EXECUÇÃO DE TESTE COM A FERRAMENTA JENKINS E CONTAINER DOCKER.

*SILVA, Ivan Jonatas 1

1 Tecnólogo, Sistema para Internet

1. INTRODUÇÃO

Teste de *software* é uma atividade no ciclo de desenvolvimento que consiste em reduzir o risco de falha de um *software* trazendo uma maior qualidade. Teste de *software* é a maneira de identificar se o comportamento será executado conforme especificado, a execução deste teste é um ambiente controlado e tem como objetivo principal revelar o maior número de falhas possível promovendo o resultado ao time de desenvolvimento e com isso o time identificar se o *software* está conforme os padrões estabelecidos.

Não se pode prever que todos funcione corretamente, sem a presença de erros, visto que os mesmos muitas vezes possuem um grande número de estados com fórmulas, atividades e algoritmos complexos. O tamanho do projeto a ser desenvolvido e a quantidade de pessoas envolvidas no processo aumentam ainda mais a complexidade. (MYERS, 2004). Com o livro "*The art of software testing*" *Glenford Myers*, trouxe ao leitor um conhecimento mais avançado sobre testes, abordando sobre as técnicas de software no ano de 1979. Este livro é muito conhecido pelos profissionais, podendo ser tratado como uma referência teórica, com o ano de 1979 um grande passo para a área.

A tarefa de efetuar testes em *software* foi considerada pouco importante por muito tempo. Antigamente os testes eram feitos pelos próprios desenvolvedores e geralmente era vista como uma tarefa onde não se deveria gastar muito com tempo e investimentos. Porém as coisas mudaram, com a globalização e a competição entre as empresas está havendo uma grande preocupação em aprimorar e aperfeiçoar os processos de testes no ciclo de desenvolvimento com vistas a reduzir custos com manutenção e produzir um produto de boa qualidade. A qualidade no desenvolvimento de *software* é fundamental para o sucesso de um projeto, tanto na concepção do produto

como nas atividades de manutenção e desenvolvimento de novas funcionalidades. A qualidade está fortemente relacionada ao atendimento dos requisitos do projeto por meio de artefatos gerados com exatidão e com confiabilidade (MATHUR, 2009).

O grande desafio das empresas é produzir *softwares* com qualidade, em um curto espaço de tempo, com baixo custo e atender as expectativas do cliente com o produto desenvolvido. Dentro do ciclo de desenvolvimento, a atividade de teste é realizada para garantir o comportamento do *software* conforme o esperado (MATHUR,2009). Então, vimos que o teste passa a ser uma atividade muito importante, e levando em consideração essas afirmações, as empresas passam então a, montar uma equipe capacitada para os testes e a atenderem todos os critérios de qualidade.

DevOps é um termo criado com o objetivo de descrever um conjunto de práticas para integração entre as equipes de desenvolvimento de *softwares*, operações (infraestrutura ou *sysadmin*) e de apoio envolvidas (como controle de qualidade) e a adoção de processos automatizados.

Tradicionalmente Desenvolvimento e Operações são setores diferentes nas empresas e com motivações distintas mas com a implementação de um modelo de *DevOps*, essas equipes não ficam mais separadas. Nessa mecânica de trabalho, elas compartilham processos, recursos, responsabilidades e ferramentas que favorecem e unificam o seu trabalho. No caso as ferramentas potencializar a produtividade, resolvendo problemas diários de ambas as equipes.

Então, o *DevOps* aumentam a capacidade de uma empresa de distribuir aplicativos e serviços em alta velocidade: otimizando e aperfeiçoando produtos em um ritmo mais rápido do que o das empresas que usam processos tradicionais de desenvolvimento de *software* e gerenciamento de infraestrutura. Essa velocidade permite que as empresas atendam melhor aos seus clientes e consigam competir de modo mais eficaz no mercado.

2. FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta toda a teoria e fundamentos usado para a produção deste trabalho.

2.1 TESTE DE SOFTWARE

Todos os testes “têm por objetivo identificar o maior número possível de erros tanto nos componentes isolados quanto na solução tecnológica como um todo” (BARTIÉ, 2002). Para encontrar todos os erros, os testes são separados em diferentes tipos e executado conforme a sua finalidade. Com isso o *software* terá várias cargas de teste cada um com seu objetivo específico

2.1.1 Tipos de teste

Com base nos objetivos do projeto define-se que tipo de teste será executado e qual características específicas será testada. O teste pode testar um sistema de *software* inteiro, ou parte dele. Na tabela 1 é demonstrado os tipos de testes e seus objetivos.

Tabela 1 - Tipos de teste

Tipo de Teste	Descrição
Teste de Unidade	Teste em um nível de componente ou classe. É o teste cujo objetivo é um “pedaço do código”.
Teste de Integração	Garante que um ou mais componentes combinados (ou unidades) funcionam. Podemos dizer que um teste de integração é composto por diversos testes de unidade.
Teste Operacional	Garante que a aplicação pode rodar muito tempo sem falhar.
Teste Positivo-negativo	Garante que a aplicação vai funcionar no “caminho feliz” de sua execução e vai funcionar no seu fluxo de exceção.
Teste de Regressão	Toda vez que algo for mudado, deve ser testada toda a aplicação novamente.
Teste de Caixa-preta	Testar todas as entradas e saídas desejadas. Não se está preocupado com o código, cada saída indesejada é visto como um erro.
Teste Caixa-branca	O objetivo é testar o código. Às vezes, existem partes do código que nunca foram testadas.
Teste Funcional	Testar as funcionalidades, requerimentos, regras de negócio presentes na documentação. Validar as funcionalidades descritas na documentação (pode acontecer de a documentação estar inválida)
Teste de Interface	Verifica se a navegabilidade e os objetivos da tela funcionam como especificados e se atendem da melhor

	forma ao usuário.
Teste de Performance	Verifica se o tempo de resposta é o desejado para o momento de utilização da aplicação.
Teste de Carga	Verifica o funcionamento da aplicação com a utilização de uma quantidade grande de usuários simultâneos.
Teste de Aceitação do usuário	Testa se a solução será bem vista pelo usuário. Ex: caso exista um botão pequeno demais para executar uma função, isso deve ser criticado em fase de testes. (aqui, cabem quesitos fora da interface, também).
Teste de Volume	Testar a quantidade de dados envolvidos (pode ser pouca, normal, grande, ou além de grande).
Testes de Stress	Testar a aplicação sem situações inesperadas. Testar caminhos, às vezes, antes não previstos no desenvolvimento/documentação.
Testes de Configuração	Testar se a aplicação funciona corretamente em diferentes ambientes de hardware ou de *software*.
Testes de Instalação	Testar se a instalação da aplicação foi OK.
Testes de Segurança	Testar a segurança da aplicação das mais diversas formas. Utilizar os diversos papéis, perfis, permissões, para navegar no sistema.

Fonte: Portal GSTI (2019)

Dos testes citados na tabela 1, os testes de regressão são um dos principais tipos de testes a serem automatizados, pois geralmente são repetitivos e de longa execução (ISTQB, 2018)

2.2 FERRAMENTA DE AUTOMAÇÃO

De acordo com o que diz o ISTQB (2018), as ferramentas de automação de testes funcionam com a execução de um conjunto de instruções redigidas em uma linguagem de programação, que na maioria das vezes é chamada de linguagem *script*. As ferramentas têm instruções claras da ordem e valores de entrada e da ordem e valores de saída assim a própria ferramenta consegue identificar os testes que executaram corretamente e os que falharam. Existem algumas ferramentas que pode ser utilizadas para automatizar os testes,Entre elas estão o *Selenium, Cucumber, TestComplete,* entre outras. Como pode ser observado na tabela 2 cada uma dessas ferramentas possui funcionalidades específicas para determinados ambientes.

Tabela 2 - Ferramentas de automação mais utilizadas

Ferramenta	Ambiente	Linguagem
SELENIUM	Web	Java, Perl, JavaScript, PHP, Python,Ruby
CUCUMBER	Web	Ruby,Java, NET, Scala, Groovy
TESTCOMPLETE	Desktop, Mobile, Web	Python, JScript, Delphi, C++ e C#
TELERIK TS	Mobile, We	Angular, Asp.Net, HTML5, JavaScript, Ajax, Silverlight, Ruby, PHP, VB.Net e C#.
ROBOTIUM	Mobile	Java

Fonte: CONTROL (2018)

Dentre as ferramentas citadas na tabela 2, o selenium foi a escolhida pelo time de teste para ser usada na criação e execução de testes regressivos.

2.3 FERRAMENTA PARA EXECUÇÃO DE TESTES AUTOMATIZADO

2.3.1 Jenkins

O *Jenkins* é uma bifurcação do projeto *Hudson*, um servidor de integração contínua, iniciada pela empresa *Sun Microsystems*, porém quando a *Sun* foi adquirida pela *Oracle Corporation*, que pretendia desenvolver uma versão comercial do *software*. Com a bifurcação do projeto *Hudson* houve algumas modificações do projeto inicial, como o nome que passou a ser chamado de *Jenkins* e transformações do projeto em um *software* de código aberto. Com isso o *jenkins* passou a ser um servidor de automação de código aberto independente que pode ser usado para automatizar todos os tipos de tarefas relacionadas à construção, teste e entrega ou implantação de *software.*

O *jenkins* é um servidor de integração contínua devido ter a capacidade de recuperar as alterações ocorridas no repositório e integrá-las,em uma *pipeline* a fim de fazer uma esteira de construção de *software*. A integração contínua é uma abordagem de desenvolvimento que implementa um *loop* de "implantação de teste de desenvolvimento", o que significa que todas as alterações no *software* são testadas automaticamente antes da implantação.

Além disso, após a falha do teste, as alterações não serão enviadas para a plataforma principal. Como resultado, os desenvolvedores podem fazer alterações sem se preocupar em quebrar as compilações principais.

Jenkins fornece um conjunto extensível de ferramentas para modelar pipelines de entrega simples a complexos 'como código' . A ideia é que dentro da *Pipeline* o código possa ser testado o mais rápido possível para identificar os problemas mais cedo, onde grande parte desse trabalho é realizado por testes automatizados. A definição de uma *pipeline* é normalmente escrita em um arquivo de texto, chamado *Jenkinsfile*.

2.3.2 Allure Test Reports

O *Allure* é uma ferramenta que criar relatórios de execução de testes multilíngue leve e flexível que mostra uma representação muito concisa do que foi testado em um formulário de relatório da *web*.

O *Allure* reduz o ciclo de vida de defeitos comuns: falhas de teste podem ser divididas em bugs e testes quebrados, além de logs, etapas, acessórios, anexos, tempos, histórico e os sistemas de rastreamento de bugs pois é fornecido uma visão geral clara de quais recursos foram abordados, onde os defeitos estão agrupados, como é a linha do tempo da execução e muitas outras coisas convenientes. Os desenvolvedores e testadores responsáveis terão todas as informações em mãos.

Figura 1 - Dashboard Allure
Fonte: Allure (2019)

Na figura 1 estão algumas características básicas de um projeto e ambiente de teste.

- *Statistics* - estatísticas gerais do relatório;
- *Launches* - se este relatório representar vários lançamentos de teste, as estatísticas por lançamento serão mostradas aqui;
- *Behaviors* - informações sobre resultados agregados de acordo com histórias e recursos;
- *Executors* - informações sobre executores de teste que foram usados para executar os testes;
- *History Trend* - se os testes acumularem alguns dados históricos, a tendência será calculada e mostrada no gráfico;
- *Environment* - informações sobre o ambiente de teste.

2.3.3 Docker

Docker é uma plataforma *Open Source* escrito em *Go*, que é uma linguagem de programação de alto desempenho desenvolvida dentro do *Google*, que facilita a criação e administração de ambientes isolados. Com o *docker* é possível automatizar a implantação de aplicações dentro de ambientes isolados denominados *containers*. É uma solução para desenvolvedores e administradores de sistema desenvolverem, embarcarem, integrarem e executarem aplicações rapidamente. Seu principal objetivo é proporcionar múltiplos ambientes isolados dentro do mesmo servidor, mas acessıveis externamente via portas.

A utilização do *docker* para a conteinerização se concentra na habilidade de desativar uma parte de uma aplicação, seja para reparo ou atualização, sem interrompê-la totalmente. Pois é possível executar e gerenciar as suas aplicações lado a lado em contêineres isolados para obter uma melhor densidade computacional.

A tecnologia pode aparentar uma semelhança estrutural de ambiente virtualizado porém a diferenças. Virtualizar um ambiente é simular um computador por meio de um *software*. Antes do conceito de virtualização, era necessário dedicar todos os recursos de um servidor a execução de uma única aplicação para garantir que ela funcionasse de forma isolada das demais. Hoje,

a virtualização permite a execução de várias aplicações no mesmo servidor criando um isolamento lógico entre elas. Embora tenha sido uma grande evolução em relação ao modelo anterior, a virtualização por meio de máquinas virtuais consome uma grande parte dos recursos do computador devido o *software* simular o funcionamento do *hardware* e do sistema operacional.

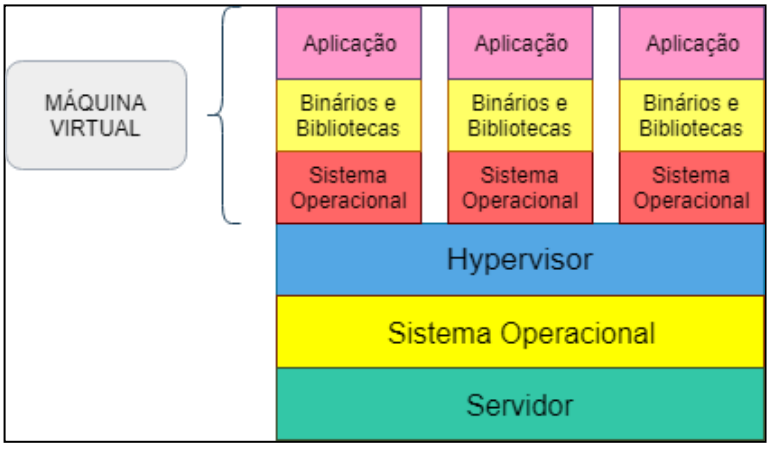

Figura 2 - Servidor utilizando Máquinas Virtuais
Fonte: Devmedia (2018)

Por outro lado, os *containers* não simula todos os componentes de um servidor físico, pois compartilha apenas o *Kernel* da máquina. Fornecendo uma virtualização a nível de sistema operacional, assim as aplicações em execução não utilizar tantos recursos da máquina.

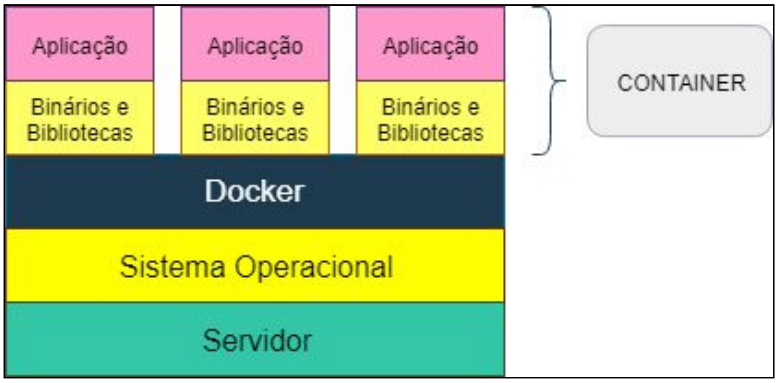

Figura 3 - Servidor utilizando Docker
Fonte: Devmedia (2018)

A criação de ambiente em *Docker* é feito usando um padrão de empacotamento, esse padrão é escrito no arquivo *Dockerfile*. Neste arquivo contém todos os comandos que normalmente seriam usados para configurar

manualmente a implantação do ambiente.

```
FROM tomcat:8.0-alpine

LABEL maintainer="Ivan Jonatas"

ADD example.war /usr/local/tomcat/webapps/example.war

EXPOSE 8080

CMD ["catalina.sh", "run"]
```

Figura 4 - Dockerfile para Construir um servidor
Fonte: Próprio Autor (2020)

Uma vez criada a estrutura do ambiente conforme a figura 4, ela pode ser facilmente replicada, usada como referência para a criação de novas estruturas.

3. METODOLOGIA

3.1 TIPOS DE PESQUISA

Este trabalho é definido como uma pesquisa qualitativa, ao observar o tipo de abordagem utilizada. O trabalho tem como natureza ser uma pesquisa aplicada, pois possuir uma prática e conter um objetivo pronto a solucionar um problema específico.

Analisando os objetivos desta pesquisa, concluir-se que ela é uma pesquisa descritiva, pois é feita uma descrição de ferramentas e *frameworks* que auxiliam na automação e execução de testes. Além de ter como objetivo a análise e a descrição do processo de automação de testes da empresa Conductor.

3.2 DESENVOLVIMENTO DA PESQUISA

O processo de automação dos testes de *software* realizados na empresa Conductor Soluções em Meios de Pagamento, com a ferramenta de automação *Selenium, Junit e Rest Assured*, e do remanejamento da execução de testes

regressivos do ambiente local para um ambiente compartilhado foi um dos motivos para a realização deste trabalho.

A rotina de execução dos testes regressivos são realizados diariamente, porém, mesmo com os testes automatizados, esta execução acarretava uma espera significativa da equipe de qualidade. Cada teste regressivo, conta em média com 200 casos de teste, levando cerca de 20 a 30 minutos até o fim de sua execução.

Assim o processo de desenvolvimento deste trabalho foi dividido nas seguintes etapas:

• Processo dos testes executado na máquina local;
• Criação dos ambientes *containerizado*;
• Execução dos testes com o *jenkins.*

3.3 RESULTADOS DA PESQUISA

Após a coleta e análise dos dados foi pensado na execução dessas classes de teste dentro de um container *docker* através da ferramenta de integração contínua *Jenkins*, onde além de permitir uma redução significativa no tempo de execução dos testes regressivos, permite também que os profissionais das equipes de qualidade trabalhem de forma simultânea à execução dos testes.

Entretanto, ao longo do projeto foi percebida a necessidade de monitorar todos os testes executados, desde o tempo de execução até o motivo pelo qual algum teste havia falhado. Esta necessidade por maiores informações os testes levou á um outro *framework*, intitulado *Allure*, que corresponde a um gerador de relatórios de testes

4. DESENVOLVIMENTO

Neste capítulo, é apresentado o estudo que tem como objetivo principal apresentar como foi realizado o processo de execução dos testes automatizado usando o *jenkins* e *container docker*.

4.1 PROCESSO DOS TESTES EXECUTADO NA MÁQUINA LOCAL

A execução dos testes era feito da seguinte maneira: primeiro o time de *Quality Assurance* (QA), criava os códigos de testes e executava em suas máquinas, enquanto os testes estava sendo executado era recomendado que o QA não trabalha-se em sua máquina para que a mesma não fica-se sobrecarregada pois sobrecarregando haveria um travamento no sistema completo e todos os teste executados teriam que ser executados novamente.

Após a execução em sua máquina o código é enviado para um repositório de versionamento de código, assim outro QA poderia executar os testes na sua maquina, porem teria que ter todas as configurações que o teste requer.

4.2 CRIAÇÃO DOS AMBIENTES CONTAINERIZADO

Todas as configurações do ambiente eram feitas de forma manual assim para os testes serem executados em outras máquinas ou em servidores era necessário configurar servidor a servidor ou máquina a máquina tornando a replicação do servidor um processo custoso e propenso a erros.

Visto que alguns testes de *software* tem versões e tecnologias diferentes, houve a necessidade da criação de um ambiente idêntico ao que os testes ja estava sendo executado. Com isso os ambientes foram configurados através da ferramenta do *Docker*, onde vários ambientes com as configurações dos testes e independentes podem ser idênticos devido às imagens conforme explicada na seção 2.3.3 DOCKER. Na figura 5 está todos os detalhes dessas configurações.

```
#Imagem com as configurações iniciais.
FROM maven:3.5-jdk-8-alpine
#Informando o caminho onde o projeto sera executado.
WORKDIR /app
#Instalando a ferramenta git.
RUN apk add git
#Variaveis necessarias para baixar o projeto e executa-lo.
ENV USER=null \
    PASS=null \
    BRANCH=null \
    URL=null \
    COMAND=null \
    PROJETO=/app
#configurando a imagem para utilizar o repositorio maven da conductor.
COPY settings.xml /root/.m2/
#enviando o arquivo com o comando de execução.
ADD run.sh run.sh
#Concedendo permissão ao arquivo para executar.
RUN chmod +x run.sh
#Instrução para executar o arquivo todas as vezes que um container for inicializado.
CMD ["/bin/bash","-c","./run.sh"]
```

Figura 5 - Configuração de um servidor
Fonte: Próprio Autor (2020)

Na figura 5 esta todos os comandos e o objetivo de cada um. Com esse comando o *docker* irá replicar um servidor para execução dos testes.

```
git clone -b ${BRANCH} https://${USER}:${PASS}@${URL}
cd ${PROJETO}
${COMAND}
```

Figura 6 - Arquivo com os comandos
Fonte: Próprio Autor (2020)

Na figura 6 está o detalhes do comando que o *docker* irá executar quando subi um container.

O *dockerfile* pode ser criado em qualquer máquina contudo a criação da imagem *docker* que servirá como base para subir um container *docker* tera que está salvar em um repositório de imagem *docker*. O repositório de imagem *docker* mais conhecido é o *docker hub*. Os comandos mostrado na tabela 3 irá criar a imagem e enviar para o *docker hub*, com a imagem no repositório qualquer pessoa pode criar o ambiente baseado na imagem.

Tabela 3 - Criando e enviado imagem *docker*

Comando	Descrição
docker build -t teste-servidor .	Criando a imagem base
docker push teste-servidor:latest	Enviando a imagem para o repositório

Fonte: Próprio autor (2020)

Para que o ambiente de teste seja executado em uma máquina ou servidor sem precisar configurar é necessário apenas instalar o *docker* e utilizando a imagem especificada acima com o seguinte comando:

*docker run -e BRANCH=${BRANCH} -e COMAND="mvn test" *
-e URL=${REPOSITÓRIO} servidor-teste:latest

É nesse comando que o desenvolvedor irá informar qual projeto será executado e qual o comando para executá-lo. Assim além de o docker criar o ambiente com todas as especificações também irá executar o projeto.

4.3 EXECUÇÃO DOS TESTES COM O JENKINS

Para que os testes de *software* automatizada seja executado de forma compartilhada, é necessário a configuração da ferramenta *Jenkin*s dentro de um único servidor em *cloud*. A ferramenta é acessada através da *VPN* que todos os desenvolvedores têm.

O *jenkins* tem a possibilidade de criar *jobs* que pode ser considerado como um compiladores de código-fonte, dentro desses *jobs* podem ser executados testes, configurações de servidores ou gerar novas versões em servidores *web*, tudo no formato de *pipelines*. Para que os testes sejam executados é necessário criar um *job*. A Figura 7 mostra como criar o *job*.

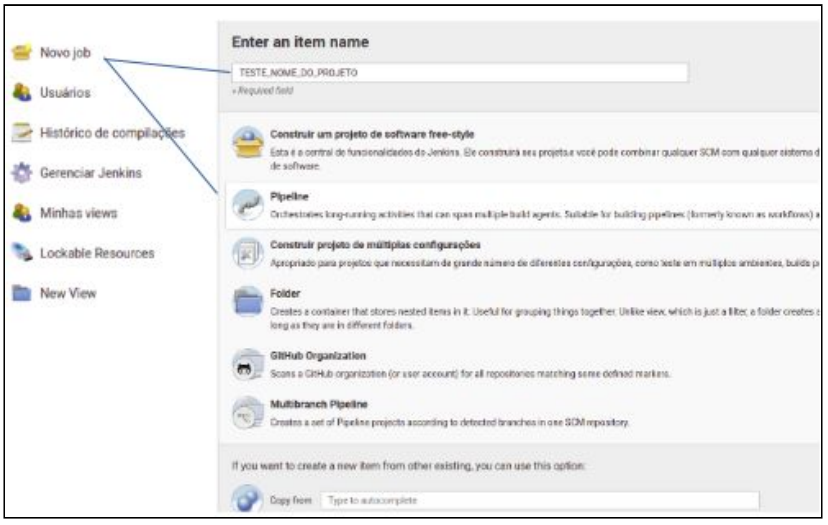

Figura 7 - Criando job
Fonte: Próprio Autor (2020)

No *job* criado é necessário fazer algumas configurações para que as variáveis que está no *Dockerfile* seja atendido. Para atender tais variáveis é necessário criar parâmetros dentro do *job*. A Figura 8 ilustra os parâmetros que terá que ser preenchido para executar o *job*.

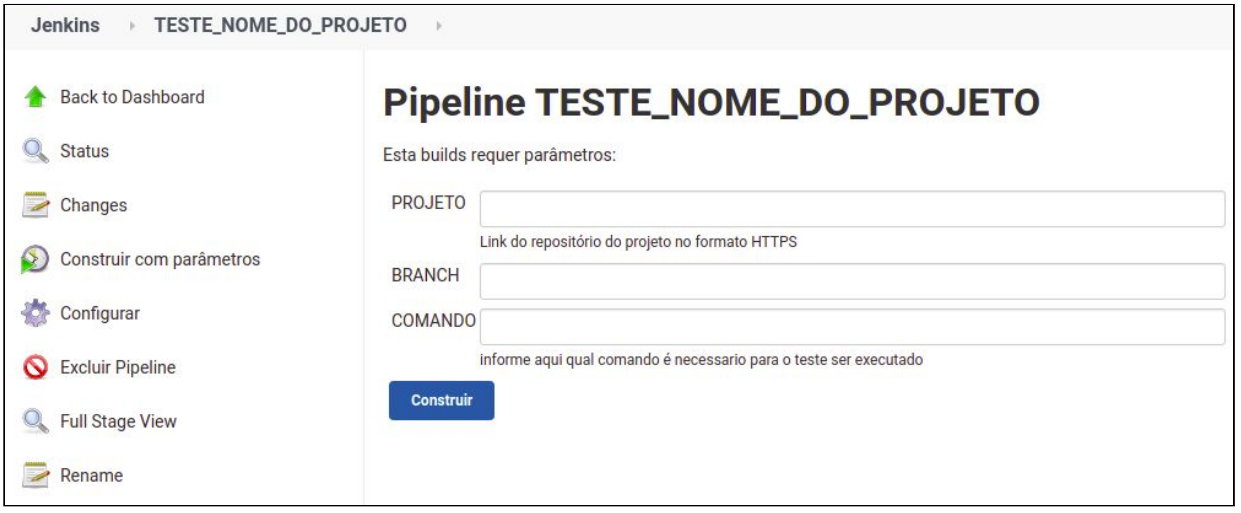

Figura 8 - Job parametrizado
Fonte: Próprio Autor (2020)

Com o *job* criado e parametrizado é necessário criar o que o *job* irá realizar. A Figura 9 mostra como será a compilação do código através do *job*.

```
// Iniciando uma pipeline onde o jenkins esta instalado
node(){
    // Primeira etapa da pipeline
    stage('Run teste'){
        // Executando o comando docker na maquina
        sh 'docker run --rm -e BRANCH=${BRANCH} -e COMAND="${COMANDO}" -e URL=${BRANCH} servido-teste:1.0.0'
    }
    // Segunda etapa da pipeline
    stage('Report Teste'){

        // Plugin para captação dos resultados dos testes
        allure([
                includeProperties: false,
                jdk: '',
                properties: [],
                reportBuildPolicy: 'ALWAYS',
                results: [[path: 'target/surefire-reports']]
            ])

    }
}
```

Figura 9 - Pipeline dentro do job
Fonte: Próprio Autor (2020)

Na figura 9 está o código que irá construir uma pipeline com dois processos, o primeiro irá *buildar* a aplicação e executar os testes e o segundo irá enviar os resultados para o *allure*.

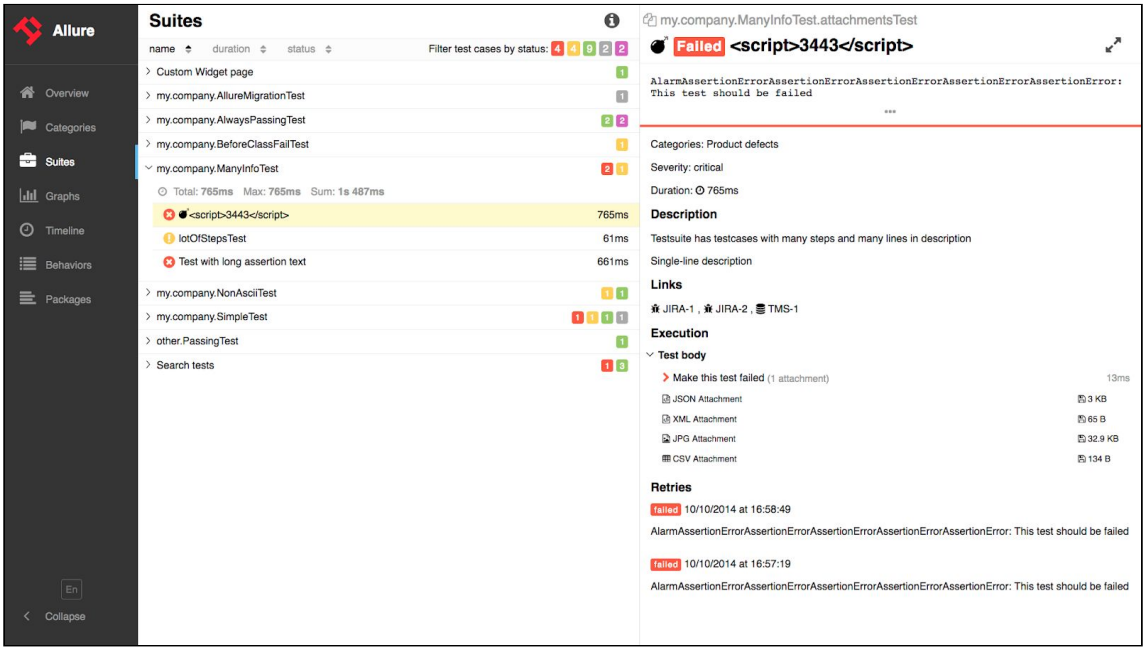

Figura 10 - Resultado dos teste
Fonte: Allure (2019)

após a execução do *job* o time de teste conseguirá visualizar os resultados dos testes no plugin *allure* que irá detalhar os resultados dos testes, conforme a Figura 10.

4. CONCLUSÃO

Neste trabalho foi apresentado uma nova forma de executar testes de *software* com tecnologia de conteinerização e ferramenta de continuous integration, com as configurações de novos ambientes de testes de forma automatizada e pragmática o time de teste tem uma melhora significativa em sua produtividade.

A configuração do ambiente utilizando a tecnologia de *containers* tem como objetivo facilitar a implantação de novos ambientes de teste. O objetivo de especificação do processo de integração contínua foi alcançado através da configuração da ferramenta do *Jenkins* como elucidado na Seção 4.3 EXECUÇÃO DOS TESTES COM O *JENKINS*, agregando uma maior produtividade e uma maior confiabilidade do cliente.

A partir disso, pode-se concluir que o novo processo de execução dos testes automática proporcionando ao time de teste um ambiente adequado para implantações de novos requisitos de maneira rápida e com garantia do mínimo de qualidade.

REFERÊNCIAS

FERNANDES, Matheus. **AUTOMAÇÃO DE TESTES DE SOFTWARE**: estudo de caso da empresa softplan. 2019. 70 f. Tese (Doutorado) - Curso de Sistema de Informação, Universidade do Sul de Santa Catarina, Florianópolis, 2019.

OLIVEIRA, Fábio Henrique Ferreira de. **Automação do processo de implantação e testes de um sistema WEB utilizando Docker**. 2019. 62 f. Tese (Doutorado) - Curso de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Universidade Federal do Rio Grande do Norte Unidade Acadêmica Especializada em Ciências Agrárias, Macaíba, 2019.

BARBIERI, Sergio Luis. **Teste de software na industria: Um estudo qualitativo**. 2018. 77 f. Tese (Doutorado) - Curso de Pós-Graduação em Sistemas de Informação˜, Universidade de São Paulo Escola de Artes, Ciências e Humanidades, São Paulo, 2018.

COSTA, Mozart Guerra. **Estratégia de Automação em Testes: Requisitos, Arquitetura e Acompanhamento de sua Implantação**. 2014. 2014 f. Tese (Doutorado) - Curso de Engenharia de Computação, Instituto de Computação Universidade Estadual de Campinas, Campinas, 2014.

VICCARI, Leonardo Davi. **AUTOMAÇÃO DE TESTE DE SOFTWARE ATRAVÉS DE LINHAS DE PRODUTOS E TESTES BASEADOS EM MODELOS**. 2009. 2009 f. Tese (Doutorado) - Curso de Pós-Graduação em Ciência da Computação, Universidade Católica do Rio Grande do Sul, Porto Alegre, 2009.

ISTQB, *International Software Testing Qualifications Board*. **Foundation Level Syllabus** - BSQTB, 2018

BARTIÉ, Alexandre. **Garantia da qualidade de software: As melhores práticas de**
Engenharia de Software aplicadas à sua empresa. Rio de Janeiro: Campus, 2002.

MATHUR, ***A. P. Foundations of software testing***. 2. ed.[S.l]: Pearson, 2009.

MYERS G. J.;SANDLER, **C. T. T. The Art of Software Testing**. [S.l.]: Wiley, New York, 2004.

CONTROL, Prime. **BDD in 3 Minutes**. Disponível em: https://hackernoon.com/bdd-in-3-minutes-c3f8fc022237. Acesso em: 16 abr. 2020.

PORTAL GSTI, **O que é um teste de software?** Florianópolis, 2019. Disponível em: https://www.portalgsti.com.br/testes-de-software/sobre. Acesso em: 20 jul. 2020.

VITORIO, diego. **Conhecendo as principais ferramentas de DevOps**. 2019. Disponível em: https://blog.dbacorp.com.br/ferramentas-devops. acesso em 07 set. 2020.

PEDRO, César Tebaldi Gomes. **A final o que é docker?**, 2018. Disponível em:https://www.opservices.com.br/o-que-e-docker/. acesso em 30 jul. 2020

REDHAT, **O que é Docker?**, 2017. Disponível em: https://www.redhat.com/pt-br/topics/containers/what-is-docker acesso em 30 jul. 2020

DIEDRICH, Cristiano, Docker 2015. Disponivel em: https://www.mundodocker.com.br/o-que-e-docker. acesso em 30 jul. 2020

DEVMEDIA. **Hello World com Docker**, 2018. Disponível em:https://www.devmedia.com.br/hello-world-com-docker. acesso em 30 jul. 2020

PEDRO, César Tebaldi Gomes. **A final o que é docker?**, 2018. Disponível em:https://www.opservices.com.br/o-que-e-docker/. acesso em 30 jul. 2020