METODOLOGIA PARA COBERTURA E QUALIDADE NO PROCESSO DE TESTE DE SOFTWARE EM NUVEM PARA APLICAÇÕES WEB

Wagner Ghedin[1], Ana Cláudia Garcia Barbosa[2]

Resumo: Projetar e desenvolver softwares com qualidade e consistência é uma tarefa complexa. Esta pesquisa integra ferramentas para propor uma metodologia para testes de software que facilite a execução dos testes e o faça antes do fluxo de desenvolvimento de software, buscando propor agilidade ao detectar problemas logo no início do fluxo de desenvolvimento e redução de custos devido aos problemas o mais cedo possível no fluxo de desenvolvimento.

Palavras-chave: Testes de software. Metodologia. Cloud. Framework. Qualidade.

ABSTRACT: Designing and developing software with quality and consistency is a complex task. This research integrates tools to propose a methodology for software testing that facilitates the execution of tests and seeking to do it as soon as possible in the software development flow, seeking agility when detecting problems early in the development flow and reducing costs due to problems, as early as possible in the development stream.

Keywords: Software testing. Methodology. Cloud. Framework. Quality.

1 INTRODUÇÃO

Quando um software é idealizado, a expectativa é que ele desempenhe as funções para as quais ele foi projetado e especificado e que atenda às necessidades do usuário final, também é esperado que o software seja capaz de desempenhar essa função ao longo do tempo demonstrando confiabilidade.

Os defeitos existentes em um software ou não atendimento às necessidades dos clientes, na maioria das vezes, constituem-se em riscos tanto para

[1] wagner.ghedin@unesc.net
[2] agb@unesc.net

os negócios, como também para a imagem da empresa. Uma melhoria da área de desenvolvimento somente é insuficiente para que os resultados melhorem, se fazendo necessário uma melhoria de maturidade da área de teste de software também. (BASTOS et al, 2007)

A criação de uma metodologia para teste de software de uma maneira ágil e que garanta qualidade testando é importante dentro do contexto de fluxo de desenvolvimento.

Essa proposta poderia auxiliar os testes de software criando um fluxo de integração contínua de CI/CD para execução de teste de uma maneira automatizada que tenha uma implantação simples, podendo ser benéfica inclusive para projetos onde uma equipe é composta por poucas pessoas.

Cloud Computing é um tipo de computação na qual recursos dinamicamente escaláveis e geralmente virtualizados são providos como um serviço por meio da internet, estes recursos oferecem grande disponibilidade e apresentam uma maneira acessível de utilizar recursos computacionais sob demanda com a facilidade de poderem ser alocados por exemplo para uma tarefa específica e serem descartados após o uso. (Song, 2020)

Essas características se mostram interessantes dado que testes de carga, testes unitários e E2E (End-to-End) demandam uma grande quantidade de recursos que no momento de execução podem chegar a ser bastante expressivos, principalmente no que tange testes de carga. A execução destes testes também demanda uma quantidade de tempo para execução. Nesse sentido existe a oportunidade de utilizar a nuvem e sua capacidade de alocar dinamicamente e de maneira escalável recursos computacionais para executar testes.

Dentro do contexto de servidores de automação o Jenkins se destaca, sendo um servidor de automação open source, é amplamente utilizado por setores de DevOps para organizar fluxos de *deployment* automatizados, e pode ser uma forma ideal de suportar esta solução a ser proposta. (JENKINS, 2021)

Não foram encontrados trabalhos correlatos com objetivos semelhantes a este, o trabalho de conhecido como *Automated testing in the continuous delivery pipeline: A case study of an online company* fornece um estudo de caso de uma equipe utilizando um pipeline que inclui testes de validação, contudo sem o foco na implementação e dos recursos utilizados. (Gmeiner, Ramler and Haslinger, 2015)

A metodologia de automação no fluxo de *CI/CD* proposta neste trabalho pode ser utilizada para executar testes de performance automaticamente após um *deploy* a fim de validar se alguma alteração na aplicação causou degradação na performance do sistema, uma melhoria que está além de testes de performance. Neste caso, de maneira automatizada logo após o *deploy*, seriam obtidas métricas de desempenho e quantidades de erros dos testes automatizados.

Os objetivos específicos deste trabalho consistem em compreender as áreas de cloud computing e software testing; aplicar estrutura de contêineres para desenvolver uma metodologia para testes de software; propor uma metodologia para aplicação de testes de software em nuvem.

2 QUALIDADE DE SOFTWARE

Dentro do fluxo de desenvolvimento contínuo temos que o custo de encontrar problemas sobe quando mais tarde ele é detectado no processo de desenvolvimento, ficando ainda mais caro caso seja descoberto em produção, algo que é observado pela regra de 10 Myers (MPT.BR, 2020).

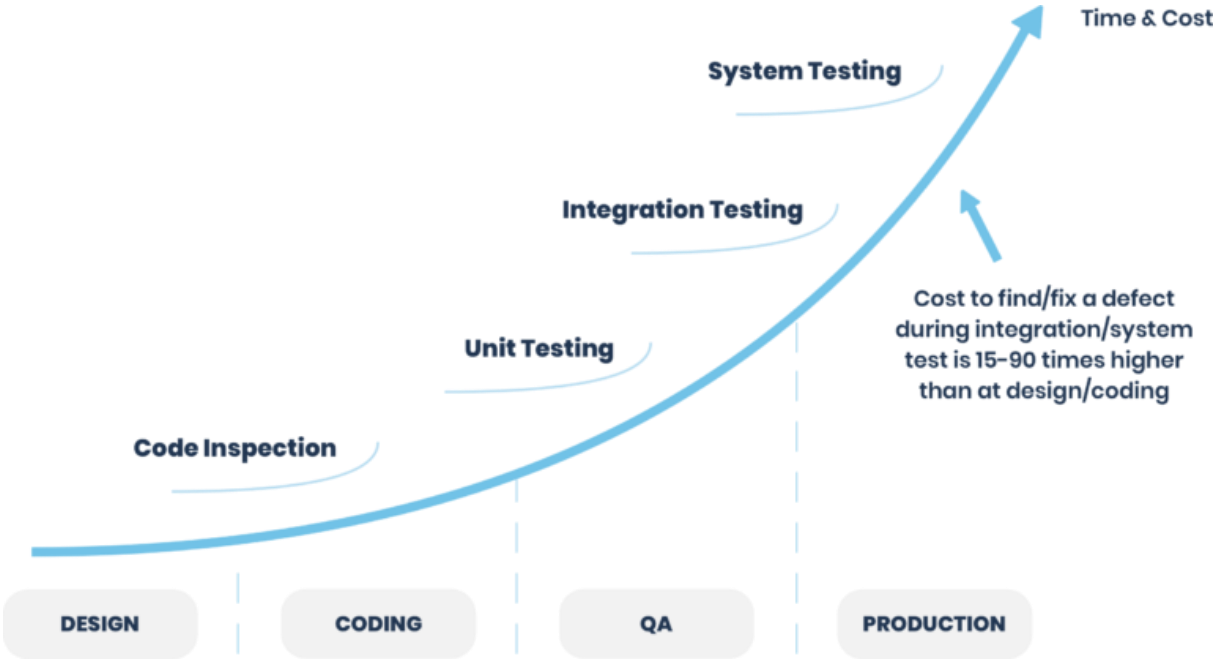

Figura 1: Regra de 10 Myers

Podemos observar neste caso a importância das validações ocorrem o quanto antes no fluxo, trazendo a maior parte possível do processo de teste e

validação para próximo do fluxo de desenvolvimento, de modo a reduzir custos e contribuir para um processo Ágil de entrega contínua.

2.1 FLUXO DE CI/CD

Um fluxo de CI/CD (*Continuous Integration e Continuous Deployment),* sendo uma prática bem estabelecida, em *Continuous Integration* membros de um time integram o seu trabalho várias vezes durante o dia fazendo merge, esta integração contínua faz uso de fluxos automatizados para aumentar a produtividade do time. Já dentro de *Continuous Deployment* tem o foco de estabelecer um ambiente pronto para produção após passar por validações automatizadas de testes e de qualidade, além de automatizar todo o fluxo de deploy da aplicação. (Shahin, Ali Babar and Zhu, 2017)

Para atingir o objetivo de manter a qualidade do software testado, propõe-se uma metodologia de teste de software que faça uso de um ambiente de deploy automatizado que consiga, logo após o deploy de uma aplicação, validar o estado das suas principais funcionalidades e a performance dela após alterações no código fonte.

Desta forma será aplicado o conceito de *Shift Left,* testando o software o quanto antes no fluxo de desenvolvimento e obtendo possíveis falhas e pontos de melhoria a fim de melhorar a qualidade e reduzir os custos.

O ciclo completo deste fluxo como a execução dos testes unitários, a execução dos testes E2E que idealmente compreenderia as principais funcionalidades do software, e os testes de desempenho ocorre após o ciclo de build e deployment.

3 MATERIAIS E MÉTODOS

O desenvolvimento uma metodologia para agilizar o teste de software em nuvem para aplicações web é uma pesquisa aplicada de base tecnológica, aplicando o Jenkins para criar um build que, além de gerenciar as configurações de deploy comuns, como: compilar, realizar *deploy* da aplicação e preparar banco de dados, iniciaria também os contêineres responsáveis pelo ambiente de execução dos testes (automatizados, teste de performance, testes unitários, testes de integração, testes de sanidade, entre outros), então os contêineres buscariam os scripts necessários para execução num repositório Git, iniciando a execução.

Os containers do Docker serão utilizados para facilitar a configuração do hardware, por terem a característica de serem máquinas virtuais executadas a partir de imagens, com toda a configuração e dependências que uma determinada aplicação necessita para executar de maneira rápida. (DOCKER, 2021)

Como pode ser observado na figura 2 o fluxograma completo da metodologia envolve um ciclo de CI/D, onde o foco na parte de testes de software inclui testes capazes de validar a performance do software desenvolvido de maneira ágil e assertiva, neste caso os testes realizados envolveram testes unitários, testes E2E e testes de performance de modo a fornecer dados suficientes para atestar o estado referente a qualidade.

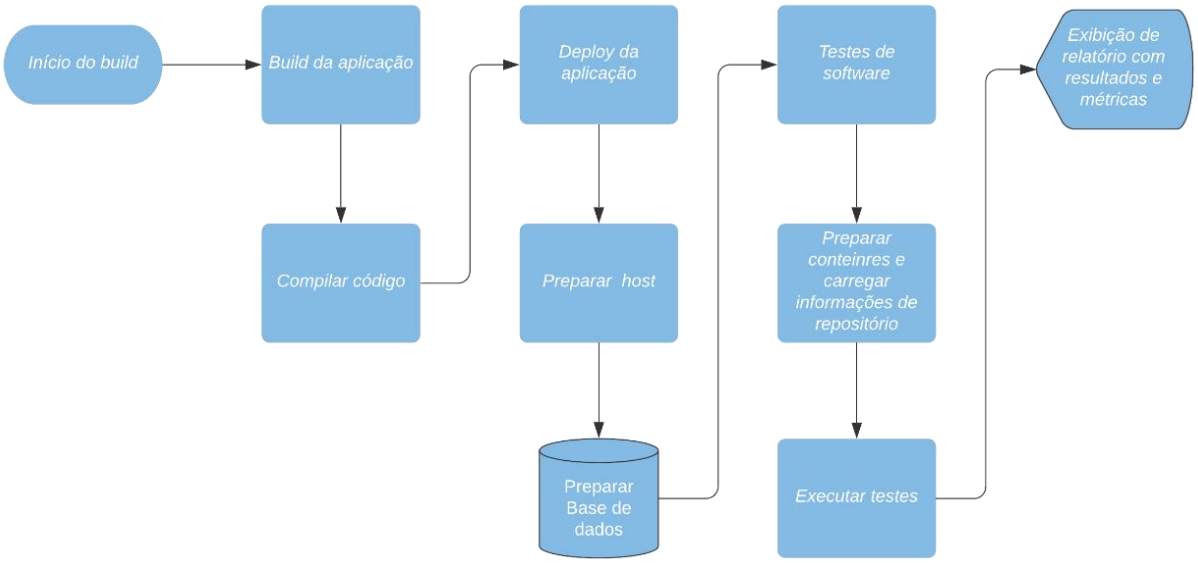

Figura 2: Fluxograma da metodologia proposta

3.1 CONFIGURAÇÃO DO DOCKER

A configuração do Docker para a implementação da metodologia requer a instalação de alguns componentes anteriores para a execução caso a máquina host seja Windows, será necessário o WSL2, uma camada de virtualização que permite a execução do kernel Linux dentro de uma máquina Windows, sistema operacional utilizado durante a execução desta pesquisa.

3.2 CONFIGURAÇÃO DO JENKINS

A configuração do Jenkins dentro do Docker é facilitada pela disponibilização de uma imagem de container dentro do Docker Hub, um repositório onde projetos open source ou proprietários disponibilizam imagens de container que podem ser facilmente instaladas no Docker.

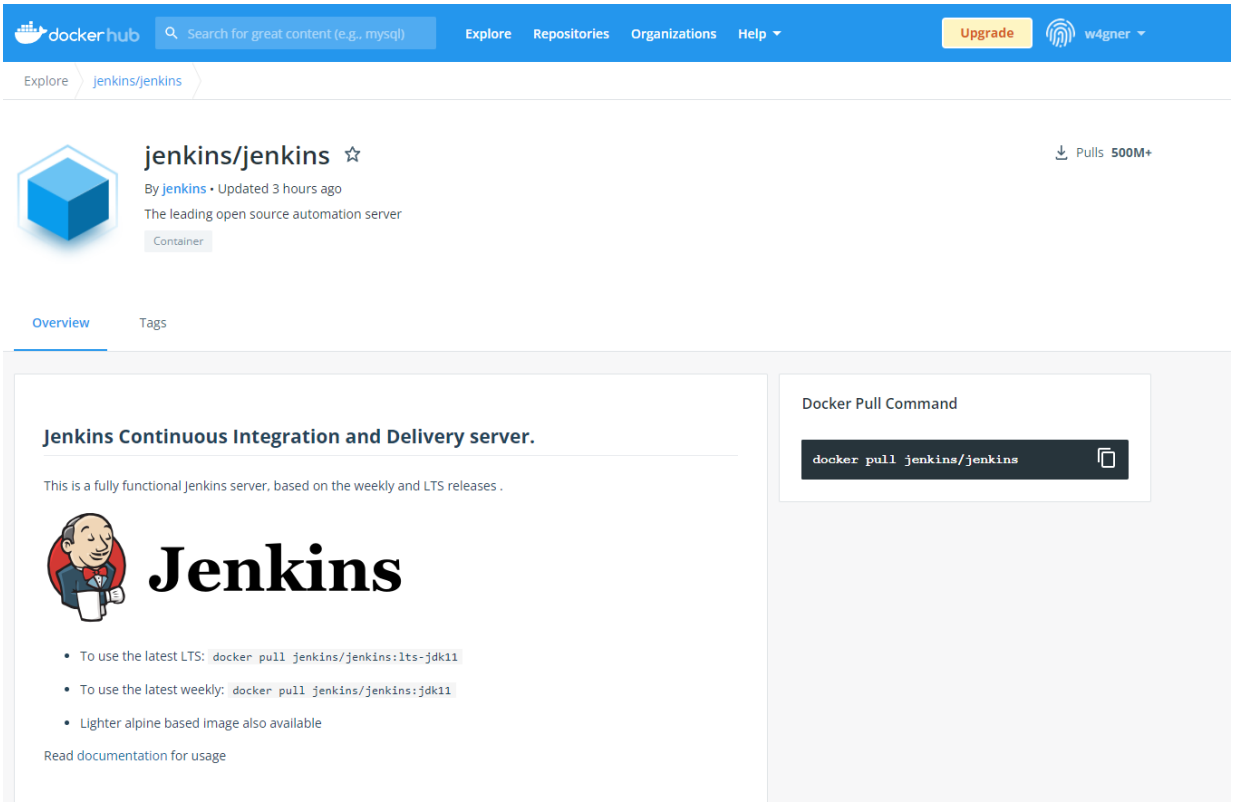

Figura 3: Imagem no Docker Hub para utilização do Jenkins

A partir da instalação da imagem no Docker é possível observar a imagem adicionada no Docker.

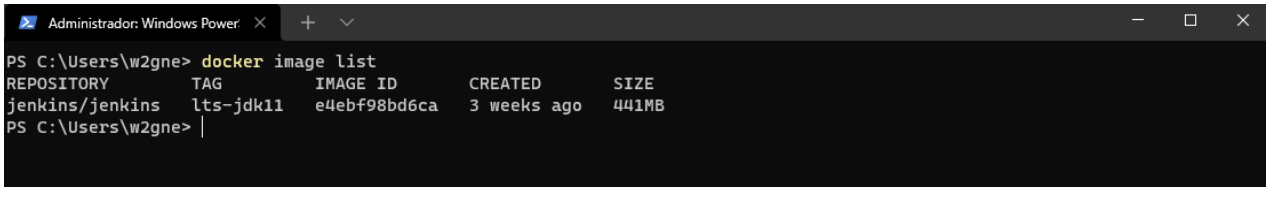

Figura 4: Imagem carregada no Docker

Após o download da imagem é possível inicializar um container com a imagem e iniciar a configuração do Jenkins, instalando os plugins necessários, e criando os usuários com acesso e demais configurações básicas.

Neste caso utilizar a imagem do Jenkins no Docker para suportar o ambiente se mostrou problemático, devido a necessidade de executar um container dentro de outro container, isto é possível conforme a própria documentação do Docker, função essa chamada de *Docker in Docker*, porém, como um dos objetivos seria manter uma estrutura simples que pudesse ser facilmente implementada e mantida, foi optado por instalar o Jenkins na máquina host sem uso de containers, desta forma o fluxo do será que o próprio Jenkins inicie o container Docker em um trabalho ou *job*, para administrar o container dentro de uma pipeline.

O Jenkins suporta a adição de plugins, alguns dos quais a instalação é recomendada por padrão. Um plugin utilizado é o plugin Performance, este acrescenta, entre outras coisas, integração para exibição de relatórios do Jmeter, função essa que será explorada mais a fundo no próximo tópico. Jmeter sendo uma das ferramentas open source mais utilizadas para testes de performance, e será utilizada para a criação dos scripts de performance e execução.

3.2.1 Configuração do Job no Jenkins

Com o Jenkins instalado na máquina host é necessário configurar o *Job* ou trabalho que executará a automatização dos fluxos desejados. Existem tipos de Jobs diferentes que podem ser criados no Jenkins, dois deles são abordados neste trabalho, sendo esses: Construir um projeto de software free-style, que será utilizado primariamente para configuração do ambiente dos testes de software, e Pipeline, que será abordado posteriormente.

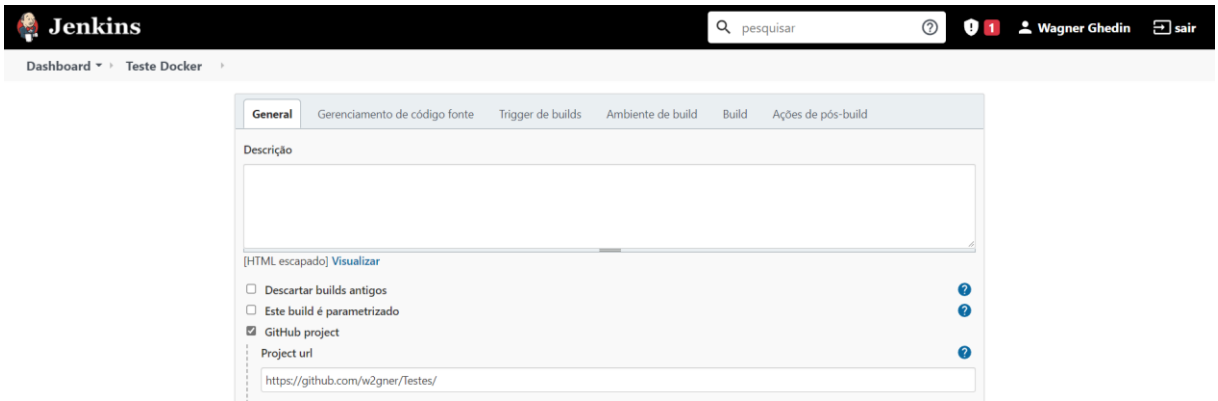

Figura 5: Configuração do build no Jenkins

A configuração é dividida em 6 categorias, sendo estas: configurações gerais, gerenciamento de código fonte, Trigger de builds, ambiente de build, Build e ações pós-build.

Existem configurações gerais do trabalho que englobam definições como a configuração de determinado número de dias para descarte dos logs de builds, inclusão de parâmetros para o build e a possibilidade de executar múltiplos builds em paralelo, para esta pesquisa, nenhuma delas foi utilizada.

Dentro do gerenciamento de código fonte foi realizada a configuração do repositório Git que armazena os scripts dos testes assim como as credenciais de autenticação e selecionada a branch correta. Com essas informações o Jenkins fará o gerenciamento de código fonte de maneira automática.

Em Trigger de builds é possível definir que este job execute em um intervalo de tempo específico, neste caso não será definido nenhum intervalo sendo realizado de forma manual. Contudo dentro de um pipeline onde um build é realizado de forma periódica esta opção poderia ser alterada.

Build permite a execução de scripts que podem ser executados para realizar a ação desejada, nesta etapa executamos os comandos necessários para que os scripts de testes sejam executados.

Neste caso foi escolhido fazer o uso de uma imagem do Jmeter para poder executar um script de um teste de performance como uma forma de demonstrar a metodologia, para tanto foi necessário baixar a imagem demonstrada na figura 6 abaixo. Nesta etapa algo que se mostrou mais problemático foi acessar arquivos do container, dado que ele é executado em uma virtualização em ambiente isolado e existe a necessidade de passar o arquivo de teste para dentro do container, executá-lo e então devolver o relatório para a máquina host para que o mesmo possa ser exibido posteriormente.

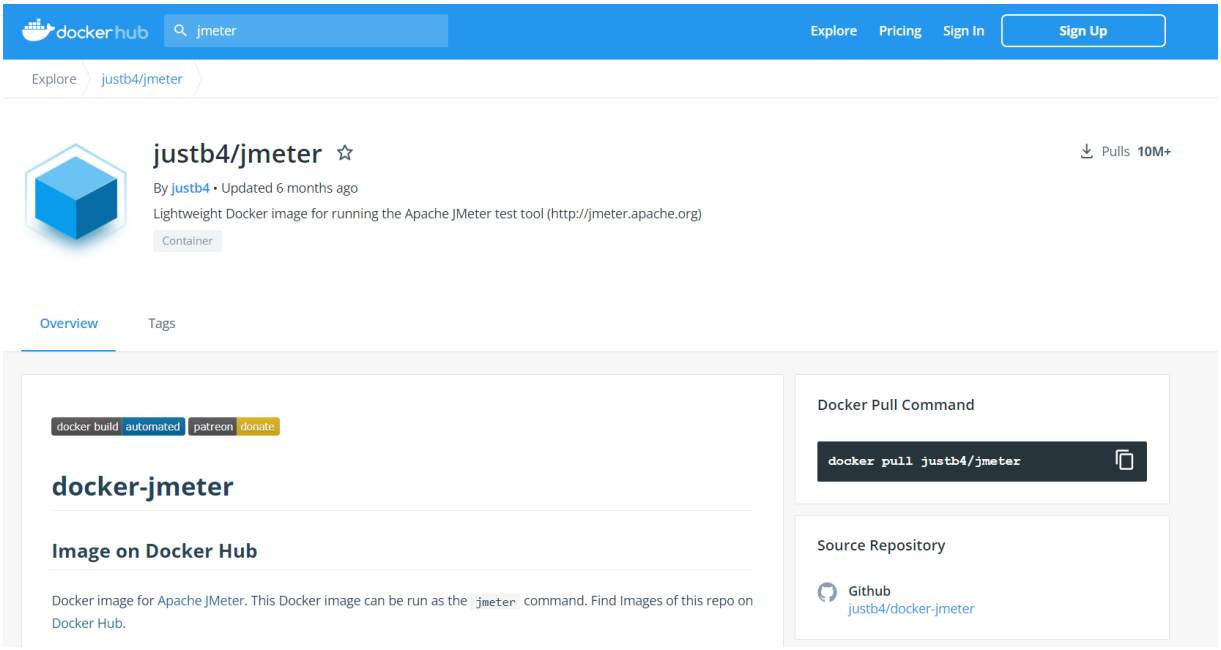

Figura 6: Imagem do Jmeter no Docker hub

Uma solução encontrada para este caso foi inserir um comando para navegar para o diretório do Jmeter no host local e utilizar uma função do Docker que permite vincular um diretório local com um diretório do container como uma forma de enviar arquivos para o container que executará o teste, e retornar o para o host local.

Como pode ser observado na figura 7, a pasta do Jmeter na máquina host foi vinculada a mesma pasta dentro do container, de modo que, ao executar o teste de performance, o arquivo do script clonado do repositório Git para esta pasta específica ficará disponível para ser acessada dentro do container, e como um espelho, ao ser executado o teste e salvo o arquivo de relatório do Jmeter, salvo nesta mesma pasta dentro do container, será acessível também pelo host.

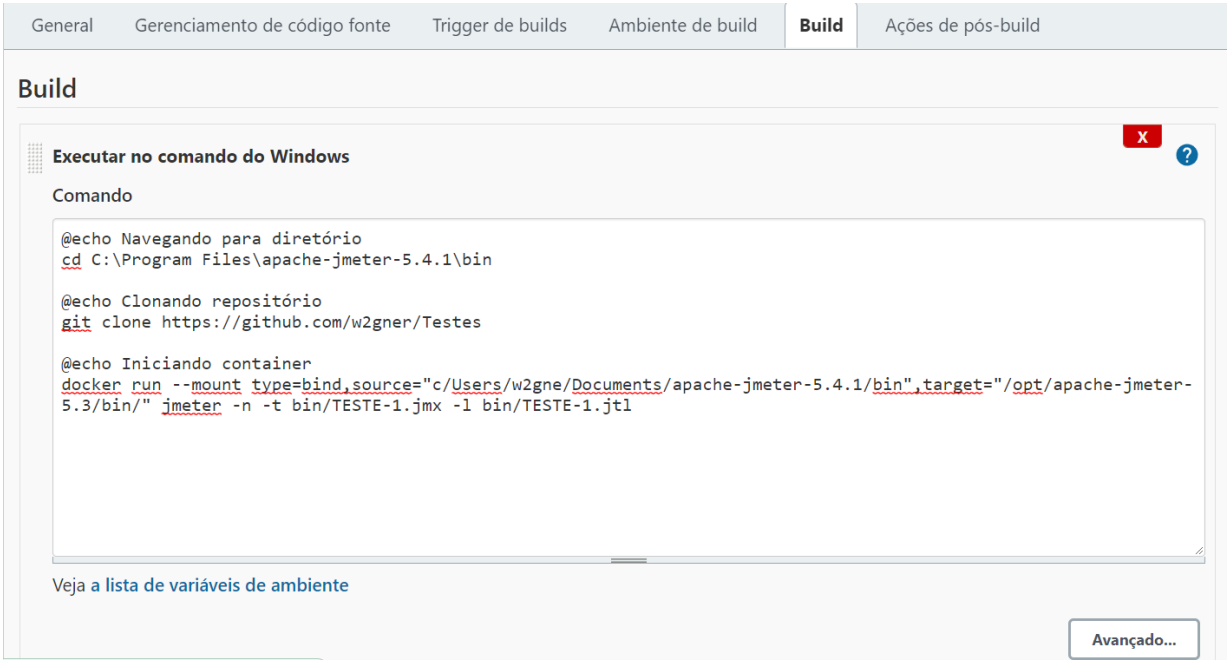

Figura 7: Script para execução do container do Jmeter

Com a configuração do build realizada, existe a possibilidade de ações a serem realizadas posteriormente ao build, o arquivo de relatório do relatório gerado após a execução do teste de performance do Jmeter pode ser inserido aqui, de modo a gerar um relatório posteriormente. Esta função é disponibilizada pelo plugin de Performance mencionado anteriormente, mas todas as suas funções serão apresentadas a seguir, nesta etapa a configuração do plugin é simples, e envolve o mapeamento do arquivo de relatório gerado pelo Jmeter, como pode ser observado na figura 8 abaixo.

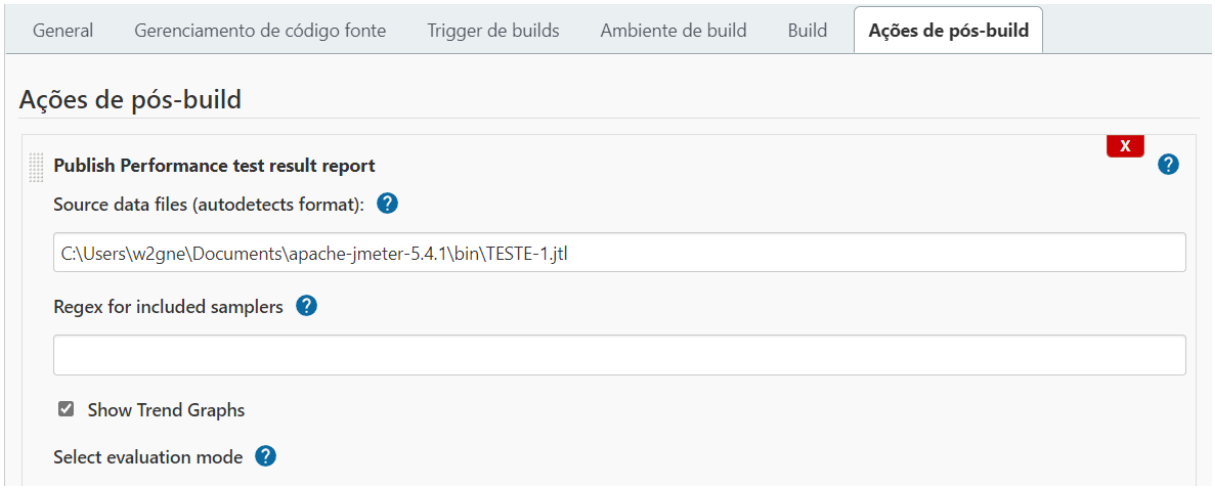

Figura 8: Configuração do relatório

3.2 EXECUÇÃO DE BUILD NO JENKINS

Com todas as configurações realizadas, a execução do job no Jenkins é realizado como definido na figura 9 abaixo.

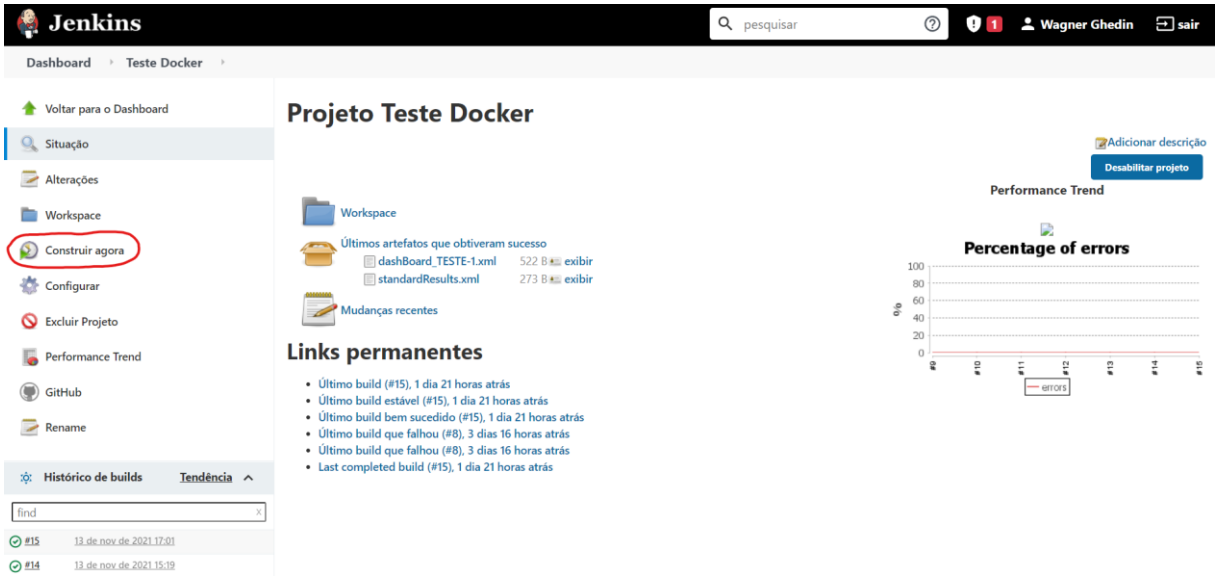

Figura 9: Execução do job

Após a execução do build é o resultado é exibido no histórico de builds, onde ficam consolidados os dados extraídos do relatório do Jmeter através do plugin de performance, exibindo a variação de percentagem de erros, vazão e tempo médio de resposta conforme os builds.

Este Job deve ser integrado no pipeline de automatização utilizado para deploy da aplicação, o Jenkins fornece essa possibilidade, na figura 10 podemos observar uma simulação do deploy, e a execução dos testes anteriormente criados e configurados, desta forma assim quando o deploy finalizar são disponibilizados todos os dados relativos à qualidade, métricas de desempenho neste caso, porém idealmente junto aos status de todos os testes executados.

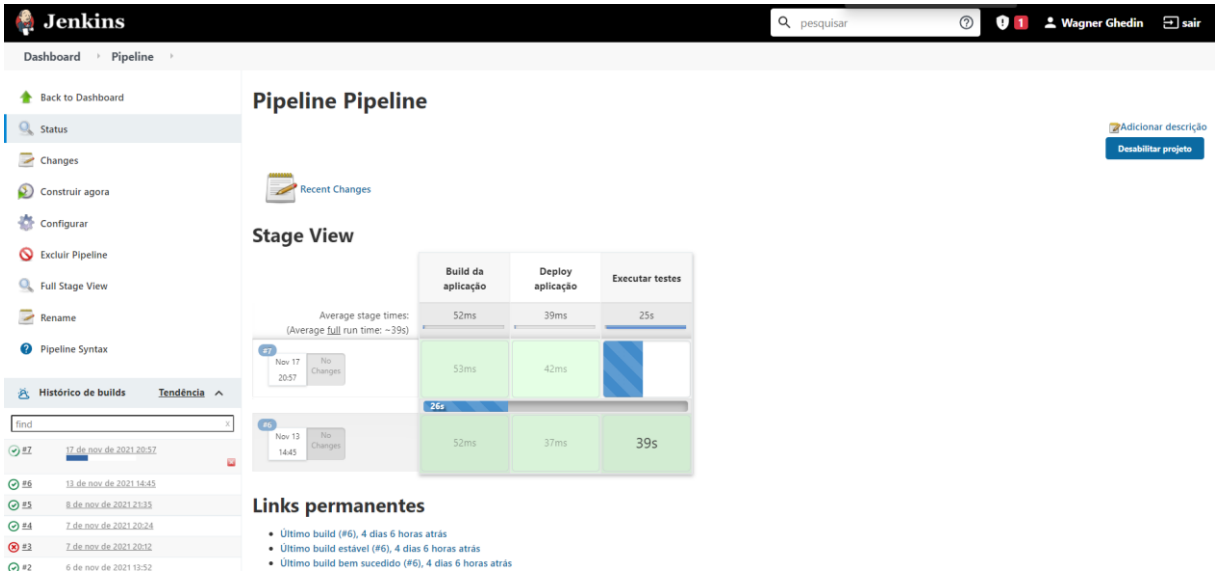

Figura 10: Execução dos testes dentro do fluxo de deploy

Todas as ferramentas foram integradas no host local, porém o modelo proposto deste fluxo é ser executado na nuvem utilizando cloud computing, de modo a permitir que as pessoas que necessitem tenham acesso à ferramenta e aos dados gerados como forma de avaliar os resultados dos builds e testes realizados, além dos benefícios apontados anteriormente inerentes ao uso desta tecnologia.

4 RESULTADOS E DISCUSSÃO

Como resultado deste trabalho foi proposta uma metodologia como forma de executar testes dentro do fluxo de desenvolvimento contínuo (CI/CD), também foi compreendida a área de cloud computing e software testing, sendo um dos principais objetivos da metodologia proposta a execução de testes o mais próximo possível do próprio desenvolvimento do software, neste caso a execução dos testes sendo automática dentro do próprio processo de build de uma aplicação.

São geradas métricas após as execuções dos testes que permitem o acompanhamento da perfomance da aplicação, no caso demonstrado o jmeter compila várias métricas, entre elas a taxa de erros, tempo médio de resposta e vazão, essas métricas podem ser comparadas entre as execuções anteriores, por exemplo, caso entre o build 13 e o build 14 seja apresentada uma variação negativa na vazão é possível que a alteração realizada tenha diminuido a capacidade da aplicação, a taxa de erros indica se existe algum comportamento inesperado e o tempo médio de

resposta, semelhante a vazão, indica se houve diminuição na perfomance com a qual a aplicação lida com as requisições.

É possível estabelecer qual variação das métricas mencionadas seria um impeditivo para um deploy, caso por exemplo a variação de vazão da aplicação caia mais de 15% poderia ser implementado um processo de impedimento de deploy, onde as alterações são revertidas para a versão anterior com a justificativa de que a alteração causou uma degradação significativa. O valor a ser definido precisa levar em consideração que no período do teste podem ocorrer leves variações no ambiente, 15% seria um valor teórico, sendo necessária uma mais discução para estabecer um valor que consigo capturar degração de perfomance com melhor precisão.

Todas as ferramentas utilizadas durante esta pesquisa demonstraram a capacidade de facilitar de maneira relevante fluxos complexos, como a metodologia de um ambiente de integração contínua e desenvolvimento.

O Docker particularmente mostrou uma ferramenta poderosa, sendo possível armazenar dentro de uma imagem todos os recursos e dependências necessárias para execução de um software. O seu uso no entanto neste caso em particular pode ter sido limitado pois não foi possível suportar toda metodologia proposta em um ambiente de containers, limitando-se apenas a execução do Jmeter.

A caraterística de teste distribuído não foi abordada devido ao fato de o foco estar na aplicação e consolidação de ferramentas com os objetivos específicos desta pesquisa. Assim como o foco em demais testes que poderiam ser executados se valendo desta estrutura não foram abordados em detalhes.

5 CONCLUSÃO

Nesta pesquisa foi pesquisada, implementada e proposta uma metodologia que utiliza dos conceitos de cloud computing e software testing de modo a criar um fluxo de integração contínua que contribua para garantia de qualidade do software desenvolvido com o teste realizado, aplicando também containers para execução dos scripts de testes.

Embora os resultados obtidos tenham sido positivos, o fluxo completo não foi implementado, sendo necessário um aprofundamento para validação dos resultados com métricas mais claras e objetivas.

Como resultado foi criado um fluxo de testes inserido em um fluxo de intrega contínua, após a execução dos testes as métricas geradas podem ser utilizadas para analisar e comparar com builds anteriores, estabelecer se houve degradação tendo as informações e a visibilidade de como a aplicação se comportou, isto gera um valor importante e auxiliará a detectar problemas e falhas mais cedo, são resultados importante mas que dada a evolução da pesquisa pode não ter resultado em um metodologia completa com todas as características.

A área de cloud computing embora pesquisada não foi implementada no trabalho, ainda que não tenha sido um objetivo, faria sentido no contexto do deste trabalho.

Além do trabalho desenvolvido, algumas lições foram aprendidas com esta pesquisa, em particular ao desenvolvimento, e a importância de conseguir estabelecer os objetivos claros desde o início seguindo um cronograma.

Com base no aprendizado sugere-se para trabalhos futuros: criar integrações com ferramentas de APM para mostrar os estados dos serviços durante a execução dos testes no ambiente; elaboração de um relatório detalhado e intuitivo com os dados de execução; criação de uma imagem que inclua todas as ferramentas utilizadas nesta pesquisa, de modo a deixar o processo de implementação da metodologia proposta mais simples.

REFERÊNCIAS

JENKINS. **Build great things at any scale**. Disponível em: https://www.jenkins.io/. Acesso em: 14 nov. 2021.

DOCKER. **Use containers to Build, Share and Run your applications**. Disponível em: https://www.docker.com/resources/what-container. Acesso em: 14 nov. 2021.

MPT.BR. **MPT – melhoria de processo de teste brasileiro.** Disponível em: http://www.mpt.org.br/wp-content/uploads/2010/12/MPT_BR_Nivel_1_v_2.2.pdf. Acesso em: 28 nov. 2020.

BASTOS, Anderson et al. **Base de conhecimento em teste de software**. São Paulo: Martins Fontes, 2007.

GMEINER, J.; RAMLER, R.; HASLINGER, J. **Automated testing in the continuous delivery pipeline: A case study of an online company**2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops

(ICSTW). **Anais**...IEEE, 13 abr. 2015Disponível em: <http://ieeexplore.ieee.org/document/7107423/>

SHAHIN, M.; ALI BABAR, M.; ZHU, L. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. **IEEE Access**, v. 5, p. 3909–3943, 2017.

SONG, H. H. **Testing and evaluation system for cloud computing information security products**Procedia Computer Science. **Anais**...Elsevier B.V., 1 jan. 2020. Acesso em: 7 maio. 2021