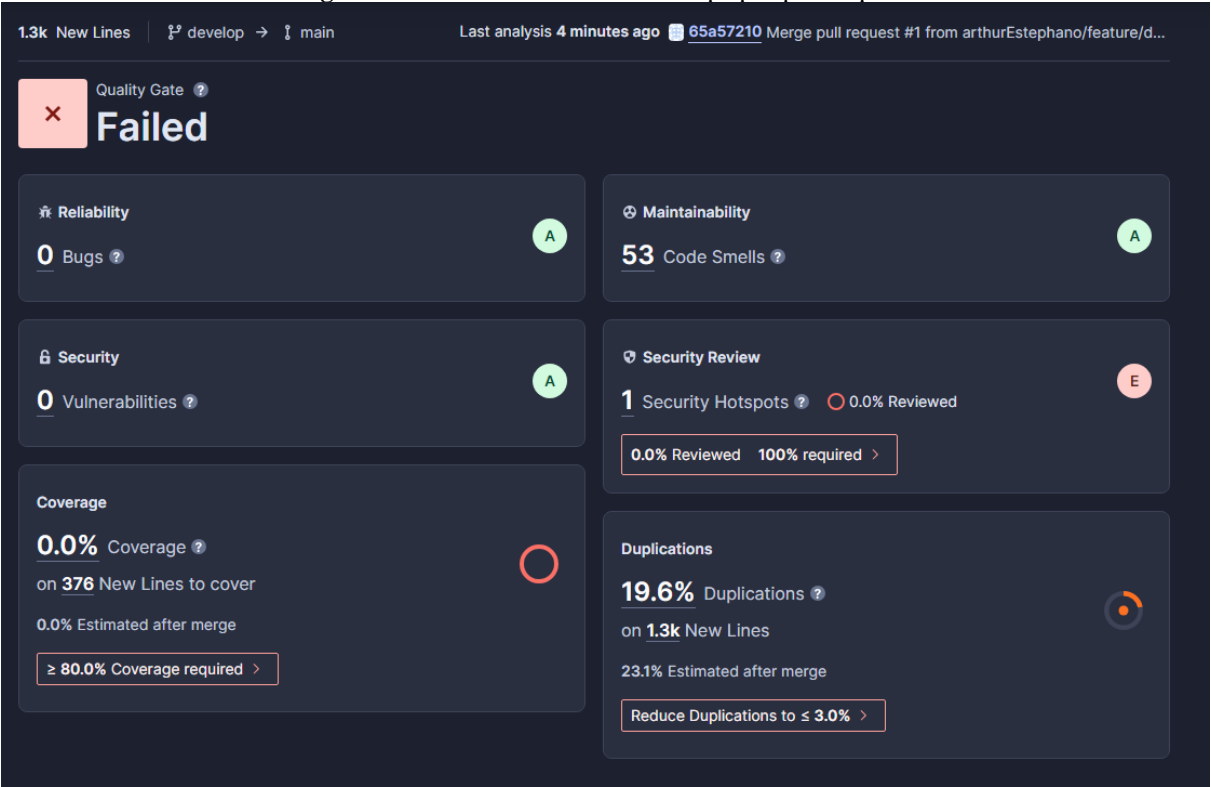


Fonte: Elaboração própria.

O sucesso de tais pipelines também podem ser visto em suas ferramentas integradas, no caso da pipeline de análise de código é visível no próprio projeto do SonarCloud navegando para *branch* develop como visto na Figura 47, onde a *branch* não passa nos padrões de qualidade da ferramenta, porém o código novo, entregue via PR, passa sem maiores problemas.

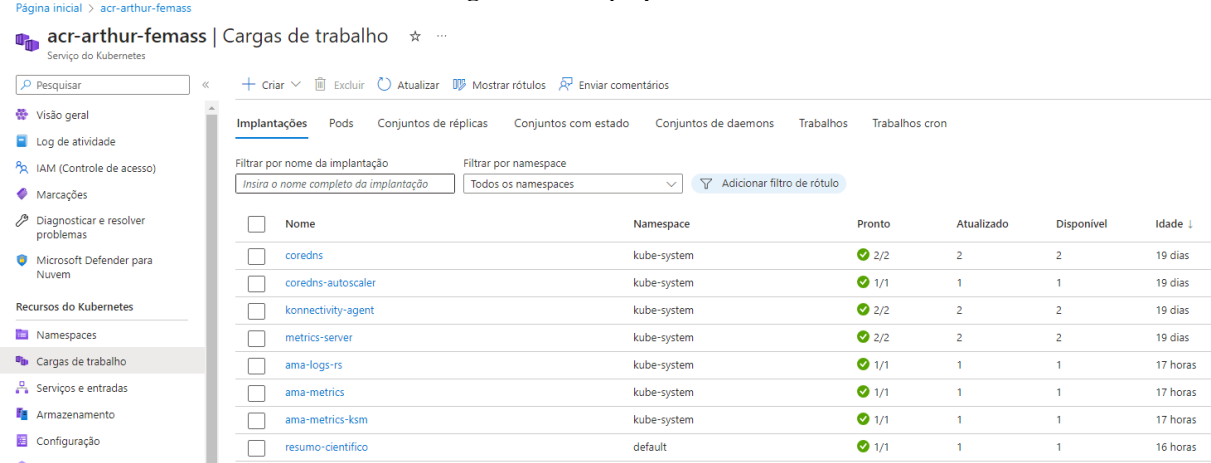
Para a pipeline de CD, o sucesso pode ser visto também nas ferramentas da Azure, mais especificamente no AKS (relativo aos momentos D, E e F da Figura 9), onde navegando até as cargas de trabalho é visível o pod referente ao *deploy* feito pela pipeline, nesse caso nomeado “resumo-cientifico” (Figura 48). Há possibilidade do teste, em ambiente de desenvolvimento via IP externo (Figura 49), também é possível validar via linha de comando na interface do AKS, buscando os pods primeiro e depois abrindo seus logs, como se observa na Figura 50.

Figura 47 – Análise da *branch* develop após pull request.



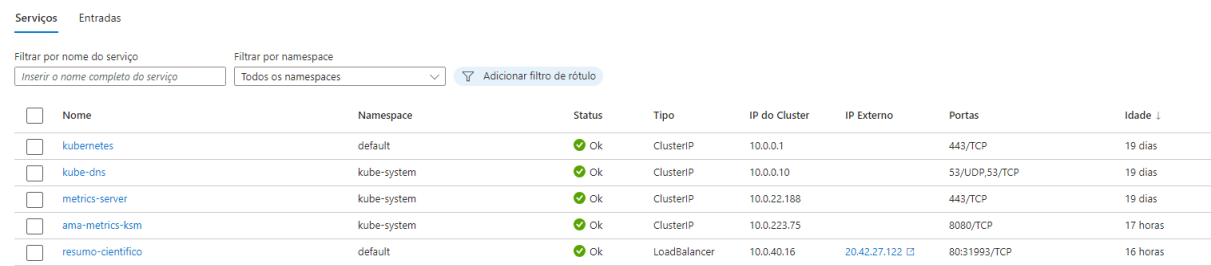
Fonte: Elaboração própria.

Figura 48 – *Deployment* feito no AKS.



Fonte: Ambiente Azure. Acesso em 16 out. 2023

Figura 49 – Consulta do IP externo do serviço.



Fonte: Ambiente Azure. Acesso em 16 out. 2023

**Figura 50** – Consulta de logs do pod.

```

1 arthurlemos9@hotmail.com> kubectl get pods
2 NAME          READY   STATUS    RESTARTS   AGE
3 resumo-cientifico-5bd68b4589-2bh24    1/1     Running   0           13m
4
5 arthurlemos9@hotmail.com> kubectl logs resumo-cientifico-5bd68b4589-2bh24
6
7
8
9
10
11
12
13 :: Spring Boot ::
14 (v3.0.4)
15 2023-10-31T16:53:43.763Z INFO 1 --- [main] d.g.r.ResumoCientificoApplication : Starting ResumoCientificoApplication v0.0.1-SNAPSHOT using Java 17.0.9 with PID 1 (/app.jar
16 started by root in /)
17 2023-10-31T16:53:43.771Z INFO 1 --- [main] d.g.r.ResumoCientificoApplication : No active profile set, falling back to 1 default profile: "default"
18 2023-10-31T16:53:45.780Z INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
19 2023-10-31T16:53:45.881Z INFO 1 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 89 ms. Found 3 JPA repository interfaces.
20 2023-10-31T16:53:46.785Z INFO 1 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
21 2023-10-31T16:53:46.798Z INFO 1 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
22 2023-10-31T16:53:46.798Z INFO 1 --- [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.5]
23 2023-10-31T16:53:46.914Z INFO 1 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
    main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2961 ms

```

Fonte: Elaboração própria.

A finalização do fluxo em questão se dá pela abertura de PR da *branch* develop para qa e posteriormente da qa para main, finalizando então o fluxo, porém dentro dessas próximas *branches* não há recursos nem validações novas feitas além das já demonstradas durante o presente capítulo.

Isso define a finalização da integração e da implementação das pipelines DevOps no projeto java referente à disciplina de Desenvolvimento de Sistemas II, assim como a disponibilização do guia no repositório especificado no Anexo B.

## CONSIDERAÇÕES FINAIS

Este trabalho explorou a criação de um guia de desenvolvimento, de implementação e de boas práticas de pipelines DevOps para os alunos da FeMASS. Para atingir esse fim, foram definidos objetivos que incluíram o desenvolvimento de uma pipeline DevOps, a documentação do código gerado, a implementação da pipeline gerada em um escopo de projeto de sistemas, a criação de um guia contemplando os demais objetivos e, a provisão desse guia para os alunos da FeMASS com as informações necessárias para implementação autônoma de uma pipeline.

No início do estudo de caso, foi fundamental o entendimento do DevOps e do CI/CD para garantir que as pipelines geradas e o seu fluxograma fossem apropriados para cada momento do ciclo de desenvolvimento de software.

Ao longo da implementação das pipelines, foi possível entender a dificuldade que muitos estudantes têm em utilizá-las nos seus projetos, uma vez que as ferramentas inicialmente pretendidas não eram gratuitas nem para estudantes. A ferramenta de análise de código também foi alterada devido à não gratuidade oferecida a estudantes, não permitindo, portanto, a alteração dos indicadores de qualidade.

Porém, a alteração do Azure Pipelines para o GitHub Actions trouxe uma compreensão maior do que de fato são pipelines e seu modelo de funcionamento, visto que a estrutura lógica independente do provedor seria a mesma, alterando somente a sintaxe empregada nos arquivos.

Foram utilizados serviços em nuvem na Azure, como ACR e AKS, principalmente na pipeline de CD, visando empregar, nesse contexto, tecnologias ainda ausentes nas disciplinas da FeMASS, possibilitando então o emprego delas não somente para utilização das pipelines, mas também para processos de *deploy* necessários em algumas disciplinas da instituição e no dia a dia de um profissional de Sistemas da Informação.

Os resultados obtidos com o emprego da pipeline de análise de código, que utiliza o SonarCloud, contribuíram para a compreensão e a aplicação de boas práticas de programação durante as etapas de desenvolvimento. Esta pipeline também alertou sobre a ausência de testes gerados durante a disciplina e sobre riscos de segurança empregados dentro do projeto.

Este trabalho contribuiu para a compreensão e a implementação das pipelines DevOps, gerando, ao final, um guia que documenta o passo a passo do desenvolvimento e da implementação delas para consulta dos alunos da FeMASS, enquanto não possuem uma disciplina que trata de tais tópicos. Todavia, existem algumas limitações atualmente e direções para futuras pesquisas. Uma possível melhoria seria a expansão dos tipos de projetos utilizados, como projeto em node, .net, Django, entre outros, para aumentar o horizonte de possibilidade

de funcionamento das pipelines. Também é possível o emprego de outros provedores de pipelines, como AWS, Azure, GoogleCloud, visando à diversificação de soluções.

Em suma, o presente trabalho evidenciou que as pipelines são ferramentas eficazes durante e após o ciclo de desenvolvimento, sendo útil para os alunos de Sistemas de Informação da FeMASS tanto durante sua graduação quanto após ela. Pesquisas futuras podem expandir o escopo de casos de uso de pipelines, habilitando mais testes e validações com ferramentas gratuitas, visando ainda mais ganhos para a qualidade do software gerado e diversificação das tecnologias hoje empregadas durante a graduação de Sistemas de Informação.

## REFERÊNCIAS

ATLASSIAN. **Atlassian Survey 2020 - DevOps Trends**. Disponível em: <https://www.atlassian.com/whitepapers/devops-survey-2020>. Acesso: 13 mar 2023

BASS, L.; WEBER, I.; & ZHU, L. **Devops: A Software Architect's Perspective**. Boston: Addison-Wesley Professional, 2015.

DOCKER. **Use containers to Build, Share and Run your applications**. Disponível em: <https://www.docker.com/resources/what-container/>. Acesso em: 11 jun. 2023.

DOCKER. **Kubernetes**. Disponível em: <<https://www.docker.com/products/kubernetes/>>. Acesso em: 12 jun. 2023.

GERALEXGR. **Azure DevOps best practices – jobs and stages**. Disponível em: <https://blog.geralexgr.com/azure/azure-devops-best-practices-jobs-and-stages>. Acesso em: 11 jun. 2023.

GIL, A. C. **Métodos e técnicas de pesquisa social**. 6. ed. São Paulo: Atlas, 2008.

GIL, A. C. **Como elaborar projetos de pesquisa**. 6. ed. São Paulo: Atlas, 2010.

GITHUB. **GitHub Actions**. Disponível em: <https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions>. Acesso em: 25 out. 2023.

GITHUB. **GitHub Education**. Disponível em: <https://education.github.com/pack?sort=company>. Acesso em: 13 out. 2023

MICROSOFT. **O que é o DevOps?** Disponível em: <https://azure.microsoft.com/pt-br/resources/cloud-computing-dictionary/what-is-devops>. Acesso em: 13 mar. 2023.

O'REILLY. **What is DevOps**. Disponível em: <https://www.oreilly.com/library/view/the-phoenix-project/9781457191350/42-resourceWhy.xhtml>. Acesso em: 02 mai. 2023.

PEREIRA NETO, Francisco. **DevOps: Uma Introdução**. eBook Kindle, 2022a.

PEREIRA NETO, Francisco. **DevOps: CI e CD**. eBook Kindle, 2022b.

PEREIRA NETO, Francisco. **DevOps: Além do CI e CD - Parte I**. eBook Kindle, 2022c.

SONARQUBE. **SonarQube Documentation**. Disponível em: <https://docs.sonarqube.org/latest/>. Acesso em: 11 jun. 2023.

SONARQUBE. **SonarQube Documentation**. Disponível em: <https://docs.sonarsource.com/sonarqube/latest/user-guide/quality-gates> Acesso em: 25 out. 2023.

## ANEXO A – Ementa disciplina de Desenvolvimento de Sistemas II FeMASS



FACULDADE PROFESSOR MIGUEL ÂNGELO DA SILVA SANTOS-FEMASS

### 8º PERÍODO DESENVOLVIMENTO DE SISTEMAS II

**DISCIPLINA: DESENVOLVIMENTO DE SISTEMAS II**  
**Créditos: 04**

**Carga Horária: 60h**

#### EMENTA:

Desenvolvimento de software usando a metodologia Ágil. Desenvolvimento Mobile e Uso de Web Services.

#### BIBLIOGRAFIA BÁSICA:

BEZERRA, Eduardo. **Princípios de análise e projeto de sistemas com UML**. Rio de Janeiro: Campus, 2006.

PRESSMAN, Roger S. **Engenharia de software**. 6. ed. Brasil: McGraw-Hill Interamericana, 2006.

FLANAGAN, D. **JavaScript: O guia definitivo**. 6ª ed. O'Reilly, 2011.

DUCKETT, J. **JavaScript e JQuery: Desenvolvimento de Interfaces web Interativa**. Alta Books, 2016.

GRANT, W. **UX Design: Guia Definitivo com as Melhores Práticas de UX**. 1ª Ed. São Paulo: Packet Novatec, 2019.

#### BIBLIOGRAFIA COMPLEMENTAR:

CARDOSO, Caíque. **UML na prática: do problema ao sistema**. Rio de Janeiro: Ciência Moderna, 2003.

DATE, C.J. **Introdução a sistemas de banco de dados**. 8. ed. Rio de Janeiro: Campus, 2004.

DEITEL, Harvey M.; DEITEL, PAUL J. **Java como programar**. 8. ed. São Paulo: Editora Prentice Hall, 2010.

LARMAN, C. **Utilizando UML e padrões**. 3ª ed. Porto Alegre: Bookman, 2008.

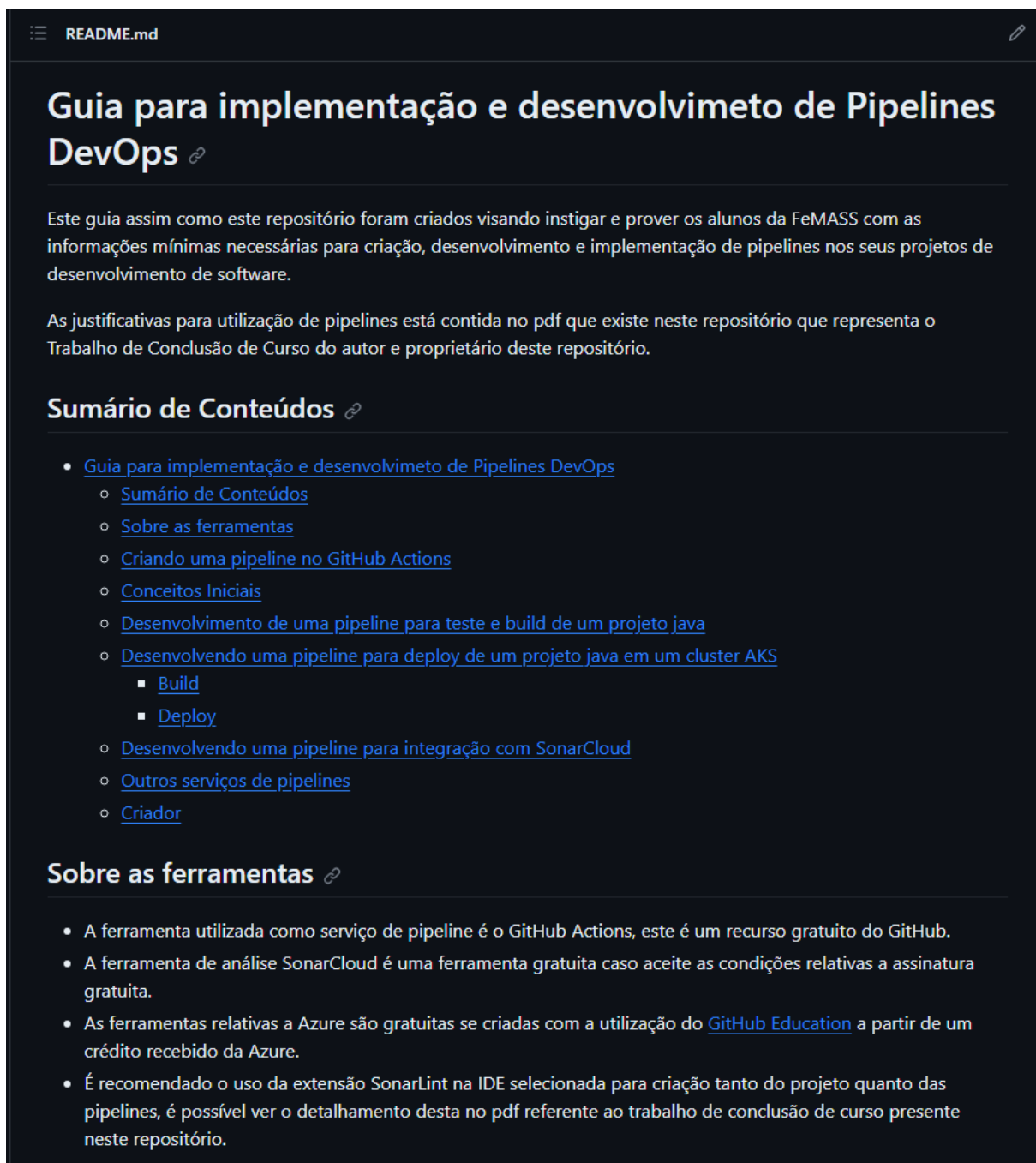
MENDES, Antônio. **Arquitetura de Software: desenvolvimento baseado na arquitetura**. Rio de Janeiro: Campus, 2002.

SILVEIRA, Paulo [et al.]. **Introdução à arquitetura e design de software: uma visão sobre a plataforma Java**. Rio de Janeiro: Elsevier, 2012.

Fonte: Ementário do Curso de Sistemas de Informação da FeMASS, p. 47.<sup>12</sup>

<sup>12</sup> Disponível em: <https://www.macaee.rj.gov.br/midia/conteudo/arquivos/1631371724.pdf/>. Acesso em: 31 out. 2023.

## ANEXO B – Guia de Implementação e Desenvolvimento de Pipelines DevOps



The image is a screenshot of a GitHub repository's README file. The title is 'Guia para implementação e desenvolvimento de Pipelines DevOps'. The text explains that the guide and repository were created to assist students of FeMASS in creating, developing, and implementing pipelines. It mentions that justifications for pipeline usage are in a PDF in the repository. The 'Sumário de Conteúdos' (Table of Contents) lists several topics: the guide itself, tools, creating a pipeline on GitHub Actions, initial concepts, developing a pipeline for testing and building a Java project, developing a pipeline for deploying a Java project on an AKS cluster (with sub-points for Build and Deploy), developing a pipeline for integration with SonarCloud, other pipeline services, and the creator. The 'Sobre as ferramentas' (About the tools) section states that GitHub Actions is a free GitHub service, SonarCloud is free with conditions, Azure tools are free with GitHub Education, and SonarLint is recommended for IDEs.

README.md

# Guia para implementação e desenvolvimento de Pipelines DevOps

Este guia assim como este repositório foram criados visando instigar e prover os alunos da FeMASS com as informações mínimas necessárias para criação, desenvolvimento e implementação de pipelines nos seus projetos de desenvolvimento de software.

As justificativas para utilização de pipelines está contida no pdf que existe neste repositório que representa o Trabalho de Conclusão de Curso do autor e proprietário deste repositório.

## Sumário de Conteúdos

- [Guia para implementação e desenvolvimento de Pipelines DevOps](#)
  - [Sumário de Conteúdos](#)
  - [Sobre as ferramentas](#)
  - [Criando uma pipeline no GitHub Actions](#)
  - [Conceitos Iniciais](#)
  - [Desenvolvimento de uma pipeline para teste e build de um projeto java](#)
  - [Desenvolvendo uma pipeline para deploy de um projeto java em um cluster AKS](#)
    - [Build](#)
    - [Deploy](#)
  - [Desenvolvendo uma pipeline para integração com SonarCloud](#)
  - [Outros serviços de pipelines](#)
  - [Criador](#)

## Sobre as ferramentas

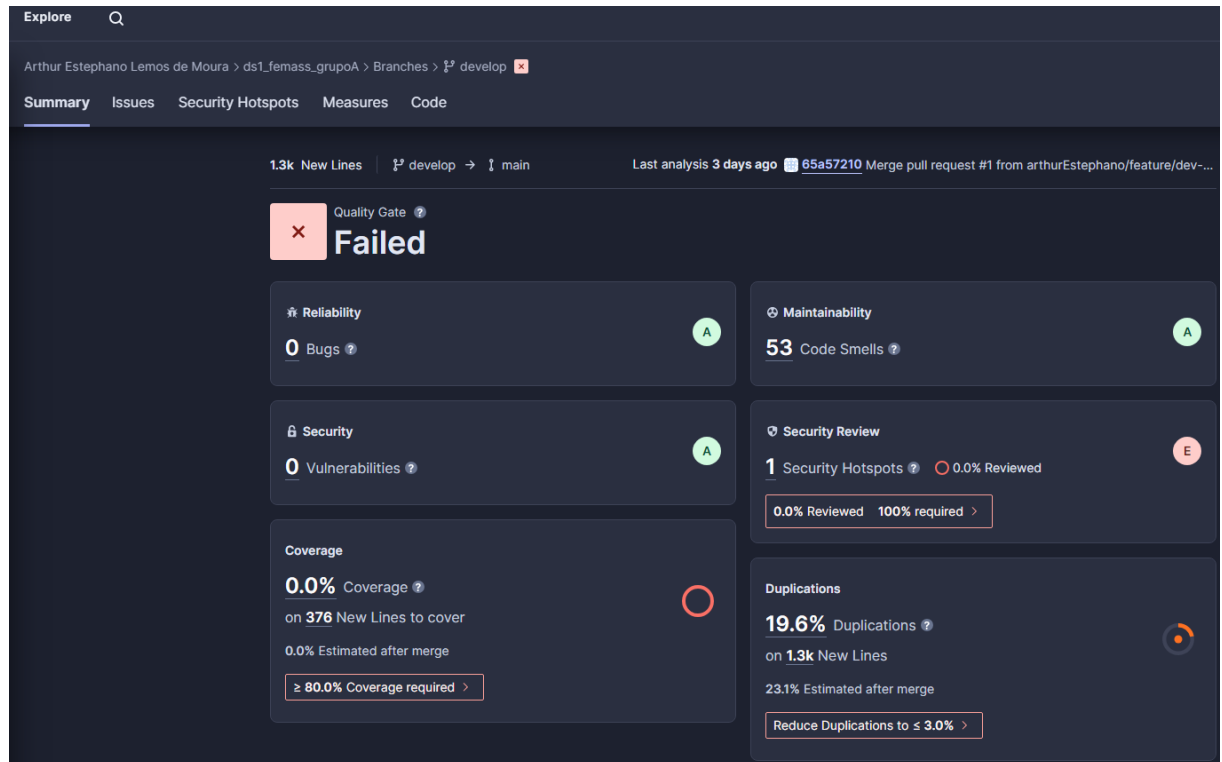
- A ferramenta utilizada como serviço de pipeline é o GitHub Actions, este é um recurso gratuito do GitHub.
- A ferramenta de análise SonarCloud é uma ferramenta gratuita caso aceite as condições relativas a assinatura gratuita.
- As ferramentas relativas a Azure são gratuitas se criadas com a utilização do [GitHub Education](#) a partir de um crédito recebido da Azure.
- É recomendado o uso da extensão SonarLint na IDE selecionada para criação tanto do projeto quanto das pipelines, é possível ver o detalhamento desta no pdf referente ao trabalho de conclusão de curso presente neste repositório.

Fonte: Repositório GitHub para etapa propositiva.<sup>13</sup>

<sup>13</sup> Disponível em: <https://github.com/arthurEstephano/guiaPipelinesArthur/>. Acesso em: 31 out. 2023.



## ANEXO C – Análise SonarCloud



Fonte: Análise SonarCloud para projeto java da disciplina de Desenvolvimento de Sistemas II.<sup>14</sup>

<sup>14</sup> Disponível em:  
[https://sonarcloud.io/summary/new\\_code?id=arthurEstephano\\_ds1\\_femass\\_grupoA&branch=develop/](https://sonarcloud.io/summary/new_code?id=arthurEstephano_ds1_femass_grupoA&branch=develop/).  
 Acesso em: 31 out. 2023.

## ANEXO D – Declaração de correção gramatical



ESTADO DO RIO DE JANEIRO  
PREFEITURA MUNICIPAL DE MACAÉ  
SECRETARIA DE EDUCAÇÃO  
SECRETARIA ADJUNTA DE ENSINO SUPERIOR  
FACULDADE PROFESSOR MIGUEL ÂNGELO DA SILVA SANTOS - FeMASS



**Macaé**  
PREFEITURA  
SECRETARIA ADJUNTA DE ENSINO SUPERIOR

Recredenciamento - Parecer CEE-RJ nº 172 de 26/05/2015, publicado no D.O./RJ nº 103, seção 1, pág. 12 de 15/06/2015

## DECLARAÇÃO DE CORREÇÃO GRAMATICAL

Eu, LUIZ CLAUDIO MACHADO DE SANTANA, portador (a) da carteira de identidade 04613742-8, formado (a) em Letras (Português/Literatura) pela UFRJ, realizei a correção gramatical do Trabalho de Conclusão de Curso do (a) aluno (a) Arthur Estephano Lemos de Moura, curso Sistemas de Informação, orientado (a) pelo (a) professor (a) Alan Carvalho Galante.

Professor Avaliador

Luiz Claudio Machado de Santana

Data: 13 / de novembro / 2023.