



**GARANTIA DA QUALIDADE DE SOFTWARE COM DEVOPS**

**SOFTWARE QUALITY ASSURANCE WITH DEVOPS**

**GARANTÍA DE CALIDAD DEL SOFTWARE CON DEVOPS**

Victor Hugo Pizzaia<sup>1</sup>, Rodrigo Daniel Malara<sup>2</sup>

e3112193

<https://doi.org/10.47820/recima21.v3i11.2193>

PUBLICADO: 11/2022

**RESUMO**

Nos dias atuais, os processos de *software* têm se tornado cada vez mais comuns, após grandes eventos de globalização como a pandemia da COVID-19, muitas empresas vieram para a era da tecnologia e esse crescente teve um pico de demanda por soluções, trazendo a necessidade de passar os processos que eram feitos de forma manual e física para o mundo virtual. Porém, com o volume de solicitações das indústrias, os processos sofreram a necessidade de serem entregues sempre o mais rápido possível e, desta forma, poderiam acabar acarretando muitas falhas humanas por conta de prazo e pressão. O projeto presente neste trabalho tem como objetivo auxiliar na velocidade das entregas, automatizando processos repetitivos e garantindo a qualidade necessária em questão. Foram analisadas metodologias que se encaixam nessas necessidades, auxiliando a criação de um fluxo completo com tecnologias novas no mercado. Como resultado do projeto foi criada uma sequência de automações que testam o *software* de ponta a ponta.

**PALAVRAS-CHAVE:** DevOps. Pipeline. Testes. Docker. Automação.

**ABSTRACT**

*Nowadays, software processes have become increasingly common, after major globalization events such as the COVID-19 pandemic, many companies came to the age of technology and this growth had a peak demand for solutions, bringing the need to pass the processes that were done manually and physically to the virtual world. However, with the volume of requests from industries, the processes suffered the need to always be delivered as quickly as possible and, this way, they could end up causing many human failures due to deadlines and pressure. The present project has the objective of helping in the speed of deliveries, automating repetitive processes and guaranteeing the necessary quality in question. Methodologies that fit these needs were analyzed, helping to create a complete flow with new technologies in the market. As a result of the project it was created a sequence of automations that test the software from end to end.*

**KEYWORDS:** DevOps. Pipeline. Tests. Docker. Automation.

**RESUMEN**

*Hoy en día, los procesos de software se han vuelto cada vez más comunes, después de grandes eventos de globalización como la pandemia de COVID-19, muchas empresas llegaron a la era de la tecnología y este crecimiento tuvo un pico de demanda de soluciones, trayendo la necesidad de pasar los procesos que se hacían manualmente y físicamente al mundo virtual. Sin embargo, con el volumen de solicitudes de las industrias, los procesos sufrieron la necesidad de ser entregados siempre lo más rápido posible y, de esta manera, podrían acabar provocando muchos fallos humanos debido al plazo y la presión. El proyecto presente en este trabajo pretende ayudar en la rapidez de las entregas, automatizando los procesos repetitivos y garantizando la calidad necesaria en cuestión. Se analizaron las metodologías que se ajustaban a estas necesidades, ayudando a crear un flujo*

<sup>1</sup> Universidade de Araraquara - Uniara

<sup>2</sup> Graduação em Engenharia de Computação pela Universidade Federal de São Carlos UFSCar e mestrado em Física Computacional e Sistemas Distribuídos pela Universidade de São Paulo USP. Docente dos cursos de Engenharia de Computação e Sistemas de Informação da Universidade de Araraquara. Projetos internacionais na Hewlett Packard, Western Union, Nortel, Fexco e governo da Irlanda.



## RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR ISSN 2675-6218

GARANTIA DA QUALIDADE DE SOFTWARE COM DEVOPS  
Victor Hugo Pizzera, Rodrigo Daniel Malara

*completo con las nuevas tecnologías del mercado. Como resultado del proyecto se creó una secuencia de automatizaciones que prueban el software de principio a fin.*

**PALABRAS CLAVE:** DevOps. Tuberías. Pruebas. Docker. Automatización.

### INTRODUÇÃO

O mercado de tecnologia vem enfrentando grandes desafios, com o alto volume de transação de dados e múltiplos usuários conectados nos sistemas. Dessa forma, as empresas de *softwares* precisam fazer com que suas ferramentas tenham disponibilidade e velocidade, necessitando empregar técnicas e metodologias para a garantia do produto. Com grandes inovações e a área em crescimento, surgiu uma cultura chamada DevOps que, segundo Len Bass (2015, p. 26), “é uma resposta ao problema de entregas lentas. Quanto mais tempo demorar uma entrega para chegar ao mercado, menos vantagem será acumulada quaisquer recursos ou melhorias de qualidade que levaram a entrega”.

Segundo Sommerville (2011, p. 38), atualmente as empresas se veem na necessidade de uma adaptação ágil, precisando responder a novas oportunidades e novos mercados, assim trazendo diversas técnicas de desenvolvimento. Para o cliente é importante que o *software* tenha disponibilidade e que ele supra as necessidades exigidas no projeto. A disponibilidade é algo que não combina com código mal escrito, podendo gerar alguma falha ou *bug* como apontado por Martin (2008, p. 3) “foi o código ruim que acabou com a empresa”.

O DevOps pode ser um grande aliado para combater estes problemas. Ao empregar metodologias ágeis e ao mesmo tempo garantir qualidade ao código fonte. Como dito pela AWS (2022), ele consiste em uma combinação de filosofias culturais, práticas e ferramentas que aumentam a capacidade das empresas de distribuir seus *softwares* e serviços com alta velocidade.

Visto que muitas empresas buscam aperfeiçoar seus processos, alguns trabalhos manuais e repetitivos podem prejudicar as suas entregas, com a interação humana estes processos podem acabar acarretando uma falha. É possível garantir a qualidade automatizando esses processos tais como fazer a revisão de código, executar os testes manualmente, coletar o código principal e colocar em produção. Com diferentes ferramentas de integração e entrega contínua, com o DevOps são criadas automações de verificação e validação de código, como também executam os testes unitários, testes de API e testes de integração, e com estes dados é possível criar métricas e bloqueios por níveis de qualidade.

Para Sommerville (2011, p. 19), em empresas com uma baixa diversificação de *softwares*, pode-se padronizar processos possibilitando uma comunicação mais clara entre a equipe, diminuindo a demanda de treinamentos e tornando o processo automatizado mais viável. A padronização introduz novas metodologias, técnicas de engenharia de software e boas práticas.

O objetivo deste trabalho é automatizar, testar e validar os processos do ciclo de vida do *software* utilizando as práticas do DevOps, tais como: a integração contínua e a entrega contínua, a



## RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR ISSN 2675-6218

GARANTIA DA QUALIDADE DE SOFTWARE COM DEVOPS  
Victor Hugo Pizzera, Rodrigo Daniel Malara

fim de garantir a qualidade das entregas. Foram realizados testes unitários, testes de integração e testes de API utilizando JUnit e algumas ferramentas para auxiliar, SonarQube para a verificação e validação de código e o GitHub Actions como ferramenta de automação de código, permitindo mitigar alguns problemas como: indisponibilidade, manutenibilidade, custo e segurança.

O processo de desenvolvimento de *software* não é totalmente automatizado, acarretando ineficiência na aplicação, trazendo diversos problemas como: a indisponibilidade da aplicação, a baixa manutenibilidade e aumento do custo de desenvolvimento.

Foi realizada a revisão bibliográfica das ferramentas utilizadas para a configuração do ambiente e execução dos processos que foram automatizados. Foram construídas duas *pipelines*, que segundo a RedHat (2022) é uma série de etapas a serem realizadas para a disponibilização de uma nova versão de um *software* bem como executado todos os testes, a validação da qualidade pelo Sonar e foi gerado uma nova versão do código e a construção dele para a produção. Por fim, foram disponibilizados os resultados dos processos realizados, conclusão e as referências.

### 1 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os conceitos das principais ferramentas e metodologias utilizadas para garantir a qualidade desde o desenvolvimento até a execução das automações, testes e validações de código.

#### QUALIDADE DE SOFTWARE

Como descrito em Sommerville (2011, p. 456) avaliar a qualidade de um software é um processo muito subjetivo, onde a equipe de qualidade deve julgar se o nível de qualidade é aceitável. Ela precisa validar se o *software* atingiu as expectativas e chegou na sua finalidade. Basicamente há a necessidade de atingir os requisitos dos usuários, existem aspectos da qualidade avaliada que não são necessariamente ligadas aos requisitos de usuário e sim das próprias métricas, como descrito por Len Bass (2015, p. 23) “qualidade significa adequação para uso por várias partes interessadas, incluindo usuários finais, desenvolvedores e administradores de sistema”.

#### DEVOPS

Conforme descrito por Huttermann (2012, p. 4) DevOps é uma junção dos desenvolvedores (incluindo programadores, testadores e o pessoal de garantia de qualidade) e o time de operações (especialistas que realizam a entrega do *software* em produção e gerenciam a infraestrutura). DevOps descreve algumas práticas para agilizar o processo de entrega de *software*, sempre trazendo a validação da produção para os desenvolvedores e melhorando o tempo dos processos do ciclo. O DevOps proporciona uma entrega de *software* mais rápida e ajuda a produzir *softwares* com maior qualidade, alinhados com os requisitos individuais e as condições básicas.



## INTEGRAÇÃO CONTÍNUA

De acordo com Duvall (2007, p. 24) integração contínua (do inglês, *continuous integration*) é a junção das técnicas dos desenvolvedores que dão a capacidade de alterar o código e saber se ele irá funcionar em um *feedback* imediato, gerando tempo para podermos corrigir.

A cada alteração do código, é executado os testes, no caso unitário e de integração, também é realizado a validação de código e outros processos para garantir a funcionalidade do software, onde após a verificação é gerado uma nova versão e disponibilizada para ser entregue a produção.

## ENTREGA CONTÍNUA

Segundo Duvall (2007, p. 190) entrega contínua (do inglês, *continuous delivery*) é uma ascensão de práticas e etapas que nos permitem entregar o *software* funcional a qualquer instante, de qualquer lugar, com o mínimo de esforço possível.

Após a varredura do código e caso aprovados as alterações, é realizado de forma automática a disponibilização da nova versão do *software* para o ambiente de produção, onde a aplicação está preparada para ser utilizada pelo cliente.

## SONARQUBE

O Sonar, conforme dito pela SonarQube é uma ferramenta de validação automática de código que serve para identificar *bugs*, vulnerabilidades e código que possam ocasionar problemas futuros (do inglês, *code smells*). É possível integrar ele no fluxo de trabalho para fazer uma varredura contínua de código em todas as ramificações do projeto e solicitações de alteração. Segundo Gaudin (2013, p. 33) “O SonarQube é uma plataforma de código aberto e gratuita para pesquisa e análise da qualidade de código. Ela fornece a você uma visão instantânea da qualidade do seu código hoje”.

Com a análise estática de código se conseguiu criar métricas para validação e verificar se o código está bom o suficiente para prosseguir, fazendo a utilização de alguns conceitos do DevOps conseguimos integrar essas métricas em nossa esteira de validações e fazer um portão de qualidade onde se reprovado o código voltará para o desenvolvedor e se passar futuramente entrará em produção.

## FERRAMENTAS DE CI/CD

Uma ferramenta de CI/CD é responsável por coletar códigos de algum versionador e executar na *pipeline* que segundo Len Bass (2015, p. 87) é o local onde os aspectos arquitetônicos e os processos do DevOps se cruzam. As etapas definidas têm suas propriedades e especificidades, cada ferramenta tem a sua maneira de se definir uma *pipeline*.

Um exemplo de uma das ferramentas mais conhecidas é o Jenkins, que utiliza a como base de configurações das *pipelines* a linguagem Groovy, já ferramentas mais modernas, como GitHub Actions, GitLab CI, Circle CI, utilizam da linguagem Yaml para montar sua lista de tarefas na *pipeline*.



## RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR ISSN 2675-6218

GARANTIA DA QUALIDADE DE SOFTWARE COM DEVOPS  
Victor Hugo Pizzaia, Rodrigo Daniel Malara

Para a escolha da ferramenta, levou-se alguns critérios em conta como a integração com outras ferramentas, facilidade da linguagem, semântica da escrita, robustez dos servidores para executar múltiplas *pipelines*, dentre outras necessidades. Neste estudo de caso foi utilizada a ferramenta GitHub Actions por ser uma plataforma gratuita, de fácil entendimento de sua linguagem de integração e maturidade no mercado, garantindo mais formas de integrar com outras ferramentas e possibilidade de armazenar nosso código.

### **GITHUB ACTIONS**

Segundo o GitHub a plataforma *GitHub Actions*, consiste em uma ferramenta de CI/CD, que facilita a automação dos processos de *software*, onde é possível criar vários fluxos de trabalho de maneira simples, fazendo com que a organização deixe de usar diversas ferramentas diferentes para cada etapa do desenvolvimento do produto.

Para integrar algumas ferramentas de CI/CD é necessário ter o código versionado em alguma ferramenta, e é aí que o GitHub entra para poder auxiliar pois nossos códigos são versionados e divididos em *branches*, que de acordo com o Chacon (2014), uma *branch* quer dizer que você está diferente da linha principal do código e consegue trabalhar sem alterar a linha principal, como por exemplo uma versão de desenvolvimento e uma versão de qualidade. Nossa ferramenta de integração, quando analisar que houve algum evento específico nas *branches*, é avisada e começa o processo que definido como testes e validações.

### **TESTES UNITÁRIOS**

Para Huttermann (2012, p. 58), os testes de unidade são responsáveis por verificar unidades únicas, geralmente em uma classe. Não há nenhuma dependência externa (por exemplo, bancos de dados). A família de ferramentas xUnit é o padrão de fato para código de teste de unidade. É possível executar os testes de unidade localmente, que consistem no espaço de trabalho do desenvolvedor e em um ambiente central em seu servidor de compilação.

Os testes unitários executam as funções isoladamente em pequenas partes, validando desde um cadastro simples até uma equação matemática complexa, consistem em receber uma entrada, processar executando a função a ser testada e retornar um valor a ser testado com o que é esperado pelo teste.

### **TESTES DE INTEGRAÇÃO**

Os testes de integração consistem em validar o sistema como um todo, validando a conversa entre todos os componentes da aplicação. Segundo Huttermann (2012, p. 59), os testes de integração conectam vários componentes e testam o sistema como um todo sem relação com as interfaces do usuário.



## **JUNIT**

Conforme descrito pela plataforma JUnit (2022), a biblioteca serve como base para executar as estruturas de teste criadas em Java, possibilitando a execução de diversos tipos de testes que segundo Huttermann (2012, p. 58) incluem os testes de unidade, testes de serviço e testes de interface do usuário.

## **MOCKITO**

O Mockito é um *framework* de injeção de dados, que permite a escrita de testes com uma API limpa e simples, os testes do Mockito são muito legíveis e produzem erros de verificação limpos (MOCKITO, 2022). A utilização deste *framework* é importante para executar os testes que têm dependências de classes internas como por exemplo a utilização da classe responsável por conectar o sistema ao banco de dados.

## **DOCKER**

O Docker é uma ferramenta que facilita a construção de ambientes virtuais isolados eliminando tarefas de configuração repetitivas e manuais, ele é usado em todo o ciclo de vida de desenvolvimento do *software* (DOCKER, 2022). A utilização de *containers* na aplicação facilita a configuração de todo o ambiente desde o desenvolvimento até a entrega contínua, torna o ambiente mais leve e seguro de possíveis falhas por conta de configurações da máquina de cada desenvolvedor.

## **2 DESENVOLVIMENTO**

No desenvolvimento desta seção, são mostradas as ferramentas necessárias (Testes Unitários, testes de integração) para o desenvolvimento deste trabalho, mostrando como foi a construção do ambiente. Como base desta pesquisa foi utilizada uma aplicação já estruturada que consiste em um cadastro de tarefas, podendo ser cadastrado nome, data de vencimento, descrição e *status*. O código desta aplicação está disponível no seguinte *site* (GITHUBVICTORPIZZAIA, 2022).

As próximas etapas deste trabalho incluem configurar o ambiente, criar os testes de unidade, criar os testes de integração, executar as validações do SonarQube e validar a cobertura de código dos testes, fazer a montagem da *pipeline* de validação, a função da *release* do projeto, a execução do *build* do container, a entrega do código em produção e a última etapa consiste na *pipeline* que orquestra os passos de produção.

## **CONFIGURAÇÃO DO AMBIENTE**

A primeira etapa que deu início ao desenvolvimento do projeto foi configurar as bibliotecas (JUnit) e os *plugins* (Release, Sonar) que foram utilizados durante os processos. Em primeiro lugar a biblioteca que é responsável por executar os testes unitários, o JUnit, com a versão que utilizada, traz





## RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR ISSN 2675-6218

GARANTIA DA QUALIDADE DE SOFTWARE COM DEVOPS  
Victor Hugo Pizzaia, Rodrigo Daniel Malara

diversas outras bibliotecas incluídas que fazem parte do desenvolvimento dos testes como o Mockito e o AssertJ.

Os *plugins* utilizados foram necessários para fazer a integração com o SonarQube, gerar os dados dos testes executados para análise posterior do Sonar e executar o passo de *release* assim alterando a versão do projeto, e gerando os artefatos que são armazenados em histórico.

### TESTES UNITÁRIOS

A principal camada do projeto a ser testada unitariamente é a camada de serviço, que contém as lógicas e validações para adicionar, alterar, buscar e excluir algum dado da aplicação. É de suma importância testar se ao enviar algum dado faltando a aplicação, se consiga perceber e barrar a chegada ao banco de dados, e os testes de unidade garantem que se alguma alteração no código quebre esse fluxo, fique bem visível e palpável ao programador o que está acontecendo de errado.

O primeiro passo foi configurar as classes que necessitam de alguma dependência, como por exemplo a função responsável por conversar com o banco de dados, os testes de unidade não podem ter dependências externas e é aí que o Mockito auxilia o nosso serviço, ele vai ser responsável por injetar uma classe fictícia que funciona de forma semelhante ao cenário real e assim foi possível a escrita dos testes.

Na camada de serviço existem algumas exceções específicas que devem ser validadas por diferentes testes, salvar uma tarefa com sucesso, não salvar uma tarefa por falta do nome ou não salvar por falta da data de vencimento, cada pequena lógica de validação deve ser testada unitariamente, onde é executado a função e verificado com o Mockito se a resposta era aquela que realmente espera-se como pode ser observado na figura 1:

Figura 1 – Testes unitários

```

1  @Test
2  void shouldSaveTaskWithSuccess() {
3      Task task = new Task();
4      task.setName("Testes");
5      task.setDueDate(LocalDate.now());
6      taskService.saveOrUpdate(task);
7      Mockito.verify(taskRepository).save(task);
8  }
9
10 @Test
11 void shouldNotSaveTaskWithoutName() {
12     Task task = new Task();
13     task.setDueDate(LocalDate.now());
14     try {
15         taskService.saveOrUpdate(task);
16     } catch (IllegalArgumentException e) {
17         assertThat(e.getMessage()).isEqualTo("Preencha o nome da tarefa");
18     }
19 }
20

```

Fonte: Elaborada pelo autor (2022)



## TESTES DE INTEGRAÇÃO

A construção dos testes de integração foi semelhante aos testes de unidade, mas além de configurar a classe responsável pelo banco, foi necessário fazer a injeção do Mockito para as demais funcionalidades do código (Camada de Serviço, Camada do Banco), além de validar somente a mensagem de retorno da função, é necessário também validar se o código de *status* está retornando corretamente.

Os testes de integração são responsáveis também por validar se os dados chegam corretamente à aplicação, ao chamar remotamente a função como na figura 2:

Figura 2 – Testes de integração

```

1  @Test
2  public void shouldReturnTaskByName() throws Exception {
3      Mockito.when(taskService.findByName("Testes")).thenReturn(utilTask);
4
5      MockHttpServletResponse response = mockMvc.perform(
6          MockMvcRequestBuilders.get("/todo?name=Testes").accept(MediaType.APPLICATION_JSON)
7      )
8          .andReturn()
9          .getResponse();
10
11     List<Task> expected = new ArrayList<>();
12     expected.add(new Task(utilDtoTask));
13     assertThat(response.getStatus()).isEqualTo(HttpStatus.OK.value());
14     assertThat(response.getContentAsString())
15         .isEqualTo(
16         listJson.write(expected).getJson()
17     );
18 }
19

```

Fonte: Elaborada pelo autor (2022)

## SONARQUBE E QUALITY GATE

A etapa de integração com o Sonar foi delegada ao uso do plugin disponibilizado pela própria empresa que traz a lógica de conversa entre o código e a plataforma e de uma maneira simples foi executada a validação do código executando o comando `'gradle sonar'`. Executando este comando, o sonar faz a análise estática do código e informa como está a saúde do código, o número de possíveis falhas, os cheiros ruins, dentre diversas outras métricas.

Ao chamar o comando de verificação do sonar, ele executa um passo de teste do código, que é responsável por testar toda a aplicação e coletar os artefatos dos resultados, após isso, o sonar executa a leitura do resultado e faz a validação se a cobertura de testes do código é maior que 80%, se essa sentença for verdadeira, ele retorna à sinalização que o código passou na validação. Esta análise foi utilizada na etapa de validação com a *pipeline*.





### **PIPELINE DE VALIDAÇÃO**

A construção da primeira *pipeline* consiste em executar uma varredura em todo código e verificar se tudo está correto para prosseguir para os próximos passos, nela está presente o passo de compilação do código, a execução dos testes, e a validação do sonar juntamente com o portão de qualidade.

A primeira etapa para criar a *pipeline* é definir quando ela será executada e o que ela executará, foi definido que a validação executará somente quando uma solicitação de alteração de código for aberta, sincronizada ou reaberta e a esta *pipeline* executa três etapas, o *download* das bibliotecas e plugins que foram definidos, todos os testes criados e na última etapa o SonarQube é chamado fazendo toda a varredura pelo código e executa a validação do *quality gate*, se a cobertura de código for maior que a meta definida de 80% a alteração de código que foi solicitada recebe uma aprovação do GitHub Actions.

### **UMA NOVA VERSÃO COM A RELEASE**

Toda alteração de código que entra em produção tem a necessidade de ser gerado uma nova versão para manter o controle das etapas de desenvolvimento, qualidade e o código que está em produção, esta etapa do trabalho é responsável por gerar a nova versão do código e armazenar toda nova alteração que entrou em produção.

Para este processo foi utilizado um *plugin* que executa todo o processo de *release* automatizado, a única alteração opcional que foi realizada foi alterar o nome que ele gerava na versão, foi definido para gerar somente números como versão.

### **CONSTRUÇÃO DO CONTAINER**

Está etapa consiste na criação do container utilizando *docker* para disponibilizar a aplicação em produção, para construir a imagem foi definido duas etapas, a primeira consiste em realizar o *build* de toda a aplicação gerando o arquivo compilado e a segunda etapa busca o arquivo gerado pela primeira e executa o comando 'java -jar' para iniciar a aplicação seguindo a figura 3:



Figura 3 – Dockerfile

```

1 FROM gradle:jdk11-alpine AS build
2 COPY --chown=gradle:gradle . /home/gradle/src
3 WORKDIR /home/gradle/src
4 RUN gradle build --no-daemon
5
6 FROM openjdk:11-jre-slim
7
8 EXPOSE 8080
9
10 RUN mkdir /app
11 WORKDIR /app
12
13 ARG PROJECT_VERSION
14 ENV PROJECT_VERSION $PROJECT_VERSION
15
16 COPY --from=build /home/gradle/src/build/libs/ToDoAPI-$PROJECT_VERSION.jar /app/ToDo-application.jar
17
18 ENTRYPOINT ["java", "-jar", "/app/ToDo-application.jar"]
19

```

Fonte: Elaborada pelo autor (2022)

### PIPELINE DE ENTREGA CONTÍNUA

A última etapa deste trabalho foi criar a *pipeline* de entrega contínua, que consiste em gerar a *release* do projeto, e construir a imagem do nosso container usando *docker*. O gatilho para essa *pipeline* executar é ativado toda vez que uma alteração no código é confirmada, ou seja, depois que toda a verificação de código que a *pipeline* de validação executou e o código foi aprovado, a entrega contínua inicia seu processo.

Está *pipeline* é constituída por duas etapas sendo que a primeira executa o comando de *release* da aplicação automaticamente e a segunda faz a construção do container a partir da configuração criada como pode ser observado na figura 4:



Figura 4 – Pipeline de entrega

```

1  name: Build Docker and Release CI
2
3  on:
4    push:
5      branches:
6        - master
7
8  jobs:
9    release:
10   runs-on: ubuntu-latest
11   steps:
12     - uses: actions/checkout@v3
13     - name: Set up JDK 11
14       uses: actions/setup-java@v3
15       with:
16         java-version: '11'
17         distribution: 'temurin'
18     - name: Set git profile
19       uses: fregante/setup-git-user@v1
20     - name: Release with Gradle
21       uses: gradle/gradle-build-action@v2
22     - run: chmod +x ./gradlew
23     - run: gradle release -Prelease.useAutomaticVersion=true
24
25   docker-build:
26     runs-on: ubuntu-latest
27     needs: release
28     steps:
29       - uses: actions/checkout@v3
30       - name: Get project version and build Docker
31         id: get-version
32         uses: gradle/gradle-build-action@v2
33       - run: chmod +x ./gradlew
34       - run: echo "projectVersion=$( gradle properties -q | grep 'version:' | grep -e '^[\[\]]' | sed -e 's/version:\ //g' )" >> $GITHUB_ENV
35       - run: docker build . --file Dockerfile --tag todo-api:${{ env.projectVersion }} --build-arg PROJECT_VERSION=${{ env.projectVersion }}
36

```

Fonte: Elaborada pelo autor (2022)

### 3 RESULTADOS

Os resultados obtidos neste trabalho foram principalmente a garantia da qualidade do código em todo ciclo de desenvolvimento da aplicação, onde a execução dos testes e a validação de código, foram automatizadas em todo o processo de desenvolvimento, que por sua vez permitiu que a cada alteração realizada no código fosse executado uma nova avaliação em todas as funcionalidades da aplicação e a partir do *quality gate* é passado para a próxima etapa, gerando *release* e fazendo a construção do *container* para a produção.

A aplicação é totalmente testada e avaliada se a cobertura do código está acima da métrica definida, desta forma garante que novos códigos sempre sejam avaliados e testados. Executando o sonar se conseguiu avaliar como está a saúde do código da aplicação, avaliando os problemas futuros, as possíveis falhas, a cobertura de testes e a repetição de código.

### CONSIDERAÇÕES FINAIS

O processo de automação de teste de *software* se mostrou muito importante no ciclo de desenvolvimento de código, visto que esses processos são realizados manualmente nas empresas. Este trabalho propôs melhorar a qualidade do código automatizando os processos utilizados pelas empresas agilizando as entregas do *software*.

Com a pesquisa de possíveis soluções para a automação de processos foi identificado o DevOps, que utilizando as ferramentas de CI/CD foi empregado em diferentes estágios para melhorar



## RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR ISSN 2675-6218

GARANTIA DA QUALIDADE DE SOFTWARE COM DEVOPS  
Victor Hugo Pizzaia, Rodrigo Daniel Malara

a qualidade do código e automatizar os processos manuais, como a execução de *release*, o *build* da aplicação, a construção do *container* com docker.

A ferramenta de CI/CD, que auxiliou a construção deste trabalho, foi o GitHub Actions, que disponibiliza de uma forma simples para armazenar o código que está em desenvolvimento e prontamente dispõe dos processos que foram utilizados neste trabalho utilizando suas *pipelines*.

Como proposta futura podem ser adicionados diferentes tipos de validação nas *pipelines*, como por exemplo inserir um passo que valida a segurança do código tornando o projeto em uma metodologia DevSecOps.

### REFERÊNCIAS

AWS. **O que é o DevOps?** [S. l.]: AWS, 2022. Disponível em: <https://aws.amazon.com/pt/devops/what-is-devops/>. Acesso em: 07 jun. 2022.

BASS, I.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice**. 3. ed. New York: Addison-Wesley Professional, 2015

CHACON, S.; STRAUB, B. **Pro Git**. 2 ed. [S. l.]: Apress, 2014.

DOCKER. **Desenvolva mais rápido**: Corra para qualquer lugar. [S. l.]: DOCKER, 2022. Disponível em: <https://www.docker.com/>. Acesso em: 23 set. 2022.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. **Continuous Integration**: Improving Software Quality and Reducing Risk. New York: Addison-Wesley Professional, 2007.

GAUDIN, O.; CAMPBELL, G. A.; PAPAPETROU, P. P. **SonarQube in Action**. [S. l.]: Manning Publications, 2013

GITHUB. **Ações do GitHub**. [S. l.]: GITHUB, 2022. Disponível em: <https://github.com/features/actions>. Acesso em: 06 jun. 2022.

GITHUBVICTORPIZZAIA. **ToDoAPI**. [S. l.]: GITHUBVICTORPIZZAIA, 2022. Disponível em: <https://github.com/Victor-Pizzaia/ToDoAPI>. Acesso em: 06 set. 2022.

HUTTERMANN, M. **Devops for Developers**. [S. l.]: Apress, 2012

JUNIT. **JUnit5 User Guide**. [S. l.]: JUNIT, 2022. Disponível em: <https://junit.org/junit5/docs/current/user-guide/>. Acesso em: 22 set. 2022.

MARTIN, R. C. **Clean Code**: A Handbook of Agile Software Craftsmanship. Nova Jersey: Prentice Hall PTR, 2008.

MEYER, B. **Object-Oriented Software Construction**. 2 ed. Nova Jersey: Prentice Hall, 1997.

MOCKITO. **Mockito**. [S. l.]: Mockito, 2022. Disponível em: <https://site.mockito.org/#intro>. Acesso em: 23 set. 2022.

REDHAT. **Pipeline de CI/CD**. [S. l.]: REDHAT, 2022. Disponível em: <https://www.redhat.com/pt-br/topics/devops/what-cicd-pipeline>. Acesso em: 07 jun. 2022.



**RECIMA21 - REVISTA CIENTÍFICA MULTIDISCIPLINAR**  
**ISSN 2675-6218**

GARANTIA DA QUALIDADE DE SOFTWARE COM DEVOPS  
Victor Hugo Pizzaia, Rodrigo Daniel Malara

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson, 2011.

SONARQUBE. **SonarQube Documentation**. [S. l.]: SONARQUBE, 2022. Disponível em:  
<https://docs.sonarqube.org/latest/>. Acesso em: 07 jun. 2022.