UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

MARLO HENRIQUE DE LIMA OLIVEIRA

CLOUD TEST: MELHORIA DA CONFIABILIDADE DE TESTES MOBILE COM A UTILIZAÇÃO DE COMPUTAÇÃO EM NUVEM

RUSSAS

2021

MARLO HENRIQUE DE LIMA OLIVEIRA

CLOUD TEST: MELHORIA DA CONFIABILIDADE DE TESTES MOBILE COM A UTILIZAÇÃO DE COMPUTAÇÃO EM NUVEM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de software do Campus de Russas da Universidade Federal do Ceará, como requisito à obtenção do grau de bacharel em Engenharia de software.

Orientador: Prof. Ms. Filipe Maciel Roberto

RUSSAS

2021

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

O48c Oliveira, Marlo Henrique de Lima.
Cloud Test : melhoria da confiabilidade de testes mobile com a utilização de computação em nuvem / Marlo Henrique de Lima Oliveira. – 2021.
67 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Engenharia de Software, Russas, 2021.
Orientação: Prof. Me. Filipe Maciel Roberto.

1. Automação de teste. 2. Dispositivos móveis. 3. Teste mobile em nuvem. 4. Ferramentas de teste mobile. I. Título.

CDD 005.1

MARLO HENRIQUE DE LIMA OLIVEIRA

CLOUD TEST: MELHORIA DA CONFIABILIDADE DE TESTES MOBILE COM A UTILIZAÇÃO DE COMPUTAÇÃO EM NUVEM

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de software do Campus de Russas da Universidade Federal do Ceará, como requisito à obtenção do grau de bacharel em Engenharia de software.

Aprovada em:

BANCA EXAMINADORA

Prof. Ms. Filipe Maciel Roberto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dra. Jacilane de Holanda Rabelo
Universidade Federal do Ceará - (UFC)

Prof. Ms. José Osvaldo Mesquita Chaves
Universidade Federal do Ceará - (UFC)

Dedico esse trabalho a minha avó Maria Arlete (in memorian), com todo o meu amor e gratidão por sempre acreditar em mim.

AGRADECIMENTOS

Aos meus pais, irmãos, tios, avós, primos por toda a confiança e amor e por terem me ajudado a chegar onde estou hoje.

Ao meu irmão Jorge Michel, por ter me ajudado a tomar sempre as decisões corretas quando elas surgiram.

A meu orientador Prof. Ms. Filipe Maciel Roberto, por ter acreditado e aceitado meu tema de pesquisa, e ter me guiado em momentos de dificuldades.

A toda a equipe da Greenmile, em especial, André Campos e Régis Melo por terem dado apoio a realização desta pesquisa.

Aos amigos que a Universidade Federal do Ceará me proporcionou e que sempre estiveram ao meu lado nos altos e baixos, em especial: Artur de Castro, Ana Lara, Cibele Rodrigues, Cleiton Monteiro, Herverson de Sousa, Humberto Damasceno, Mateus Franco, Susana Moreira, e Williana Leite, nossas noites em claro não foram em vão, obrigado por toda a ajuda e motivação, sempre lembrarei de vocês!

Aos amigos(a) Alexandro, Daniel, Eliana e Fernanda Carla por serem amigos em vários momentos.

A todos os professores, servidores e colaboradores da Universidade Federal do Ceará - Campus Russas, por terem proporcionado uma experiência incrível no curso de engenharia de software e me preparar para a vida.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do template utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

Por fim, a todos que participaram diretamente ou indiretamente da minha formação nesses quatro anos de graduação.

Muito obrigado a todos.

“Só é digno da liberdade, como da vida, aquele que se empenha em conquistá-la.”

(Johann Goethe)

RESUMO

A execução de testes automatizados para validação das principais funcionalidades de uma aplicação frequentemente é um processo realizado utilizando uma infraestrutura local, principalmente quando se busca a garantia de qualidade realizando testes em dispositivos reais. O processo de manter uma infraestrutura local demanda mão de obra especializada e recursos financeiros, uma vez que *smartphones* estão em constante reciclagem, diante da ascensão e lançamento de novos modelos no mercado. Logo os desenvolvedores deparam-se com uma situação: garantir o desempenho e execução do seu aplicativo em diferentes modelos de *smartphones* atualmente em uso, enquanto que entregam eficiência de execução dos testes. Diante deste cenário, é cada vez mais evidente a busca por novos métodos de execução de testes mobile. É nesse contexto que a utilização de ambientes de computação em nuvem para a execução de testes mobile utilizando dispositivos reais ganha destaque. Este trabalho apresenta um estudo comparativo de testes mobile entre os ambientes local e de nuvem utilizando dispositivos reais para testes. Para realização dessa comparação foi construído um projeto de testes mobile utilizando ferramentas como Selenium(*Framework* para automação de testes), Appium(*Framework* para automação de testes mobile), TestNG(*Framework* de execução de testes) e linguagem de programação Java para um aplicativo Android. Foi selecionado um ambiente de nuvem para comparação com o ambiente local. Logo se realizou uma série de execuções utilizando o mesmo dispositivo real em ambos os ambientes, sendo colhido os resultados de execução ao final de cada ciclo. Por fim, foi calculado desvio padrão, média dos resultados e o intervalo de confiança de 95 % para o experimento com o propósito de verificar os ganhos obtidos com a utilização de ambiente de nuvem para testes.

Palavras-chave: Automação de teste. Dispositivos móveis. Teste mobile em nuvem. Ferramentas de teste mobile.

ABSTRACT

The execution of automated tests to validate the main functionalities of an application is often a process carried out using a local infrastructure, especially when seeking quality assurance by performing tests on real devices. The process of maintaining a local infrastructure requires specialized labor and financial resources, since smartphones are constantly being recycled, given the rise and launch of new models in the market. Soon, developers are faced with a situation: guaranteeing the performance and execution of their application in different models of smartphones that are currently in use, while delivering efficiency in the execution of tests. In the face of this scenario, the search for new methods of executing mobile tests is increasingly evident. It is in this context that the use of cloud computing environments to perform mobile tests using real devices is highlighted. This work presents a comparative study of mobile tests between the local and cloud environments using real devices for testing. To carry out this comparison, a mobile testing project was built using tools such as Selenium (Framework for testing automation), Appium (Framework for automation of mobile tests), TestNG (Framework for testing execution) and Javascript programming language for an Android application. A cloud environment was selected for comparison with the local environment. Then a series of executions was carried out using the same real device in both environments, with the execution results being collected at the end of each cycle. Finally, standard deviation, average of results and 95% confidence interval for the experiment were calculated in order to verify the gains obtained with the use of cloud for testing.

Keywords: Test automation. Mobile devices. Mobile cloud testing. Mobile testing tools

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

AM-TaaS	*Automated mobile testing as a service*
AOP	*Programação Orientada a Aspectos*
API	*Interface de programação de aplicações*
AWS	*Amazon Web Services*
E2E	*End-To-End*
FCFS	*first come, first served*
GPC	*Google Cloud Platform*
GUI	*Graphical User Interface*
IaaS	*Infraestrutura como Serviço*
JSON	*JavaScript Object Notation*
PaaS	*Plataforma como Serviço*
SaaS	*Software como Serviço*
TaaS	*Teste como Serviço*
TI	*Tecnologia da Informação*
VPN	*Rede Privada Virtual*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivação

O teste de *software* é uma atividade desafiadora para muitos projetos, envolve planejamento, implementação e execução do software com dados, análise de suas saídas, de modo a verificar o comportamento do *software* (SOMMERVILLE, 2011). A quantidade de casos de testes, os passos que são executados para realização do procedimento, pode ir de algumas centenas em aplicações mais simples, a casa dos milhares em sistemas de larga escala.

Por meio da automação, o tempo gasto com a execução dos testes é considerado inferior quando comparado a um processo de teste manual, podendo ser realizado de forma rápida, sem a necessidade de um testador no processo executado(VIEIRA *et al.*, 2018). Testes manuais, que poderiam levar inúmeras horas, tem tempo reduzido e são executados em minutos, quando automatizados.

Apesar dos benefícios, inúmeros desafios devem ser considerados na implementação de testes automatizados em um projeto, tais como: maior esforço por parte dos desenvolvedores para implementar casos de teste automatizados; manutenção dos scripts de teste; custos iniciais mais elevados comparados a execução de testes manuais (RAFI *et al.*, 2012). No entanto, à medida que cresce o número de testes automatizados a serem executados, se faz necessário um ambiente com configurações e poder computacional adequado para uma execução mais rápida (FEWSTAR; GRAHAM, 1999).

O teste *End-To-End* (E2E), modelo de teste que interage com a *interface web* no navegador ou aplicação móvel, é mais lento que os testes realizados diretamente no código da aplicação, devido a necessidade de iteração constante com a estrutura de UI da aplicação (FOWLER, 2012). Os testes E2E são vistos como uma das principais categorias de testes a serem automatizados no contexto de aplicações mobile e *web*. Com o aumento do número de casos de testes, o tempo da execução aumenta consideravelmente, podendo ser proporcional à quantidade de passos executados nos cenários de execução do teste e das verificações que são realizadas na *interface* da aplicação (BRUNS *et al.*, 2009).

1.2 Especificação do problema

Atualmente, muitas empresas ainda realizam a execução dos seus testes automatizados mobile em um ambiente local, utilizando equipamentos e infraestrutura da própria organização, podendo ocasionar uma baixa eficiência no processo de execução de testes.

Um caso real de problema de infraestrutura para testes acontecia recorrentemente na empresa Greenmile, onde a suíte de testes automatizados da aplicação GM Driver7 é executado em um computador Mac mini, utilizando um hub para conexão, comunicação e execução dos testes em múltiplos devices.

Com o decorrer do tempo se faz necessário reciclar a infraestrutura existente ou adquirir equipamentos com poder computacional superior, para que haja uma melhoria desse processo de execução, no entanto, esse é um processo que demanda recursos financeiros e mão de obra especializada.

1.3 Solução para o problema

O uso da computação em nuvem como um laboratório virtual de teste mobile, torna possível fazer uso de diversas combinações de infraestrutura e dispositivos, além de servir como uma ferramenta que possibilita potencializar a profundidade, amplitude bem como a possibilidade de se escalar testes de *software*. Entretanto, desenvolver e processar testes em um ambiente de nuvem requer esforços na configuração do ambiente, distribuição e execução de testes (HAVANA, 2010).

O uso de dispositivos móveis baseados em uma infraestrutura de nuvem abre espaço para que o usuário aloque somente os recursos que ele necessita para o momento de execução, tornando possível acelerar o processo de execução de testes em dispositivos móveis virtualizados ou reais. Realizar testes de aplicativos móveis utilizando computação em nuvem representa uma evolução na forma de se testar (PARVEEN; TILLEY, 2010).

1.4 Objetivo geral

O objetivo principal deste trabalho é apresenta o nível de confiabilidade de testes mobile entre os ambientes local e de nuvem utilizando dispositivos reais.

1.5 Objetivo específico

Como objetivos específicos, tem-se:

- Mensurar os ganhos obtidos utilizando ambiente de nuvem para execução de teste mobile;
- Demonstrar a integração entre execução de testes automatizados e serviço de computação em nuvem;
- Apresentar o intervalo de confiança obtido da utilização de testes mobile em nuvem;

1.6 Justificativa

A computação em nuvem é uma das quatro tendências de transformação digital com altas perspectivas de adoção (SKYONE, 2017). As empresas têm procurado serviços ofertados por nuvens públicas, buscando suprir a demanda de recursos necessários de aplicações móveis e *web* através de *Infraestrutura como Serviço* (IaaS). Como principais justificativas na busca por essa modalidade de serviço está a redução de custo que se tem com infraestrutura adquirida, e pela alta disponibilidades dos serviços contratados.

Em cenário de teste de *software* mobile, dispositivos para execução de teste são um custo que as empresas têm recorrentemente, com a renovação da gama de dispositivos necessários para o ambiente. Por exemplo, se uma empresa deseja adicionar o IPhone 12 MAX aos dispositivos da suíte de testes local, haveria um custo aproximado de 10.999 reais para a inclusão desses dispositivo para o modelo de 256GB de armazenamento (APPLE, 2020).

Fazer uso de infraestruturas já adquiridas por meio da computação em nuvem para a execução de testes mobile é uma possibilidade de corte de custos, com o uso de aparelhos virtualizados ou reais sob demanda. Outros pontos benéficos da utilização podem ser vistos na facilidades de construção e preparação dos recursos necessários, maior realismo com a possibilidade de simulação de picos de tráfego inesperado e redução de custos em todo o processo de PD necessário para a execução de testes (Jun; Meng, 2011).

Outro ponto importante é, o aperfeiçoamento que pode ser adquirido na execução dos testes mobile em um ambiente de nuvem. A possibilidade de se executar diversos dispositivos virtualizados ou reais, distribuindo a execução dos testes, aumentando o desempenho da execução e consequentemente obtendo melhores resultados no tempo gasto para realização dos testes.

1.7 Estrutura do trabalho

O trabalho de conclusão de curso está estruturado da seguinte forma: O primeiro capítulo apresenta uma introdução ao problema. O segundo capítulo apresenta a fundamentação teórica e alguns trabalhos relacionados necessários para o entendimento do restante do trabalho, apresentando conceitos relacionados à computação em nuvem, teste automatizados, virtualização e testes em nuvem. No terceiro capítulo é apresentado a proposta de solução para o problema. O quarto capítulo apresenta a modelagem e o processo de execução dos testes. no quinto capítulo é apresentado os resultados obtidos no trabalho. Por fim, no sexto capítulo é apresentados as conclusões finais deste trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

O teste de *software* é algo presente no desenvolvimento de sistemas, envolve planejamento, escrita dos cenários e casos de testes, construção utilizando ferramentas e a sua execução. No princípio de um projeto de teste, poucas funcionalidades testadas e um baixo nível de cobertura do código proporcionam uma execução rápida e sem grande necessidade de um alto poder computacional. Com o passar do tempo, novas funcionalidades são acrescentadas ao sistema, cresce o número de casos de teste para funcionalidades existentes e sobe o nível de cobertura do código. Surge então os primeiros problemas relacionados a eficiência da execução de testes de *software*.

Essa baixa eficiência pode estar relacionada a vários fatores, entre os que podem ser apontados se tem: baixo poder computacional e a ausência de um ambiente de execução distribuído (EMERY, 2009). Alguns analistas de qualidade têm buscado aumentar a eficiência de execução do ambiente de testes, adquirindo mais memória, processamento ou um equipamento completo com mais poder computacional. Outra forma vista para melhorar a eficiência está em aumentar a quantidade de dispositivos móveis para que múltiplos testes possam ser executados ao mesmo tempo, através de um cenário de execução de testes funcionais mobile distribuído. Ambos os pontos são dependentes de recursos financeiros para serem aplicados, sendo assim ambientes de nuvem sob demanda que forneçam esses recursos para o usuário podem ser vistos como alternativa de solução para este ponto.

Os problemas de mau planejamento de uma suíte de testes mobile podem resultar em complicações e falta de eficiência no momento de execução. Muitos projetos de teste de *software* são constituídos inicialmente de um único processo que executa toda a pilha de teste de modo sequencial, devido as dificuldade que as equipem enfrentam para ter disponível um ambiente escalável e seguro a nível de requisitos (TAN, 2020). Com a inclusão do paralelismo, haveria a necessidade de gerenciar múltiplos trabalhos executados simultaneamente, bem como garantir que os recursos disponíveis serão devidamente balanceados na quantidade de trabalho que recebem e se certificar sobre a comunicação entre a aplicação e os recursos disponíveis no momento de execução. Além dos pontos anteriores é preciso assegurar que não haverá condição de corrida entre recursos, o que impactaria diretamente no sucesso da execução.

Suítes de testes mobile, que executam de forma totalmente distribuída, podem ter perdas de desempenho devido a uma variável, a ordem de execução dos casos de teste. A maioria dos projetos de teste de *software* segue um modelo Flow-Shop, o que representa dizer

que independente do número de execuções do projeto, a ordem de execução será sempre a mesma. Ferramentas de programação já permitem determinar a ordem de execução dentro do projeto, o que se busca é agrupar testes que fazem parte de um mesmo cenário, respeitando o critério de independência entre os testes, permitindo diminuir o tempo ocioso dos equipamentos e, consequentemente, o tempo total de processamento.

Nas secções seguintes para complementar a temática do problema abordado na pesquisa e, posteriormente, modelar uma proposta de solução, serão apresentados alguns trabalhos relacionados e conceitos utilizados nas áreas de teste de *software* e computação em nuvem.

2.1 Trabalhos relacionados

(ROJAS *et al.*, 2016) apresentaram um resumo dos principais serviços e tecnologias para teste mobile em nuvem, e propuseram a arquitetura e implementação de uma estrutura de teste móvel baseada em nuvem chamada *Automated mobile testing as a service* (AM-TaaS). O AM-TaaS é constituído por um conjunto de camadas, que se divide em: *interface* do usuário, camada de informações e camada de recursos. Por meio dessas camadas os autores puderam abstrair a complexidade de toda estrutura necessária para que fosse possível executar testes mobile em nuvem. Os autores propuseram um modelo de serviço baseado em uma arquitetura local com a camada de recursos baseada na plataforma Android por meio do SDK *Manager* (Software Development Kit) para gerenciamento dos emuladores usados na execução dos testes, no entanto, não apresentaram a eficiência dessa proposta de *framework* em relação a ambientes já existentes, ou mesmo os resultados obtidos entre arquitetura local e arquitetura em nuvem.

(Guo *et al.*, 2018) demonstraram um modelo de nuvem hibrida móvel para testes de aplicativo mobile chamado FeT, que busca atender as necessidades impostas para a validação de aplicações bancárias em diferentes dispositivos através de teste de aceitação do usuário e teste de integração do sistema. A solução consiste em um modelo híbrido, com uma parte do sistema em nuvem pública e o restante em uma nuvem privada, sendo conectados por um túnel de uma *Rede Privada Virtual* (VPN). Esse modelo hibrido permitiu que os autores dividissem a nuvem em zonas de testes, determinando as categorias de testes que poderiam ser agrupadas em uma determinada zona. Os autores propuseram essa solução em um modelo de três camadas de componentes, integrados por meio do gerenciamento de teste de aplicativo, serviço de teste de aplicativo e agente de teste de aplicativo, cada umas dessas camadas podendo ter variantes internas. Os autores validaram a solução desenvolvida executando rotinas de testes, apresentaram

dados que demostram que a utilização de um modelo hibrido com a divisão da nuvem em zonas para execução de diferentes tipos de teste pode acelerar o processo de liberação de aplicativos em diferentes dispositivos moveis.

(Ma *et al.*, 2016) Propuseram o BugRocket, um ambiente que combina testes distribuídos e automação de testes mobile integrados a uma nuvem de testes escalável e flexível. O BugRocket utiliza um conceito de exploração automática da *Graphical User Interface* (GUI) da aplicação para a geração de *scripts* de teste automaticamente. A geração é baseada em técnicas de engenharia reversa e por meio da utilização de dados de configuração do usuário. Os testes são executados em um servidor mestre que é responsável por geração de relatórios, interface para o usuário e distribuição dos testes para um conjunto de máquinas escravas, conectadas a um conjunto de dispositivos móveis virtualizados. Os autores indicaram que uma alta capacidade de dispositivos móveis diferentes conectados aos escravos aumentaria o nível de detecção de erros, no entanto, não apresentaram os ganhos reais da ferramenta em relação a outros ambientes existentes.

(GAMBI *et al.*, 2017) apresentam Cloud Unit Testing(CUT), uma ferramenta para acelerar a execução de testes unitários utilizando computação em nuvem. CUT aloca recursos computacionais como máquinas virtuais ou contêineres e programa a execução dos testes sobre eles, seguindo as estratégias MaxParallel, OnlyCloud ou Round Robin através de *Programação Orientada a Aspectos* (AOP). Os autores utilizaram um barramento de eventos sobre uma implementação do Apache ActiveMQ[1], um broker de mensageria para a comunicação entre todos os componentes da ferramenta. Um dos grandes diferenciais do projeto é permitir a execução de testes unitários em ambiente de nuvem sem a necessidades de mudança no código original, os desenvolvedores forneceriam as estradas para o ambiente, como credenciais de acesso e estratégias para agrupar, implementar e agendar a execução dos testes unitários. Os resultados apresentados pelos autores demonstraram que os testes executados com CUT atingiram um novo mínimo de tempo de execução, abrindo espaço para que novas técnicas melhorarem os testes de *software* em ambiente de nuvem.

(Lou *et al.*, 2014) propõe um ambiente para agendamento dinâmico de tarefas em ambiente de nuvem. A proposta dos autores segue duas fases sendo a primeira a análise do modelo de tarefas disponível para testes, para eliminar os relacionamentos de dependência e de conflitos, buscando um modelo de tarefas de testes com o máximo de paralelismo. Na

[1] http://activemq.apache.org/

segunda fase um modelo de agendamento dinâmico de tarefas é construído utilizando o modelo de tarefa de teste criado anteriormente, para que seja possível criar uma fila de tarefas alocadas aos recursos disponíveis. O modelo de agendamento dinâmico segue três estágios, sendo eles o agendamento local das tarefas, estágio que utiliza algoritmos genéticos para sua concepção, regulamento local e a espera pelo agendamento, que segue um modelo *first come, first served* (FCFS) para lidar com tarefas em fila. Os autores apresentaram ganhos significativos com a proposta de excluir as dependências e com a utilização de um modelo de agendamento dinâmico utilizando algoritmo genético.

Entre os trabalhos relacionados citados o de (ROJAS *et al.*, 2016), (Ma *et al.*, 2016), (Guo *et al.*, 2018) discutem propostas para a melhoria na execução de testes mobile, utilizando computação em nuvem e propondo sua própria forma de arquitetura de solução, no entanto, em nenhum dos três trabalhos foi apresentado os ganhos obtidos comparados a outros ambientes de nuvem disponíveis no mercado. O trabalho de (GAMBI *et al.*, 2017) apresenta uma proposta para a execução de testes unitários em nuvem, facilitando a execução sem necessidade de reescrita. Visto que o estudo proposto nesse trabalho está relacionado a testes mobile, o trabalho de GAMBI teria limitações, uma vez que a execução de testes unitários não faz uso de dispositivos mobile no processo de execução. O trabalho de (Lou *et al.*, 2014) demostrou a eficiência de algoritmos genéticos para a otimização de tempo de trabalhos através de ordenação de tarefas, abrindo espaço para o estudo de outros algoritmos, como de agendamento de tarefas, na busca por melhor tempo com otimização.

São apresentados na tabela a seguir os aspectos que diferenciam os trabalhos relacionados a este. Os aspectos marcados com X indicam sua existência nos trabalhos, aqueles que apresentamo traço - indica a ausência do aspecto no trabalho. Este trabalho é apresentado na coluna com o título TCC - Oliveira.

Tabela 1 – Trabalhos Relacionados

Lista de Atividade	Rojas et al. (2016)	Guo et al. (2018)	Ma et al. (2016)	GAMBI et al. (2017)	Lou et al. (2014)	TCC - Oliveira
Construção de um ambiente para execução de testes utilizando computação em nuvem.	X	X	X	X	X	-
Execução de testes mobile utilizando computação em nuvem.	X	X	X	-	-	X
Comparação entre os resultados obtidos em ambientes desenvolvidos/existentes e a execução em ambiente local	-	-	-	-	-	X
Execução de teste mobile utilizando dispositivos reais.	-	-	-	-	-	X
Aplicação de algoritmos de otimização.	-	-	-	-	X	X

2.2 Automação de testes

Segundo (MOLINARI, 2008), o teste é a principal forma de se atingir os principais aspectos de qualidade de *software* que existem desde o levantamento de requisitos, definição de arquitetura, codificação e estando presente, principalmente, nos momentos de integração de componentes do sistema e validação das funcionalidades existentes. Logo, é essencial que seja possível executar todos os testes de maneira ágil, de forma simplificada e a qualquer momento, isso só é possível com a ajuda de testes automatizados (BARTIÉ, 2002).

O processo de automação de testes se fundamenta no uso de ferramentas computacionais para execução, contrário aos testes manuais onde os recursos humanos são um fator essencial. Os teste automatizados utilizam uma estratégia de traçar um conjunto de *scripts*, que quando executados, realizam o trabalho pelo testador. A automação permite que em momentos de modificações sobre uma funcionalidade, o restante das funcionalidades que não sofreram mudanças possam ser testadas mais rapidamente, evitando um cenário de extrema repetição e um grande esforço na execução de testes manuais para a equipe.

A importância do uso de teste automático de *software* é definida por (DUARTE *et al.*, 2010) como ferramenta essencial ao desenvolvimento de aplicações de grande porte. Seu uso permite que exista abstração entre equipes dispersas em um mesmo ambiente, possibilitando analisar o comportamento do *software* de maneira rápida e automática, demandando o mínimo de esforço para autoavaliar-se, caso a implementação de uma determinada funcionalidade seja executada como previsto.

Para (BARTIÉ, 2002), a automação de testes permite a execução contínua em um ambiente controlado de testes, tornando possível que se passe mais tempo testando o sistema do que realizando correções. Um maior nível de testabilidade reflete na qualidade do produto final entregue, por meio da identificação de forma prematura de erros presentes na aplicação, evitando que os erros migrem para próximas fases do processo de testes e reduzindo custos e impactos na aplicação.

Logo, a computação em nuvem se mostra como um grande campo a ser explorado para a automação de teste. O uso de um maior poder computacional e recursos sob demanda pode ser usado como meio para acelerar todo o processo de execução, de modo a melhorar o ambiente de execução e otimização de tempo.

2.3 Computação em nuvem

Segundo (TAURION, 2009), o termo computação em nuvem surgiu em 2006, em uma palestra de gerenciamento de data centers ministrada por Eric Schmidt da Google. O termo se referia ao local que se encontravam serviços, dados, arquitetura e armazenamento, fornecendo acesso independente do equipamento ou sistema operacional.

Para (JOSEP *et al.*, 2010), a computação em nuvem surgiu como um novo paradigma, para a oferta de hospedagem e entrega de serviços pela *Internet*. A principal ideia da computação em nuvem é disponibilizar um modelo ubíquo, oportuno e disponível sob demanda, fornecendo acesso a recursos computacionais como redes, servidores, pontos de armazenamento, aplicações e serviços em um agrupamento compartilhável e configurável. Os recursos providos podem ser tanto na forma de serviço quanto de infraestrutura.

A computação em nuvem é vista como uma grande mudança na forma que empresas buscam recursos de *Tecnologia da Informação* (TI), entre os principais benefícios da adoção desse modelo de serviço estão:

- Custo: eliminação de gastos com hardware e *software*, configuração e execução de data centers locais.
- Velocidade: grande maioria dos serviços fornecidos por auto serviço e por demanda, onde recursos computacionais podem ser providos em minutos, bastando alguns cliques.
- Confiabilidade: facilitando e reduzindo os gastos com backup de dados, recuperação de desastres e alto nível de redundância.
- Segurança: conjunto de políticas, tecnologias e controles fortalecem a postura do seu ambiente em relação à segurança.

Além das características anteriormente citada, foram definidos quatro modelos de desenvolvimento dentro do contexto da computação na nuvem que são descritos abaixo:

- Nuvem Privada (Private cloud): a infraestrutura é provisionada para uso exclusivo por uma única organização. Pode ser gerenciada e operada pela organização, por um terceiro ou uma combinação dos dois.
- Nuvem Comunitária (Community cloud): a infraestrutura é provisionada para uso exclusivo de uma comunidade específica de consumidores de organizações com interesses comuns.
- Nuvem Pública (Public cloud): a infraestrutura é provisionada para uso aberto ao

público em geral , pode ser operada por uma organização de negócios, acadêmica ou governamental ou uma combinação entre elas.

- Nuvem Híbrida(Hybrid cloud):a infraestrutura é a composição de dois ou mais modelos de desenvolvimento (privada, comunitária ou pública) que permanecem como entidades únicas, mas são ligados por tecnologias padronizadas ou proprietárias que permite a portabilidade de dados.

2.3.1 Moldelos de computação em nuvem

A forma de distribuição e oferta dos serviços de computação em nuvem apresentam diferentes características, existindo basicamente três modelos de serviços ofertados:

- *Software como Serviço* (SaaS): baseado no conceito de locação de um *software* de um provedor, sem a necessidade de adquiri-lo como convencionalmente. Conhecido principalmente como *software* sob demanda.
- *Plataforma como Serviço* (PaaS): modelo composto por hardware virtual que é oferecido como um serviço que pode ser contratado. Oferece tipos específicos de serviços como sistemas operacionais, banco de dados, serviço de mensageira, armazenamento de dados. Este modelo de serviço é altamente disponível para ambientes de desenvolvimento e implantação de aplicações.
- *Infraestrutura como serviço* (IaaS): composto por plataformas para desenvolvimento, teste, implantação e execução proprietárias, estando muito ligada a recursos como servidores, redes, firewall, entre outros que são essenciais na construção de um ambiente sob demanda.

2.4 Virtualização

O conceito de virtualização é extremamente discutido e usado em assuntos voltados à computação. Para (KIM *et al.*, 2015), virtualização pode ser definida como sendo a criação de recursos computacionais utilizando para isso recursos físicos, como ocorre em servidores, onde uma máquina física com alto poder de recursos é utilizada como meio para suportar máquinas virtuais com operações paralelas.

A virtualização é possível graças ao gerenciamento de recursos virtuais realizado pelo *Hypervisor*. Por meio dele, múltiplos sistemas operacionais compartilham de hardware

em um único host, com especificações e usos que variam de acordo com a finalidade de cada máquina virtualizada.

Frequentemente, se costuma confundir computação em nuvem com virtualização, devido a ambas estarem relacionadas a criação de ambientes através de recursos abstratos. Enquanto a virtualização permite criar vários ambientes simulados, os recursos dedicados por um único sistema ou hardware físico, a computação em nuvem permite abstrair, agrupar e compartilhar recursos escaláveis mediante uma rede (MESEVAGE, 2019).

2.5 Testes em nuvem

A computação em nuvem vem sendo utilizada para diversas linhas de pesquisa, na área de teste de *software* um novo modelo de serviço em nuvem vem ganhando espaço, *Teste como Serviço* (TaaS). TaaS permite testar o *software* em um ambiente de nuvem como um Web Service, estando facilmente acessível, e utilizando a vasta quantidade de recursos que estão disponíveis em uma infraestrutura de nuvem (CANDEA *et al.*, 2010).

O teste em nuvem é um modelo que busca aproveitar ao máximo as tecnologias de computação em nuvem, tornando possível diminuir o tempo gasto na execução de testes e permitindo simular tráfego que a aplicação vai enfrentar quando for liberada para produção. O modelo possibilita ainda reduzir o uso de dispositivos móveis aproveitando as tecnologias atuais da nuvem, usando o modelo de serviço de aluguel de dispositivos(GAO *et al.*, 2014).

Diferente de um ambiente configurado em uma arquitetura local, testes em um ambiente de nuvem apresentam como principal diferença o fato de existir um ambiente virtualizado e compartilhado. Esse modelo de estratégia busca atender a diversos critérios de qualidade de *software* como performance, tempo de resposta, segurança entre outros que são importantes em qualquer plataforma.

Inúmeros serviços de teste em nuvem estão disponíveis no mercado, no entanto, apesar de parecer que todos apresentam uma forma de serviço semelhante, eles se diferenciam na forma de teste que permitem ser executados em ambiente de computação em nuvem (KILINÇ *et al.*, 2018).

Como principais fatores para a adoção de teste de nuvem (SATASIYA, 2018) aponta 6 pontos que garantem produtos de *software* de alta qualidade com tempo de entrega mínimo, sendo eles:

- colaboração em tempo real;

- tempo e custo efetivo;
- fácil acesso a recursos;
- suporte a vários ambientes e recursos;
- compatibilidade com o ambiente de teste;
- permite que o aplicativo alcance o mercado mais rapidamente.

2.5.1 Plataformas de teste móvel em nuvem

2.5.1.1 Genymotion Cloud

O Genymotion Cloud[2] é um serviço de emulação de dispositivos android na nuvem. Permite utilizar dispositivos para diversos fins, entre eles, execução de testes automatizados. O serviço permite o redimensionamento dos testes e o acesso ao dispositivo virtual android ao vivo de qualquer lugar(GENYMOTION, 2020).

Genymotion surgiu como um *software* de emulação de dispositivos android *desktop*, Genymotion cloud segue o mesmo modelo de dispositivo executado diretamente em uma máquina virtual, no entanto, utilizando uma infraestrutura de nuvem, sendo compatível com as mais conhecidas estruturas de automação de testes mobile como: Appium[3], Expresso[4] e Robotium[5]. O serviço permite que se escale sua infraestrutura com quantos dispositivos virtuais você precisar, de acordo com as necessidades de uso.

O acesso aos recursos disponíveis pelo Genymotion Cloud podem ser realizados de duas maneiras, a primeira através de um serviço baseado em nuvem executado de maneira SaaS, onde você paga pelo tempo de consumo, a segunda utilizando um modelo de serviço disponível com o uso de imagens Docker[6] de maneira PaaS, que são disponibilizados pelos principais provedores de nuvem, entre os listados no site do próprio Genymotion que ofertam o serviço está: *Google Cloud Platform* (GPC)[7], *Amazon Web Services* (AWS)[8] e *Alibaba Cloud*[9] (GENYMOTION, 2020).

[2] https://www.genymotion.com/cloud/
[3] http://appium.io/
[4] https://developer.android.com/training/testing/espresso/intents?hl=pt-br
[5] https://github.com/RobotiumTech/robotium
[6] https://www.docker.com/
[7] https://cloud.google.com/
[8] https://aws.amazon.com/pt/
[9] https://www.alibabacloud.com/

2.5.1.2 AWS Device Farm

Segundo a descrição do próprio produto, o AWS Device Farm é um serviço de teste que permite melhorar a qualidade dos aplicativos por meio da execução de testes em uma ampla variedade de navegadores desktop e dispositivos móveis reais ou virtuais, sem a necessidade de provisionamento e gerenciamento. A execução em diversos dispositivos busca acelerar a execução do conjunto de testes. Ao final da execução o serviço provisiona para o usuário logs e vídeos da execução para ajudar a identificar os problemas com a aplicação e acelerar o processo de correção de problemas (DEVICEFARM, 2020).

Um dos diferenciais do ambiente disponibilizado pelo Device Farm está no formato de execução, além da forma de execução remota, com os teste em ambiente local e dispositivos conectados remotamente, existe também a possibilidade de *upload* do projeto construído e a execução em um ambiente de nuvem, juntamente com seus dispositivos para teste, o que acelera o processo de execução.

O AWS Device Farm oferece suporte a testes para sistemas Android, IOS e Fire OS em aplicativos nativos e híbridos, sendo compatível com a ferramenta de automação Appium. O acesso aos recursos disponibilizados pelo Device farm[10] é feito mediante uma assinatura, onde o usuário paga pelo tempo de uso dos dispositivos.

2.5.1.3 BrowserStack

O BrowserStack é um ambiente de teste móvel em nuvem que permite que desenvolvedores testem seus aplicativos em uma série de navegadores desde os mais recentes até versões mais antigas. O serviço disponibiliza para os usuários uma nuvem de dispositivos reais Android e IOS que podem ser utilizados para a realização de testes de aplicações mobile e obter resultados mais precisos em comparado a teste em dispositivos no ambiente físico (BROWSERSTACK, 2020).

Um dos grandes trunfos do BrowserStack está na forma de funcionamento do ambiente de execução de testes, possibilitando flexibilidades aos desenvolvedores, ao permitir a execução de um conjunto de testes utilizando a própria infraestrutura de armazenamento do serviço, ou por meio de uma execução remota, utilizando uma conexão altamente segura entre máquina local e ambiente de nuvem.

[10] https://aws.amazon.com/pt/device-farm/

A ferramenta tem se destacado no âmbito corporativo pela alta compatibilidade com ferramentas de integração contínua, como Jenkins[11], Travis[12], Bamboo[13]. É possível ainda trabalhar diretamente no ambiente de nuvem com ferramentas de automação, como Appium e Selenium Grid, neste ponto vale destacar a facilidade de integração entre ambiente de nuvem e e um ambiente local de execução.

2.5.2 Comparativo entre ambientes

Perante as breves descrições dos ambientes citados para essa pesquisa, os autores realizaram uma análise, com intuito de definir um ambiente para a continuidade dessa pesquisa.Os principais pontos analisados encontram-se no Apêndice A. Após um análise dos principais pontos os autores escolheram o ambiente Browserstack para realização dessa pesquisa. Os pontos mais relevantes para a escolha do ambiente foram: A disponibilidade de utilização do ambiente de forma gratuita por um ano utilizando uma licença de estudantes; A facilidade de integração do ambiente a suíte de testes, visto que sua documentação é extremamente prática com exemplos e passos de como realizar a integração ao ambiente.

2.5.2.1 Ambiente escolhido

O BrowserStack é uma plataforma de teste móvel e web em nuvem que permite aos desenvolvedores testar sites e aplicativos móveis em diferentes navegadores, sistemas operacionais e dispositivos móveis reais . Por meio do ambiente, os usuários podem selecionar entre uma gama de mais de 1.200 dispositivos móveis de diferentes sistemas, sendo os principais o Android e IOS, principais navegadores do mercado e sistemas operacionais reais como Windows e Mac Os.

O usuário conta com uma infraestrutura extremamente segura e estável que permite realizar milhares de testes manuais e de automação simultaneamente por meio de execução distribuídas.

O BrowserStack funciona conectando o usuário a uma máquina real em execução em um dos milhares data centers espalhados geograficamente, e que permite uma melhor distribuição e acesso aos recursos.

[11] https://www.jenkins.io/
[12] https://travis-ci.org/
[13] https://www.atlassian.com/br/software/bamboo

Entre as principais funcionalidades disponibilizadas pelo ambiente de nuvem BrowserStack se destacam:

- Live e Automate: Para a realização de testes em sites web.
- App live e App automate: Para a realização de testes em aplicativos móveis.
- Percy: para a realização de testes visuais em aplicações web.

Olhando especificamente para testes mobile por meio do app automate, o ambiente Browserstack permite se realizar testes utilizando diferentes estruturas e diferentes linguagens. A pluralidade de *frameworks* suportados o torna compatível para diversos projetos independente de linguagem.

Existe uma série de recursos encontrados nesse ambiente que permitem uma maior adaptação dos testes a cenários específicos, como por exemplo a possibilidade de customizar a velocidade de acesso ou conexão de dados e simulação de localização GPS do dispositivo.

2.6 Ferramentas para automação de teste de software

2.6.1 Appium

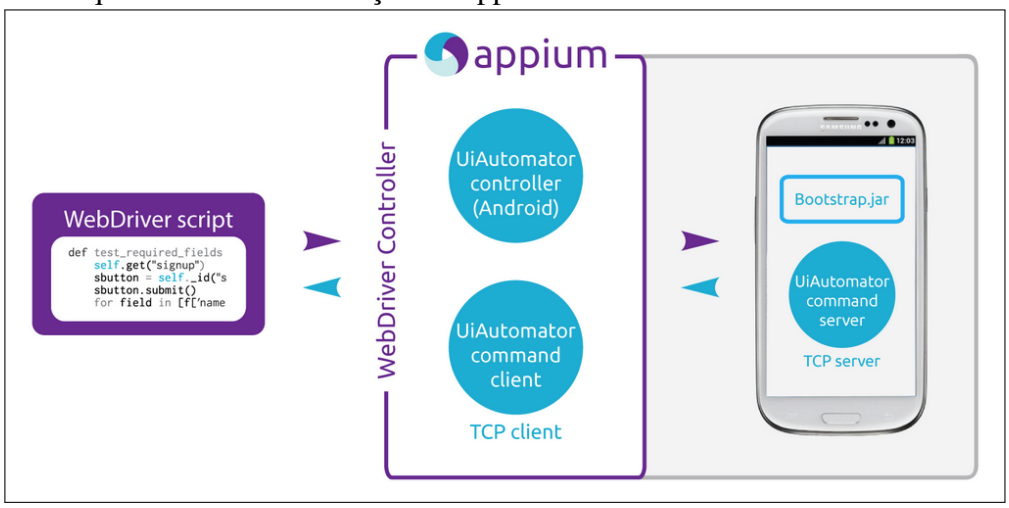
O Appium é uma ferramenta 'open-source', multi-plataforma que permite a automatização de testes de aplicações nativas e híbridas para as plataformas Android e IOS. Com o Appium é possível escrever testes em várias plataformas utilizando a mesma *Interface de programação de aplicações* (API), possibilitando a reutilização de código para teste mobile (APPIUM, 2020).

A ferramenta Appium utiliza a API WebDriver do Selenium para enviar comandos de teste, por esse motivo é possível que os testes possam ser escritos em diversas linguagens de programação. Programadores já habituados à escrita de teste utilizando selenium[14] tendem a se sentir habituados para a escrita de teste mobile utilizando appium.

O Appium funciona por meio de um servidor escrito em Node[15]. O servidor funciona em uma arquitetura cliente-servidor, onde o cliente envia solicitações relacionadas a automação para o servidor Appium, estas solicitações enviadas pelo cliente são conhecidas como solicitações de sessão, e transportam informações em um formato *JavaScript Object Notation* (JSON) um formato de comunicação também conhecido como 'JSON Wire Protocol'.

[14] https://www.selenium.dev/
[15] https://nodejs.org/en/

Figura 1 – Arquitetura de comunicação do appium.



Fonte: (KOçU, 2017)).

A Figura 1 acima ilustra o modelo de comunicação do Appium, onde um modulo central é composto por pequenos módulos executados na aplicação, utilizando o protocolo WebDriver. Por meio desses dois módulos, é possível que o Appium realize ações na aplicação, como, clicar em botões realizar a captura de telas e realizar chamadas a API.

2.6.2 Selenium Grid

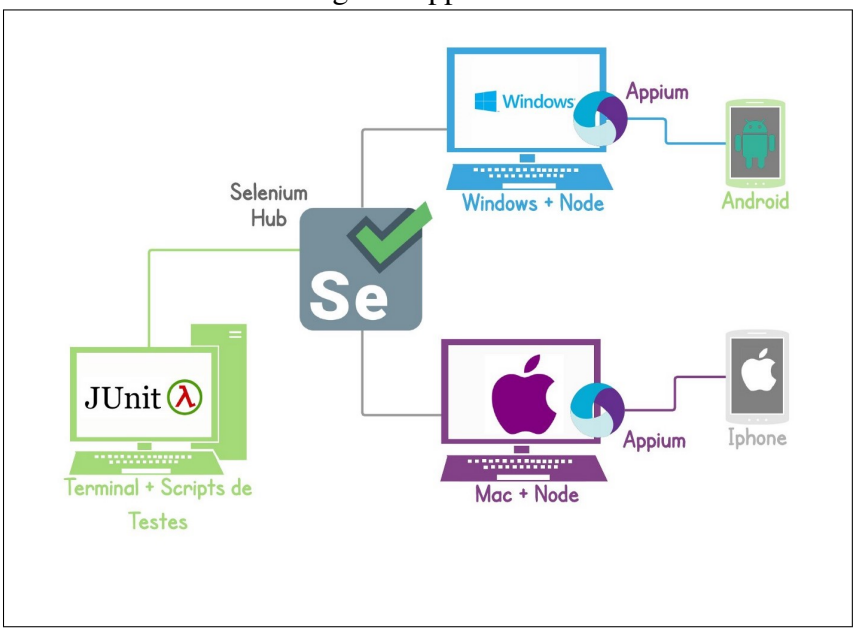
O Selenium Grid é uma ferramentar importante da suite Selenium, por meio da sua utilização é possível que vários testes possam ser executados simultaneamente em vários navegadores da web e sistemas operacionais em máquinas distintas. Dois elementos importantes compõem o Selenium grid, um Hub e os nós (KUHN, 2019).

Em uma representação arquitetural no Hub temos um computador central onde são carregados os testes, ele desempenha o papel de servidor, atuando no controle da rede de máquinas para testes. Quando um conjunto de testes é entregue ao Hub é realizado uma verificação das máquinas ou nós disponíveis, tais nós podem ser máquinas físicas ou virtuais. Com nós disponíveis e conectados ao Hub é enviado uma ordem de execução do teste para nó, dessa forma é possível criar uma rede de máquinas de teste conectadas.

Outro papel que o Selenium Grid pode desempenhar é o de balanceador de carga, dividindo todo trabalho que precisa ser realizado no processo de teste, entre os nós conectados ao Hub, evitando que um determinado nó fique sobrecarregado com as atividades que precisa executar.

A Figura 2 ilustra um fluxo de trabalho utilizando selenium grid. Nesse fluxo é

Figura 2 – Fluxo de trabalho selenium grid e appium.



Fonte: (MENEZES, 2018)).

possível notar um Hub como unidade central de conexão para outros equipamentos, facilitando o processo de utilização de múltiplas máquinas, e realizando o controle da distribuição dos trabalhos para os nós disponíveis.

2.7 Job-Shop Scheduling

Segundo (APPLEGATE; COOK, 1991) Job-Shop Scheduling é um problema de otimização que busca agendar um conjunto de tarefas em um conjunto de máquinas sujeitas a restrições, como a quantidade de tarefas executadas por cada máquina e ordem de processamento através das máquinas, com objetivo de minimizar o tempo de conclusão das tarefas. Cada trabalho consiste em uma sequência de tarefas que devem ser executadas em uma determinada ordem, e cada tarefa deve ser processada em uma máquina específica.

Algumas restrições estão presentes no problema de Job-Shop, caracterizando o problema: a primeira delas diz que nenhuma tarefa atribuída a um trabalho pode ser iniciada até que a tarefa anterior para esse trabalho seja concluída; a segunda restrição diz que uma máquina só pode trabalhar em uma tarefa por vez; a terceira restrição aponta que uma tarefa, uma vez iniciada deve ser executada até sua conclusão.

Diversos algoritmos foram desenvolvidos buscando oferecer uma solução eficiente para problemas de job-shop, o Google fornece a exemplo uma biblioteca denominada Or-tools, um conjunto de software de código aberto para problemas de otimização ajustado para enfrentar

os problemas mais difíceis do mundo em roteamento de veículos, fluxos, programação linear e inteira e programação de restrições entre os problemas o Job Shop Problem (OR-TOOLS, 2020).

Outros problemas de otimização estão presentes na literatura além do Job-Shop, onde tradicionalmente as diferenças entre os métodos se encontram na forma de classificação e alocação de tarefas. No método Flow-Shop de agendamento, cada tarefa tem a mesma sequência de processamento na sequencia de maquinas. Já o método Open-shop não há uma sequência específica ou preestabelecida para o processamento das tarefas(NAGANO *et al.*, 2004).

No contexto de testes de *software* o problema de Job-Shop se apresenta como uma forma de auxiliar na otimização do tempo de execução dos testes. Comumente a execução de teste de *software* segue uma sequência de trabalho Flow-Shop sendo fixa e repetitiva, com o Job-Shop se busca encontrar uma sequência ideal de execução das tarefas, esforços podem ser concentrados ainda na redução da quantidade de tempo ocioso entre os centros de trabalhos, ou seja, repasse de testes para execução aos dispositivos disponíveis.

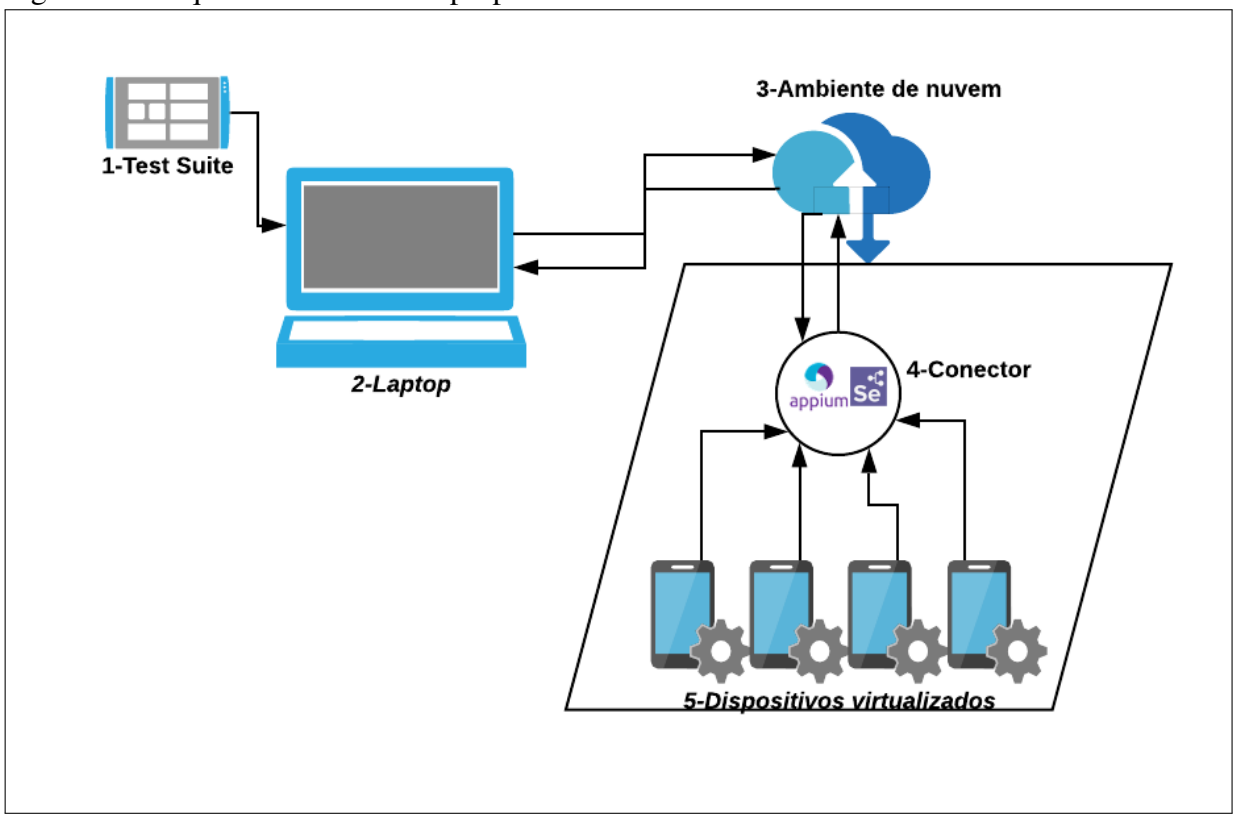
3 PROPOSTA

Este capítulo apresenta a arquitetura da solução proposta. Serão descritas as partes que compõem a solução, tecnologias e *softwares* escolhidos pelos autores, buscando resolver os problemas de pesquisa citado anteriormente.

3.1 Proposta

A proposta deste trabalho consiste na escrita de um conjunto de testes com base em uma aplicação Android fornecida para estudo. Em seguida, a execução dos testes construídos anteriormente em dispositivos virtualizados ou reais utilizando ambientes em nuvem que forneçam suporte a esse tipo de serviço. Ao final a apresentação dos resultados comparativos dos ganhos obtidos com a solução desenvolvida.

Figura 3 – Arquitetura do sistema proposto



Fonte: elaborado pelo autor (2020).

A Figura 3 apresenta o modelo de arquitetura proposta. Há cinco partes principais, cada uma delas com um propósito para a solução, são elas:

1. Test suite: um conjunto de testes mobile que serão escritos com base na aplicação Android

disponível para estudo.

2. Laptop: máquina local que será utilizada para a execução dos testes gerados na etapa anterior e para a chamada aos dispositivos no ambiente de nuvem.
3. Ambiente de nuvem: ambiente que permite a execução dos testes mobile em dispositivos disponíveis na plataforma.
4. Conector: ferramentas de *software*, em sua maioria *frameworks* que permitem a interação através de testes automatizados em dispositivos móveis.
5. Dispositivos: conjunto de dispositivos móveis virtualizados ou reais, disponíveis em um ambiente de nuvem, que podem ser replicados conforme as necessidades do usuário.

3.2 Como é implementado

3.2.1 Camada de Test Suite

A camada de *test suite* tem como objetivo principal servir como fonte de entrada para o processo de automação. Por meio dessa camada a execução dos testes poderá ser feita de forma local, executando em uma máquina do usuário, ou utilizando o ambiente de nuvem, realizando o *upload* do conjunto de testes para o ambiente.

Para a idealização dessa camada, será necessário um conjunto de ferramentas, buscando tornar o processo de automação e escrita dos testes uma prática viável e versátil. No Capítulo 2, foram descritos algumas tecnologias úteis no processo de automação mobile.

A linguagem de programação utilizada para o desenvolvimento dos testes automatizados foi o JAVA[1], em sua versão 8. A escolha dessa linguagem para a idealização do projeto se dá principalmente por ser uma linguagem orientada a objetos e por ter compatibilidade com uma série de *frameworks* e ferramentas utilizadas em testes mobile, que podem ser utilizadas nesse trabalho.

Para a criação desta camada, um *framework* de testes é necessário, principalmente para verificar os resultados obtidos em testes funcionais. Um dos mais utilizados quando se trabalha com java é o Junit[2], no entanto, os autores preferiram utilizar o *framework* TestNG[3]. TestNG é baseado no Junit, apresenta uma série de melhorias relacionadas à rotina de criação de testes, execução de testes automatizados e um número maior de anotações que podem ser

[1] https://www.java.com/pt$_BR$/
[2] https://junit.org/junit5/
[3] https://testng.org/doc/

utilizadas para a definição de comandos ou rotinas na estrutura de testes.

Uma das notações mais utilizadas em *scripts* de teste utilizando TestNG é a anotação "@Test", ela informa que um determinado método ou classe é um componente de teste. As anotações utilizando está ferramenta podem ser complementadas com uma série de parâmetros definidos em arquivo *Extensible Markup Language* (XML) de configurações de testes. A Figura 4 ilustra um exemplo de método de teste utilizando anotações do TestNG:

Figura 4 – Método de teste utilizando a anotação @Test

```
1.    @Test
2.    public void deveInteragirSwitchcheckBox(){
3.        //verificar o status dos elementos
4.        Assert.assertFalse(page.isCheckMarcado());
5.        Assert.assertTrue(page.isSwitchMarcado());
6.        //clicar nos elementos
7.        page.clicarCheck();
8.        page.clicarSwitch();
9.        //verificar estados
10.       Assert.assertTrue(page.isCheckMarcado());
11.       Assert.assertFalse(page.isSwitchMarcado());
12.   }
```

Fonte: elaborado pelo autor (2020).

A utilização do gerenciador de dependências nessa camada tem como objetivo permitir a portabilidade do projeto para diversos ambientes, de modo que todas as dependências do projeto sejam empacotadas e baixadas corretamente no ambiente onde o projeto será carregado. O Maven[4] será utilizado como o gerenciador de dependências nessa camada.

3.2.2 Camada de execução

A camada de execução desempenha o papel de construir a camada de *test suite*, sendo uma intermediadora entre, ambiente de nuvem e o projeto de teste. No diagrama da Figura 3 essa camada é representado pelo item 2. Nesta etapa uma unidade com poder computacional realizará a execução dos testes, repassando as requests com os comandos que devem ser executados no ambiente de nuvem. O repasse dos comandos e feito através de uma conexão remota.

[4] http://maven.apache.org/

Logo, a camada de execução desempenha uma papel de orquestrador dos testes, podendo executar todos os testes ou parte deles, controlando o fluxo de execução e realização de assertivas, bem como a geração de dados para serem usados como evidencias de execução.

Inicialmente a camada de teste suíte executa sobre esse ponto utilizando a vitual machine do Java, ou seja, um simples projeto java empacotado em um arquivo .Jar, no entanto, os autores pretendem para o futuro tornar esse projeto em uma imagem docker, o que facilitaria o processo de execução em ambientes de computação em nuvem e escalonamento desse projeto por meio de containers.

A necessidade de execução em um ambiente local se dá pelo fato da realização de uma comparação entre os dados obtidos quando realizada a execução utilizando dispositivos reais em um ambiente de nuvem, e os dados obtidos por meio da execução híbrida entre dispositivos reais e virtualizados localmente.

3.2.3 Camada de nuvem

Essa camada do projeto é responsável pela utilização dos dispositivos móveis, empregando o poder de computação em nuvem para alocação e provisionamento dos recursos, além da entrega dos testes que devem ser executados nos dispositivos. No diagrama da Figura 3 esta camada diz respeito ao ponto 3, que representa o ambiente de nuvem escolhido. No entanto, os seguintes pontos também estão relacionados a essa camada: O ponto 4 representa as tecnologias utilizadas para conexão entre dispositivos móveis e testes, o ponto 5 os dispositivos móveis utilizados para a execução dos testes.

O conector no diagrama da Figura 3 está relacionado às tecnologias necessárias para a conexão entre teste e dispositivo móvel. Este é um módulo que pode estar localizado interno ao ambiente de nuvem, ou externo, na camada de execução. A escolha dos autores de utilizar os conectores no próprio ambiente de nuvem tem como objetivo aumentar a eficiência e garantir uma maior estabilidade na execução dos testes, visto que a parte responsável por conexão e repasse dos comandos de testes estará mais próximo dos dispositivos alocados.

Apesar de os dispositivos se encontrarem no ambiente de nuvem, a definição de qual dispositivo deve ser criado para a execução dos testes se encontra na camada de *Test Suite*, ponto 1 da Figura 3. Logo, um conjunto de testes pode ser realizado utilizando N instâncias em nuvem de um mesmo dispositivo móvel, ou então, N dispositivos móveis para os N testes que devem ser realizados. A figura 5 acima ilustra por exemplo a definição de um aparelho Samsung galaxy S9

Figura 5 – Definição de um dispositivo no arquivo testng.xml

```
1. <test name="Galaxy S9 - 8.0">
2.         <parameter name="deviceName" value="Galaxy S9" />
3.         <parameter name="browserName" value="android" />
4.         <parameter name="realMobile" value="true" />
5.         <parameter name="os_version" value="8.0" />
6.         <classes>
7.             <class name="test.BaseTest" />
8.         </classes>
9. </test>
```

Fonte: elaborado pelo autor (2020).

na camada de *Test Suite* para a realização dos testes.

3.3 Otimização dos testes

Neste ponto os autores propõem um objetivo secundário, buscar uma forma de otimização dos testes utilizando algoritmos de agendamento de trabalhos. No capítulo 2 foi apresentado um tópico sobre o problema do Job-Shop Scheduling, problema de agendamento de trabalho que autores pretendem utilizar durante o estudo, buscando apresentar uma sequência ótima na ordem de execução dos testes.

A sequência ótima de execução se resume a uma ordem dos testes automatizados que devem ser executados, buscando reduzir o tempo de inatividade dos dispositivos entre o intervalo de alternância dos testes. O framework TestNG utilizado na camada de *Test Suite* é uma peça fundamental, pois permite definir a ordem que os testes serão executados por meio do uso de anotações.

4 MODELAGEM E PROCESSO DE TESTE

Este capítulo apresenta de forma descritiva a estrutura de desenvolvimento do projeto, as etapas necessárias de configuração e execução do projeto, avaliação e validação dos ganhos da execução dos testes em uma máquina local e comparativo utilizando um ambiente de nuvem. O processo de validação tem início com a execução dos testes mobile em uma máquina local, com o intuito de comparar os resultados obtidos a partir da execução em um ambiente de nuvem. Essa comparação tem como principal objetivo verificar a eficiência e o *speedup* entre teste mobile em ambiente local e os testes mobile utilizando ambiente de computação em nuvem.

4.1 Metodologia

Para realização dos experimentos, desenvolveu-se um conjunto de scripts de testes mobile para a execução de alguns dos fluxos da aplicação Greenmile Driver7. A construção e escrita desses testes teve como objetivo obter o tempo médio de processamento e execução quando executado em uma máquina local, assim, facilitando a comparação com os resultados obtidos através da execução no ambiente de nuvem selecionado para esse trabalho. O conjunto de testes desenvolvido possui 22 testes funcionais que cobrem cenários pequenos, médios e longos da aplicação Greenmile Driver7. Adiante será descrito as principais diferenças e características dos testes desenvolvidos para a aplicação.

Os experimentos deste trabalho se dividem em 2 cenários: O primeiro, onde o conjunto de testes é executado 33 vezes em uma máquina local com dispositivos físicos reais simulando um ambiente de teste comum à muitas empresas. O segundo cenário, onde o conjunto de testes será executado 33 vezes utilizando agora uma máquina local conectada a dispositivos reais em um ambiente de computação em nuvem SaaS. A execução em ambos os cenários será feita utilizando um dispositivo móvel samsung galaxy s10e.

A análise dos resultados desse experimento foi realizada em sub-tarefas. A princípio, será feita a atividade de erradicar anomalias presentes nos dados de teste, com o objetivo de melhorar os resultados finais, evitando falsos negativos. Em seguida, será calculado a média do tempo de execução e o desvio padrão com base nos dados tratados da etapa anterior. Logo será calculado o intervalo de confiança utilizando como parâmetro um intervalo com $\alpha = 95\%$.

Os testes seguem um processo de preparação prévia para sua execução, o processo definido ajuda a garantir que os equipamentos utilizados estejam completamente configurados e

preparados para a execução. Existem algumas diferenças entre o processo de execução remoto e local, essas diferenças serão detalhadas nos tópicos seguintes. Tais tópicos explicam os comandos para a execução do projeto, ou seja, se a chamada será para execução remota ou local. Esse processo de preparação garante que os recursos computacionais necessários para a execução do projeto e respectivamente dos testes estará em perfeito funcionamento.

A configuração da máquina de execução do projeto de testes possui processador Intel core i7 2.20 GHz, 16 Gigabytes de memória RAM, SSD NVMe de 256 Gigabytes e placa gráfica NVIDIA GeForce MX250 com sistema operacional Windows 10 Home de 64Bits. As altas margens de processamento e memória tém como objetivo garantir que o equipamento central de execução do projeto não seja um gargalo no momento de execução. No processo de execução dos testes todos os recursos computacionais da máquina estarão dedicados exclusivamente ao processo de execução.

4.1.1 Aplicação utilizada

O Greenmile Driver7 é uma aplicação híbrida desenvolvida utilizando ReactJS. A aplicação fornece informações atualizadas sobre rotas de diversos clientes, entregas em diferentes estágios, podendo esses estágios ser pendente, realizada ou cancelada. A aplicação disponibiliza informações sobre o status de pedidos para motoristas e despachantes através de uma integração com outros produtos do ecossistema Greenmile, bem como atualizações em tempo real conforme elas ocorrem sobre uma rota.

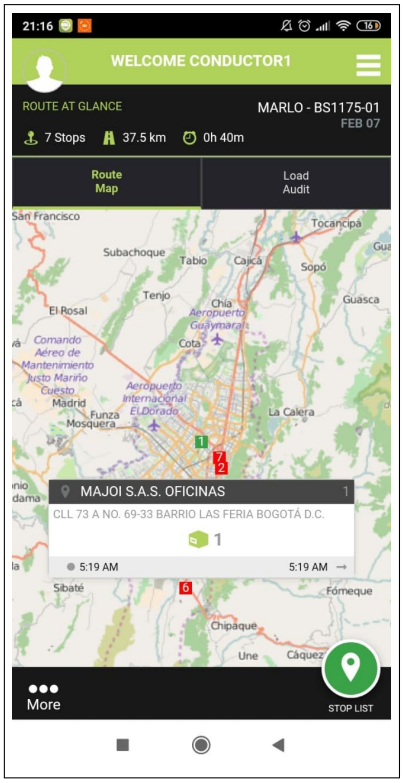
Para a realização de testes de software, a aplicação Greenmile Driver7 permite exercitar a utilização de diversos componentes de um dispositivo móvel. A aplicação utiliza integração com a câmera para a realização de fotos, integração com GPS para captura de coordenadas, hora do dispositivo para o controle de jornada de trabalho, utilização da tela como canvas de assinatura, envio de ações através de notificações bem como a possibilidade de realização de ações com ou sem rede de dados ativa, permitindo testes em modo offline.

A aplicação Greenmile Driver7 realiza a troca de informações com uma aplicação servidora, funcionando em uma arquitetura cliente servidor. Essa comunicação é necessária para que a aplicação tenha acesso à informações registradas por outros softwares da suíte Greenmile. Para a utilização da aplicação por motoristas, é necessário que ele esteja registrado no sistema, este cadastro é realizado por meio do Greenmile Live, aplicação onde uma série de outras informações podem ser cadastradas, como equipamento utilizado pelo motorista, paradas que o

motorista deverá atender entre outras informações que compõem uma rota.

Para a idealização dos testes e realização desse trabalho é necessário que se crie uma massa de dados prévia no ambiente servidor que será utilizado para integração e execução dos testes do Driver7. Como caráter de demostração a Figura 6 ilustra a tela de boas vindas onde o motorista consegue visualizar dados do trajeto que será realizado.

Figura 6 – Tela de welcome da aplicação.



Fonte: elaborado pelo autor (2020).

No presente trabalho, os proprietários legais do aplicativo Greenmile Driver7 cederam permissão para realização dos testes e vinculação da marca ao trabalho. O e-mail de autorização se encontra no apêndice B.

4.1.2 Cenário de teste

Um cenário de teste é um tipo menos detalhado de documentação, sendo a descrição de um objetivo que o usuário espera alcançar ao executar uma aplicação. Um cenário de teste precisa de diferentes categorias de testes para garantir que o objetivo definido tenha sido bem testado. Podemos utilizar o cenário da funcionalidade de login e verificar que há diferentes cenários válidos a serem testados para garantir que a funcionalidade na totalidade esteja completamente coberta por testes, a Figura 7 abaixo ilustra diferentes cenários que podem

ser realizados sobre a funcionalidade de login.

Figura 7 – Tabela com combinações de login.

CT	Login	Senha
CT1	Em branco	Em branco
CT2	Inválido	Inválida
CT3	Válido	Inválida
CT4	Válido	Válida

Fonte: elaborado pelo autor (2020).

Para viabilizar a realização desta pesquisa, os autores realizaram um estudo sobre algumas das funcionalidades da aplicação Driver7 e selecionaram um conjunto de funcionalidades para serem automatizadas. Devido ao fator de tempo para a construção do projeto os autores deram ênfase as funcionalidades mais simples da aplicação, no entanto, incluíram algumas funcionalidades mais complexas para análise dos resultados. Os cenários escolhidos foram classificados em três tipos: pequeno, médio e longo.

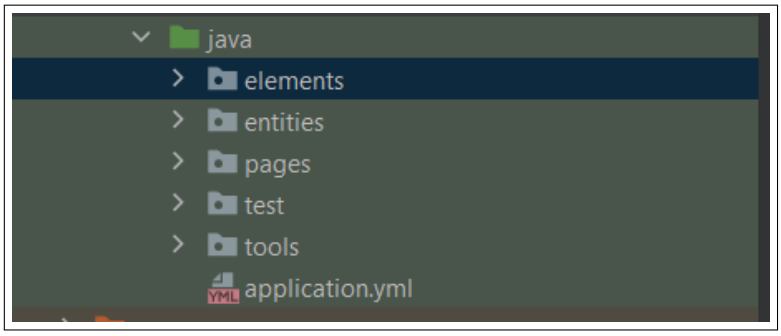
Foram definidos alguns critérios para classificar os cenários de teste. O primeiro deles é a quantidade de passos executados por um cenário, cenários com até 5 passos foram classificados como pequenos, cenários de testes com até 10 passos foram classificados como médio, e cenários com mais de 10 passos foram classificados como longos. Essas métricas visam equilibrar a execução, visto que um cenário com mais passos trará um tempo de execução mais longo quando comparado a um cenário com passos mais curtos. Vale salientar que esse critério foi definido especificamente para esse trabalho, não tendo sido definido em pesquisar anteriores como parâmetro de medição.

O segundo critério definido foi utilizado para determinar o tamanho dos cenários de teste, a definição foi feita pela quantidade de telas que o cenário cobre, ou seja, quantas páginas da aplicação são necessárias para que o teste seja executado. Os autores definiram como pequeno um cenário onde duas páginas fossem cobertas, médio para cenários onde até quatro páginas fossem cobertas e longo para testes onde mais de 4 páginas fossem cobertas. Os cenários automatizados são os listados no apêndice C.

4.2 Estrutura do projeto de testes

A estrutura do projeto segue o padrão Page Objects, padrão que funciona como interface de acesso a elementos da camada de visão. Ele é aplicado para abstrair as páginas de uma aplicação com o objetivo de reduzir o acoplamento entre os casos de teste e a aplicação a ser testada. A estrutura do projeto seguindo o padrão page objects está definida como mostrado na Figura 8 a seguir:

Figura 8 – Estrutura pacotes projeto.



Fonte: elaborado pelo autor (2020).

O pacote elements tem por finalidade agrupar todos os elementos por tela, ou seja, unificar todos os elementos listados em um tela para que eles possam ser consultados em um momento oportuno. Todos os elementos definidos em cada um dos arquivos segue o padrão de busca pelo *Xpath* para busca dos elementos na interface. Ainda no pacote Elements existe um arquivo que lista todas a *pages* que então disponíveis na aplicação, sendo utilizadas quando for necessário verificar a tela atual da aplicação. A Figura 9 ilustra o conteúdo do arquivo *ElementsCurentPage* onde temos variáveis representando algumas telas da aplicação, com o valor seguindo o caminho do *Xpath* para busca:

Figura 9 – Arquivo com elementos da tela atual.

```
package elements;

public class ElementsCurrentPage {

    public static String setting = "/hierarchy/android.widget.FrameLayout/android.widg
    public static String login = "/hierarchy/android.widget.FrameLayout/android.widget
    public static String tutorial = "/hierarchy/android.widget.FrameLayout/android.wid
    public static String equipament = "/hierarchy/android.widget.FrameLayout/android.w
    public static String routeAtGlance = "/hierarchy/android.widget.FrameLayout/androi
    public static String changePassword = "/hierarchy/android.widget.FrameLayout/andro
    public static String stopList = "/hierarchy/android.widget.FrameLayout/android.wid
    public static String routeSelection = "/hierarchy/android.widget.FrameLayout/andro
    public static String reasonCode = "/hierarchy/android.widget.FrameLayout/android.w
}
```

Fonte: elaborado pelo autor (2020).

O Pacote Pages está relacionado a cada cenário no aplicativo ou software que será automatizado. Um exemplo claro é uma tela de login, que possui diversas iterações e componentes presentes na tela, que por sua vez são interações específicas da tela de login. A Figura 10 ilustra os métodos particulares de ações de cada componente da tela de login, que estão implementados e interagindo com os elementos definidos pelas classes do pacote de *Elements* referente a página de login:

Figura 10 – Comandos do arquivo LoginPage.

Fonte: elaborado pelo autor (2020).

No pacote Tools se encontram classes indispensáveis para a execução desse projeto, nesse pacotes estão definidos o *core* de configuração, integração e execução dos testes, tanto para ambiente local quanto em nuvem. A classe DriverFactory é a responsável pela geração de uma conexão entre a suíte de teste e os dispositivos móveis. Nessa classe se encontram duas definições, a primeira para criação de uma conexão local e a segunda para criação de uma conexão remota. Essa classe segue um padrão *Singleton*, sempre existindo apenas uma instância da conexão com o dispositivo por teste. A Figura 11 ilustra as definições para conexão local.

Figura 11 – Criação de conexão local.

Fonte: elaborado pelo autor (2020).

Por fim temos o pacote Test, esse pacote agrupa todas as classes de alto nível, que fazem o contato com as pages e determinam quais passos serão executados.

4.3 Cenário de teste local

A seguir será descrito o processo para execução do projeto de teste em um ambiente local.

4.3.1 Preparação do ambiente

Para a execução dos testes de forma local, toda a preparação dos equipamentos e *softwares* necessários é realizado por quem vai conduzir a execução. Por se tratar de um modelo de teste mobile, é necessário que dispositivos móveis estejam presentes durante toda a execução. Os dispositivos disponíveis servem como elemento de interação e execução da aplicação, algo semelhante ao papel desempenhado pelos navegadores em testes web.

Originalmente, os dispositivos móveis do mercado não estão preparados para esse processo de execução de testes e depuração de informações por meio da conexão USB, é necessário que seja habilitado as permissões de desenvolvedor para que a depuração possar ser utilizada. A permissão de desenvolvedor abre para o executor do teste uma série de opções, tais como: instalação de aplicativos via USB, geração e captura de logs das aplicações, manipulação e iteração com dispositivo e a execução de programas de terceiros. Vale salientar que a opção de desenvolvedor é um processo que precisa ser habilitado somente uma única vez para cada dispositivo móvel utilizado enquanto uma restauração de fábrica não for executada.

Apesar da possibilidade de conexão sem fio, os autores optarão por uma conexão via USB, devido a conexão permitir uma maior estabilidade durante o processo de execução. Os autores utilizaram um HUB USB para que múltiplos dispositivos pudessem ser conectados, evitando assim a limitação de dispositivos que poderiam ser conectados a máquina local.

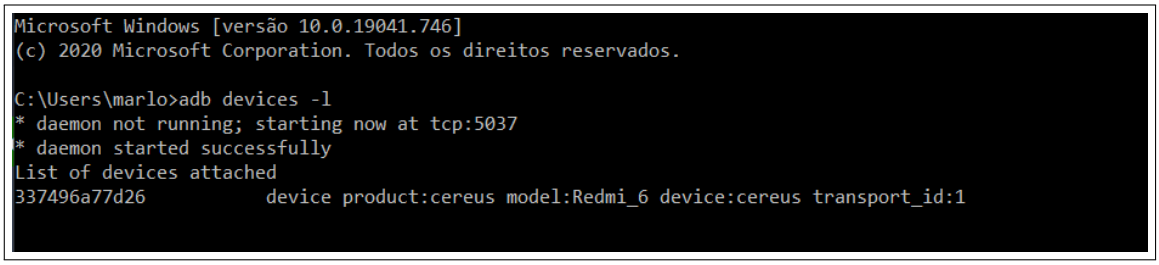
Com os dispositivos móveis conectados ao ambiente, dois *softwares* para comunicação da suíte de testes com o dispositivo são necessários. O primeiro deles é o Android Debug Bridge(ADB), que se trata de uma ferramenta de linha de comando presente na suíte Android Studio[1], que permite a comunicação com dispositivo móvel através de virtualização. Por meio dos comandos disponíveis no Android Debug Bridge(ADB), o usuário pode realizar uma série de operações como instalar e depurar aplicativos e acesso ao *Shell UNIX* que pode ser usado para executar diversos comandos em um dispositivo durante os testes.

Para esse trabalho, o Android Debug Bridge(ADB) foi utilizado para demonstrar

[1] https://developer.android.com/studio

se um dispositivo está realmente conectado ao seu ambiente, demonstrando o identificador do dispositivo conectado, e a porta utilizada para conexão. Neste trabalho os autores utilizaram a versão 1.0.41 do Android Debug Bridge. A Figura 12 abaixo demonstra os dispositivos atualmente conectados que são listados por meio do comando: adb devices -l:

Figura 12 – Log ADB devices.



Fonte: elaborado pelo autor (2021).

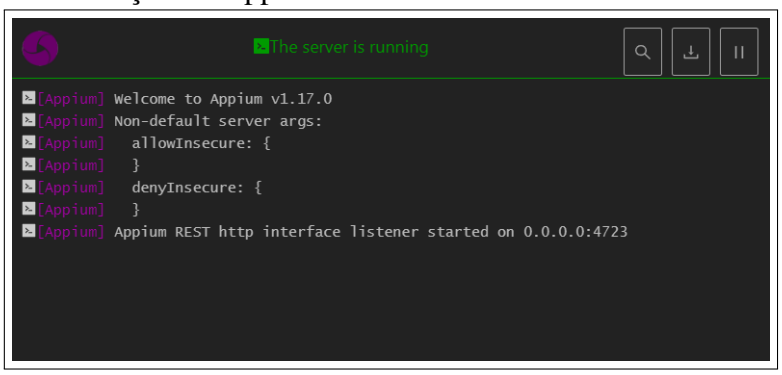
O segundo software necessário para que o processo de execução seja realizado com êxito é um cliente Appium. Como já explicado no capítulo de fundamentação teórica, o Appium permite a automatização de aplicações nativas e híbridas. Além da dependência do Appium no projeto de teste, se faz necessário um cliente para comunicação com os dispositivos conectados.

Existem duas formas disponíveis de utilização de um cliente Appium, a primeira através de um módulo Node[2], no entanto, essa forma necessita de uma versão do node.js e NPM instalado em sua máquina. A segunda, e usada pelos autores é o Appium desktop[3], um binário executável para as plataformas Linux, Windows e Mac. Por meio desse binário é possível adicionar alguns recursos extras como a inspeção de elementos de uma aplicação atualmente ativa. Uma vez instalado no seu ambiente, será apresentado uma tela ao executar o Appium desktop, a tela inicial permite primordialmente que você inicie o Appium por meio da opção start server.

Outras opções estão disponíveis na tela inicial do Appium desktop, permitindo configurações avançadas e a conexão em ambientes e *host* remotos. Como o propósito deste trabalho é execução local, os autores manterão as configurações como default e realizaram a execução do Appium por meio da opção start server. A versão do Appium desktop utilizada pelos autores nesse trabalho foi a 1.17.0. A Figura 13 demonstra a tela do Appium desktop após o servidor estar em execução, todas as chamadas de iteração com a aplicação podem ser acompanhadas por meio dessa tela.

[2] https://nodejs.org/en/
[3] https://github.com/appium/appium-desktop/

Figura 13 – Tela de execução do Appium



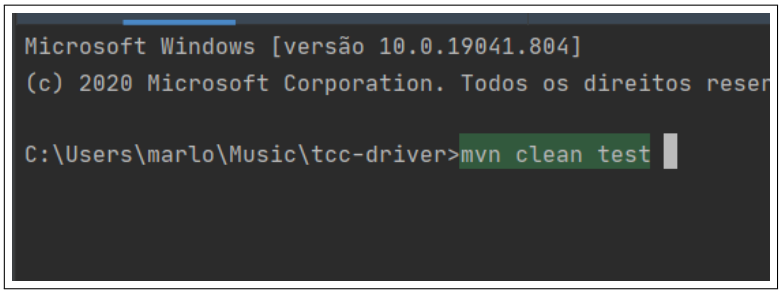
Fonte: elaborado pelo autor (2021).

A aplicação Greenmile Driver 7 é necessária para a execução do projeto, principalmente no momento em que a suite de testes realiza a instalação do arquivo .APK nos devices conectados. Atualmente existe uma pasta no projeto de nome driver, onde o APK da aplicação deve existir no momento de execução do projeto.

4.3.2 Execução

Com toda a parte de configuração do ambiente realizada, o processo de execução de testes pode ser realizado. Uma vez que o Maven foi escolhido pelos autores como gerenciador de dependências para o projeto, podemos utilizar os comandos disponíveis para execução do projeto via linha de comando em um terminal ativo da IDE utilizada. O comando básico para a execução do projeto é: "mvn clean test", esse comando executa todos os testes com a anotação @Test, conforme a Figura 14 a seguir:

Figura 14 – Comando para a execução dos testes via Maven.

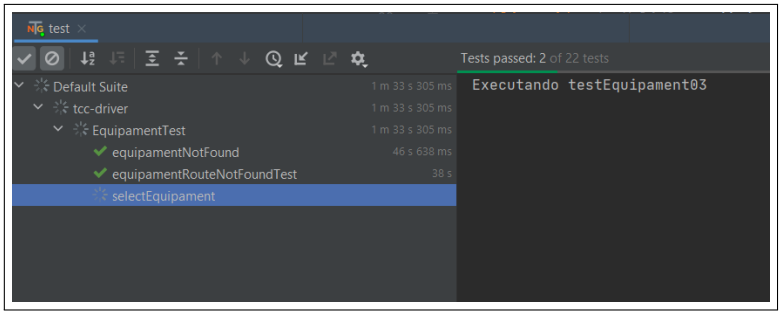


Fonte: elaborado pelo autor (2021).

A execução dos scripts de automação de teste pode ser acompanhada através da tela de execuções da IDE utilizada, bem como no console onde o comando de execução do MVN foi realizado. O acompanhamento da execução é mais prático quando realizado por meio da área de execuções da própria IDE, pois, demonstra o teste atual em execução, sendo possível

também verificar o resultado de assertivas existentes no teste. A Figura 15 a seguir demonstra por exemplo o resultado de alguns dos testes executados por meio do dashboard da IDE.

Figura 15 – Status de execução dos testes via IDE.



Fonte: elaborado pelo autor (2021).

4.4 Cenário de teste em nuvem

A seguir será descrito o processo para execução dos projetos de testes em um ambiente de nuvem.

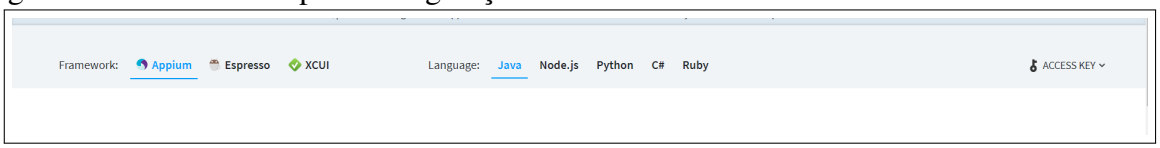
4.4.1 Preparação do ambiente

Diferente do processo de execução de forma local, o processo de execução em um ambiente de nuvem não envolve a preparação de dispositivos físicos para utilização, todos os dispositivos estão no próprio ambiente de nuvem, são iniciados e preparados para o teste de forma totalmente automática. O usuário que irá realizar a execução do projeto precisa apenas definir variáveis de ambiente. As variáveis de ambiente são utilizadas por exemplo para definir qual modelo de dispositivo móvel será usado, ou que versão do Android[4] será utilizada, caso o teste seja para dispositivos da plataforma Android.

Toda a configuração pode ser realizada por meio da dashboard do App-automate do Browserstack. O processo de configuração segue um fluxograma dividido em 4 steps, os steps definem critérios para os testes à nível de configuração, como: a estrutura de testes do projeto linguagem que o projeto de testes foi desenvolvida, chave de acesso (conhecida como *Secret key*), e uma chave de autenticação utilizada para interação com o ambiente Browserstack. A Figura 16 ilustra a primeira seção de configuração da página do *App-automate*, onde o usuário escolhe o *Framework* de testes, linguagem que o projeto foi desenvolvido e tem acesso à sua *Secret key*:

[4] https://www.android.com/intl/pt-BR$_br$/

Figura 16 – Primeiro step de configuração do ambiente de nuvem.



Fonte: elaborado pelo autor (2021).

A seguir será exemplificado todo o processo de configuração do ambiente de nuvem, iniciando com o *upload* do aplicativo Greenmile Driver7 para o ambiente.

O processo de *upload* da aplicação que será utilizada para teste pode ser realizada de duas formas para o ambiente de nuvem Browserstack. A primeira é utilizando o *upload* de arquivos disponíveis na própria plataforma, onde o usuário seleciona através de um gerenciador de arquivos o arquivo APK. Após o APK ser carregado para plataforma, é gerado então um código *token* utilizado como variável de ambiente para as configurações de acesso a plataforma de testes em nuvem.

A segunda forma de envio da aplicação a ser testada pode ser feito via API. Utilizando sua *Secret key* o usuário pode enviar um arquivo APK disponível em um diretório do seu gerenciador de arquivos, após o envio, deve-se utilizar o *token* que é enviado como resposta à chamada da API. A Figura 17 ilustra o comando utilizado para o *upload* via API:

Figura 17 – Comando de envio de APK via API.

```
curl -u "USER:KEY" -X POST "https://api-cloud.browserstack.com/app-automate/upload"
-F "file=@/path/to/app/file/Application-debug.apk"
```

Fonte: elaborado pelo autor (2021).

A última etapa necessária para execução no ambiente de nuvem é a configuração do script de teste. Configurar o script de teste visa definir parâmetros para a execução do projeto no ambiente de nuvem utilizado. Visto que o projeto foi desenvolvido utilizando o *Framework* Appium, essas definições são chamadas de *capabilities*. Para que a execução com o ambiente de nuvem seja estável durante todo o processo de execução dos testes, a primeira *capibilitie* que temos que definir são as credenciais de acesso ao ambiente Browserstack, sendo elas a *browserstack.user* e *browserstack.key*, essa etapa é necessária para autenticar seus testes junto ao ambiente de nuvem. A Figura 18 ilustra a definição das *capability* de credenciais de acesso. Os autores precisaram borrar as informações das credenciais por motivo segurança:

Figura 18 – Capibilities de autenticação no sistema.

```
20
21        DesiredCapabilities caps = new DesiredCapabilities();
22
23        // Set your access credentials
24        caps.setCapability("browserstack.user", "[redacted]");
25        caps.setCapability("browserstack.key", "[redacted]");
26
```

Fonte: elaborado pelo autor (2021).

No processo de upload do aplicativo para o ambiente de nuvem Browserstack é gerado um *token*, esse *token* deve ser usado para definir uma *capabilitie* referente ao aplicativo que será usado no ambiente de nuvem para execução dos testes, a Figura 19 ilustra o trecho de código necessário para a utilização do aplicativo Greenmile Driver7 na execução dos testes no ambiente de nuvem:

Figura 19 – Capibilities de definição do aplicativo.

```
// Set URL of the application under test
caps.setCapability("app", "bs://9e98c7daa013dec3b586d637c93cc355acde99be");
```

Fonte: elaborado pelo autor (2021).

A próxima *capibility* que o usuário precisa definir para execução no ambiente de nuvem está relacionado ao dispositivo utilizado para teste, uma série de dispositivos podem ser selecionados para a plataforma Android e IOS. Ao acessar o dashboard do App-automated existe um step específico para a escolha do dispositivo usado, esse step fornece para o usuário um seletor, para que seja possível escolher o sistema operacional mobile e dispositivo. A Figura 20 ilustra essa área de definição:

Figura 20 – Capibilities de definição do dispositivo.

```
// Specify device and os_version for testing
caps.setCapability("device", "Google Pixel 3");
caps.setCapability("os_version", "9.0");
```

Fonte: elaborado pelo autor (2021).

Uma vez selecionado o dispositivo e sistema operacional é gerado um trecho de código referente a *capabilitie* para o uso dos dispositivos na execução dos testes.

Apesar de a configuração ter sido realizada por meio dos steps apresentados na

página do App-automate, essa configuração também pode ser realizada via linha de comando, ou diretamente no código do projeto. O tutorial dos steps tem finalidade didática, para ajudar o usuário a ter conhecimento de quais *capability* são obrigatórias para seu projeto. Vale salientar que uma série de outras *capibilities* podem ser definidas, aumentando as restrições ou tornando seu teste muito mais específico. A Figura 21 é ilustrado uma imagem com as *capibilities* mínimas necessárias para o projeto no ambiente de nuvem Browserstack:

Figura 21 – Capibilities minímas necessárias.

```
20
21        DesiredCapabilities caps = new DesiredCapabilities();
22
23        // Set your access credentials
24        caps.setCapability("browserstack.user", "marlohenrique1");
25        caps.setCapability("browserstack.key", "2sTzxpV9sCXW2qAo385V");
26
27        // Set URL of the application under test
28        caps.setCapability("app", "bs://9e98c7daa013dec3b586d637c93cc355acde99be");
29
30        // Specify device and os_version for testing
31        caps.setCapability("device", "Google Pixel 3");
32        caps.setCapability("os_version", "9.0");
33
34        // Set other BrowserStack capabilities
35        caps.setCapability("project", "First Java Project");
36        caps.setCapability("build", "Java Android");
37        caps.setCapability("name", "first_test");
38
39
40        // Initialise the remote Webdriver using BrowserStack remote URL
41        // and desired capabilities defined above
42          AndroidDriver<AndroidElement> driver = new AndroidDriver<AndroidElement>(
43              new URL("http://hub.browserstack.com/wd/hub"), caps);
44
45
```

Fonte: elaborado pelo autor (2021).

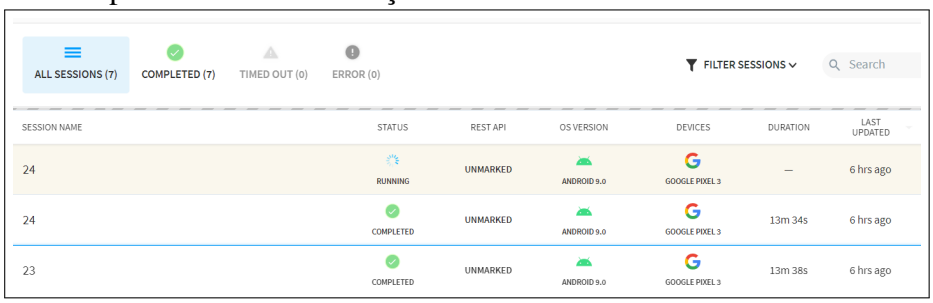
4.4.2 Execução

O processo de execução no ambiente de nuvem segue o mesmo processo definido no ambiente local, uma vez que todas as configurações necessárias foram realizadas é executar o comando do MVN referente a execução dos testes.

O processo de execução pode ser acompanhado pelo usuário através da tela de execuções da IDE utilizada, assim como no processo de execução local. No entanto, a plataforma em nuvem Browserstack permite que a execução também seja acompanhada através do dashboard de execução do App-automate, conforme podemos visualizar na Figura 22 abaixo:

Por meio da dashboard é possível acompanhar de forma instantânea a reprodução do teste, sendo possível um feedback em tempo real caso algum problema seja detectado no teste. Após a execução, uma captura de vídeo da execução de testes fica disponível para o usuário,

Figura 22 – Acompanhamento de execução via ambiente de nuvem.



Fonte: elaborado pelo autor (2021).

podendo ser usado para download ou reprodução futuras. Outros recursos também podem ser acessados após o término de um teste como, o textlog de todos os passos executados no teste, logs de network, logs do device e logs gerados pelo appium na

Essa série de informações disponíveis após a execução dos testes é de extrema importância em situações onde o teste apresenta alguma instabilidade. Identificar se o problema foi causado por um erro ou devido à existência de um falso negativo na aplicação se torna um processo extremamente simples quando temos dados gerados após a execução.

5 RESULTADOS E DISCUSSÃO

A seguir serão apresentados os resultados comparativos após a execução da suíte de testes utilizando dispositivos reais em ambiente local e dispositivos reais em ambiente de nuvem. Uma crítica comparativa entre as etapas de preparação e execução também será apresentada previamente com a análise e problemas presenciados pelos autores. Em seguida os dados de tempo de execução entre ambiente e gráficos comparativos.

5.1 Etapa de preparação

O primeiro ponto que merece ser comparado entre a execução do projeto de forma local e remota é a etapa de preparação do ambiente, como essa é uma etapa primordial para execução do projeto os autores levantaram diversos pontos que atenuam as vantagens de uma forma de execução sobre outra.

No ambiente de nuvem todo o processo de configuração e preparação dos dispositivos para torná-los aptos a execução é um processo totalmente desnecessário, não existe a necessidade de verificação dos dispositivos em modo debug, verificação se os equipamentos foram reconhecido pela suíte de testes, se os dispositivos estão com uma capacidade de carga compatível para a execução entre outros pontos que precisam de uma verificação de forma manual no ambiente de teste local.

Houve sempre a necessidade de verificação dos dispositivos no ambiente real estarem com o modo de debug ativo, pois o processo de reconhecimento via comando ADB só ocorre após o dispositivo estar conectado. Houve uma instabilidade do ambiente local na conectividade dos dispositivos, com dispositivos sendo reconhecidos somente em determinados momentos, sendo necessário em muitas situações realizar um completo *reset* no dispositivo móvel para reconhecimento e confirmação de conexão via comando ADB. A instabilidade também foi vista durante a execução, fatores externos como o ambiente influenciaram a perda da conexão em muitos momentos por pequenos contatos com o cabo de conexão USB.

As dependências para execução do projeto de testes também foram um ponto que diferenciou os dois ambiente. O Appium é executado de forma automática ao executar o projeto em um ambiente de nuvem, no entanto no ambiente local existe a necessidade de iniciar o Appium previamente para que o projeto seja executado com sucesso. Em muitos casos, houve problema na execução do Appium localmente, esse problema de execução da dependência

resultava muitas vezes em falsos negativos no reconhecimento de elementos ou na execução de comandos nos dispositivos.

Em resumo, a etapa de configuração do ambiente é um processo extremamente mais prático quando realizado no ambiente de nuvem Browserstack, visto que quase todas as etapas necessárias de preparação realizadas localmente são feitas de forma automatizada pelo ambiente de nuvem utilizado para essa pesquisa. A documentação disponibilizada pelo ambiente de nuvem contorna as possíveis dúvidas e ajuda em possíveis problema durante a configuração.

5.2 Etapa de execução

Os critérios de execução são semelhantes em ambos os ambientes, local e em nuvem. O mesmo comando do maven é utilizado para a execução, com diferença apenas no parâmetro de execução que define qual dos ambientes será usado. No entanto, uma vez executado o comando do maven, o acompanhamento da execução é totalmente distinto em ambos os ambientes. O ambiente local peca pela falta de informação sobre o estágio que o teste se encontra, se determinado teste está realizando a instalação do APK, se o processo de inicialização do dispositivo está sendo realizado, esses mínimos detalhes dificultam na verificação de uma ordem correta no fluxo de execução.

O ambiente de nuvem Browserstack contorna todos os problemas citados anteriormente, existe todo um sistema de logs que fica registrado ao se executar um teste em um determinado fluxo de execução. Esses logs evidenciam todas as etapas realizadas pelo ambiente, desde a preparação do dispositivo, instalação do APK até o ponto mais importante que é a execução do scripts de teste, os logs facilitam a rastreabilidade de possíveis problema, visto que podemos chegar diretamente no comando que não foi possível ser executado.

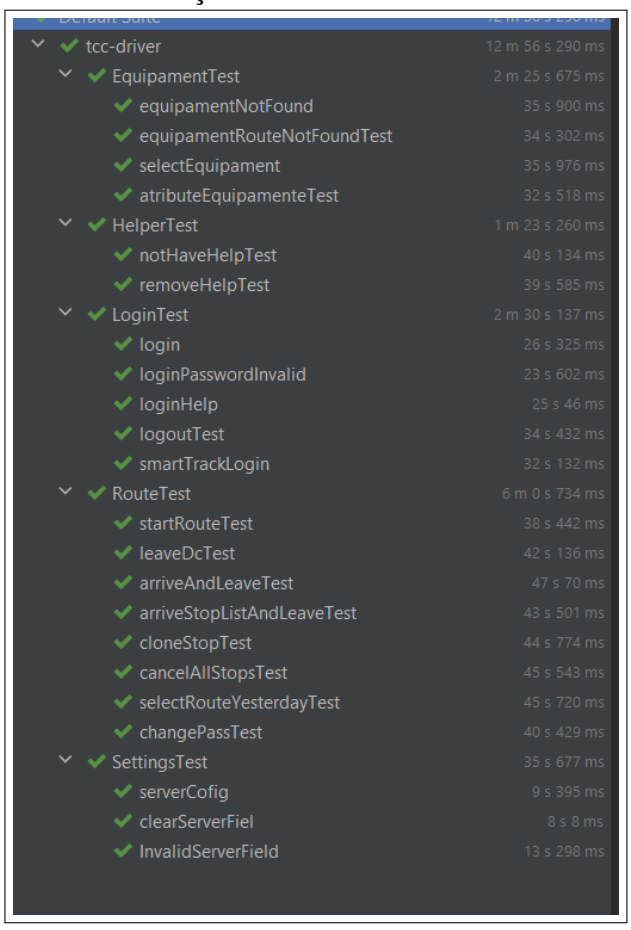
Em situações onde um determinado teste não executa no ambiente local, seja por perda de conexão com o dispositivo ou na localização de um seletor ou execução de um comando, existe um sério problema na continuidade dos testes. O dispositivo utilizado tende a ficar preso em muitos momentos no teste defeituoso, prejudicando a execução dos demais. No ambiente de nuvem temos a cada execução um recurso totalmente novo disponível, ou seja, quando um teste é executado, o dispositivo utilizado realiza o próximo teste em um estado de *restarted*.

Na etapa de execução fica evidente que o ambiente de nuvem Browserstack como ferramenta para a execução de testes mobile facilita no rastreamento de problemas e na auto recuperação na ocorrência de problemas de execução.

5.3 Resultados

Para a aferição dos resultados, foram realizados 33 execuções em ambos os ambientes, sendo utilizado o mesmo dispositivo móvel. A cada execução, eram armazenados em uma planilha o tempo total de execução do projeto, a quantidade de falhas ocorridas na execução, sendo distinguidos entre falhas reais na aplicação (verdadeiro negativo) ou falhas no ambiente(falso negativo). Foram registrados evidências dos *logs* fornecidos pela IDE onde o projeto de testes foi executado. A Figura 23 exemplifica o registro da 20º execução utilizando o ambiente de nuvem BrowserStack.

Figura 23 – Resultados da 20º execução utilizando ambiente de nuvem.

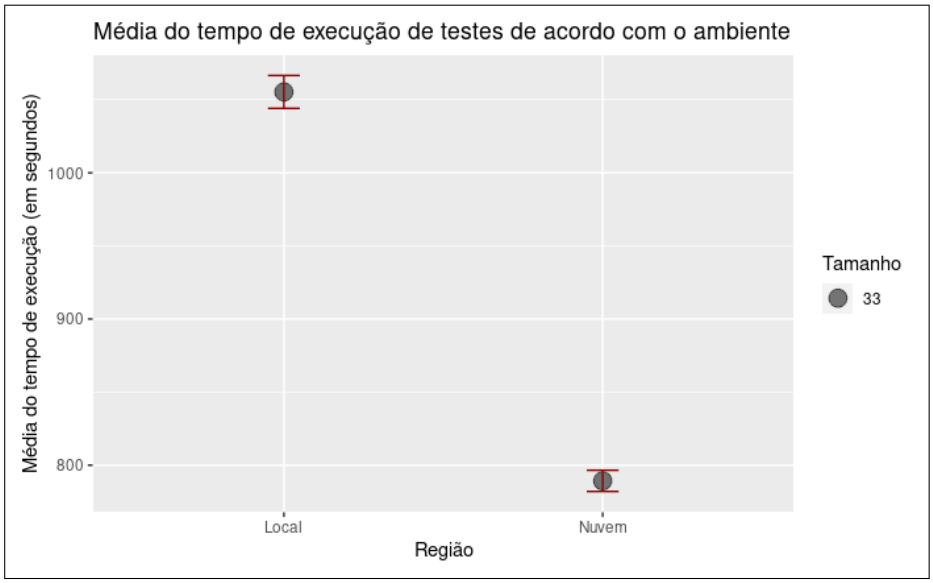


Fonte: elaborado pelo autor (2021).

O primeiro ponto analisado após o processo de execução foi a media de tempo para as 33 execuções em ambos os ambientes. Foi realizada a conversão para segundos de todos os tempos obtidos após execução, como forma de garantir que todas as medidas de tempo estivessem em uma mesma unidade base. Após a conversão e cálculo da média se observou um tempo médio inferior para o processo de testes utilizando o ambiente de nuvem.

O gráfico da Figura 24 mostra uma redução de aproximadamente 25% no tempo médio entre as execuções, onde o ambiente local tem 1055*s* e o ambiente de nuvem com média aproximada de 789*s*. Como forma de apresentar uma garantia dos resultados obtidos, se calculou o intervalo de confiança de 95% para os resultados após o ciclo de execução dos testes, a Figura 24 apresenta o gráfico dos resultados dos intervalos de confiança para os testes em ambiente local e utilizando ambiente de computação em nuvem:

Figura 24 – Comparação dos intervalos de confiança calculados.



Fonte: elaborado pelo autor (2021).

Os resultados apresentados na Figura 24 apontam um maior nível de incerteza sobre a média para os resultados dos testes em ambiente local e apontam um menor nível de incerteza sobre os resultados dos testes utilizando o ambiente de nuvem Browserstack. Para a execução local o valor do intervalo de confiança calculado foi de aproximadamente 11,24*s* com um limite superior e inferior com os respectivos valores: 1066.27*s* 1043,79*s*. Já o resultado do intervalo de confiança do ambiente de nuvem apresentou valor aproximado de 7.25*s* com um limite superior e inferior de tempo com os respectivos valores: 797.31*s* e 782.56*s*, apontando um maior nível de segurança sobre as estimativas calculadas.

Por meio dos resultados da Figura 24 é possível descartar uma outra hipótese, a influência da latência de comunicação entre máquina local de execução dos testes e o ambiente de nuvem utilizado para pesquisa. Mesmo com atraso recorrente da comunicação entre máquina local de execução e ambiente de nuvem, os resultados finais apontam benefícios em se adotar esse modelo de testes em nuvem para a realização de testes mobile.

A quantidade de falhas ocorridas durante a etapa de execução foi outro fator impre-

visto, com uma alta margem de diferença entre os testes em ambos os ambientes. Os autores buscaram identificar pontos que possam ter influenciado em uma alta quantidade de falhas no ambiente local, todo o processo pré e pós testes sempre foi realizado em todas as 33 execuções, independente disso, se observou casos onde apenas 1 teste falhava e casos onde a máxima de 10 testes falharam durante a execução em ambiente local.

O processo pré e pós teste tinha como objetivo eliminar vestígios de testes realizados anteriormente, para isso, sempre se realizava o processo de desinstalar o aplicativo utilizado na seção anterior, reiniciar o dispositivo móvel para a limpeza de cache, e encerrar a seção do Appium desktop para garantir que a nova sessão não teria vestígios de busca de elementos de execuções anteriores.

É importante destacar que a grande maioria das falhas ocorridas durante o processo de execução de forma local se resumem a falsos negativos, ou seja, devido a inconsistência no testes ou no ambiente utilizado para execução. Foi descartado a possibilidade de inconsistência nos testes, uma vez que o mesmo projeto foi utilizado para a etapa de execução dos testes em nuvem, não tendo sido observado uma alta na quantidade de falhas. Em algumas situações da execução do projeto de testes de forma local era esperado que o Appium retornasse a existência de determinando elemento ou informação que em tese estavam visíveis na aplicação. A ausências do fornecimento desse recurso prejudicava a continuidade do fluxo de execução considerando aquele testes como falho.

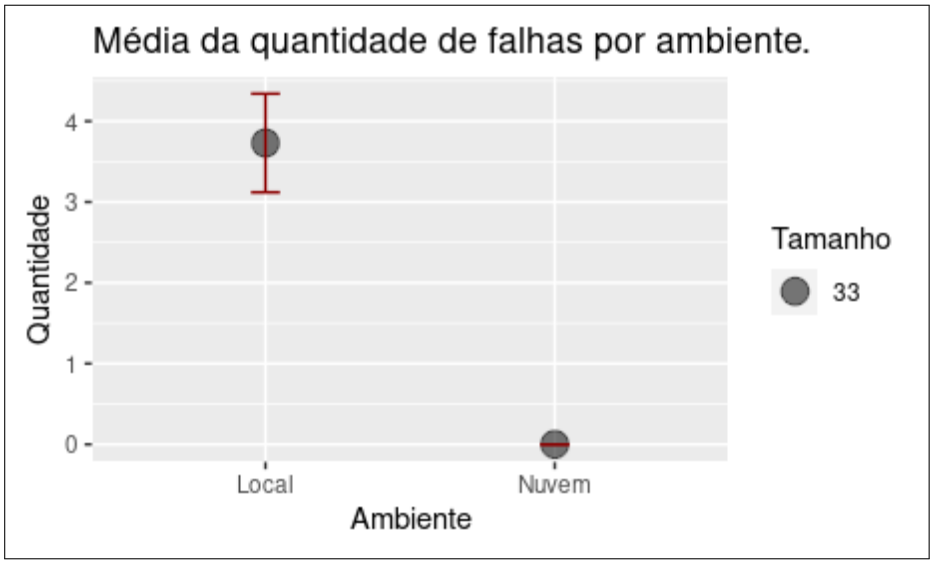
Para a pesquisa realizada, foi possível observar que o ambiente de testes local teve influência na quantidade de falsos negativos, no entanto, devido ao tempo não foi possível realizar um investigação sobre qual elemento específico seria o responsável pelas falhas ocorridas. A hipótese levantada aponta três prováveis causadores: problemas de conexão entre dispositivo móvel utilizado, possibilidade de problemas na instância do Appium executada localmente e por fim a falta de poder/recursos computacionais oferecidos pelo ambiente local para a execução dos testes.

É notório que para os três pontos citados anteriormente o ambiente de nuvem Browserstack apresenta benefícios e diferenças na forma de funcionamento que torna visível a menor quantidade de falsos negativos presentes na execução. Os recursos computacionais escaláveis permitem que o ambiente de nuvem tenham uma maior estabilidade nos seus resultados alocando ou desalocando recursos sob demanda ou mesmo fornecendo uma máquina com uma maior quantidade de recursos computacionais que o ambiente local. Os autores buscaram

investigar as configurações da máquina utilizada pelo ambiente de nuvem, no entanto não tiveram acesso a essas informações.

Foi calculado a média de erros após as 33 execuções, as diferenças podem ser observadas na Figura 25 onde podemos observar o intervalo de confiança para ambos os ambientes, com uma alta diferença entre os resultados. Para a execução de forma local o valor do intervalo de confiança calculado foi de aproximadamente 0,61 com um limite superior e inferior com os respectivos valores: 3,11 e 4,33. Já para o ambiente de nuvem o intervalo de confiança tem valor 0, apontando que não existe diferença entre as médias.

Figura 25 – Intervalos de confiança calculados.



Fonte: elaborado pelo autor (2021).

A quantidade de falhas média do ambiente local influenciou nos resultados de tempo médio apresentados. Uma vez que um determinado elemento não era encontrado para a realização de um comando para a execução do teste, existe um *timout* definido de 15 segundos que aguardava o elemento surgir na interface da aplicação antes de considerar o elemento como ausente e torna o teste como falho. A Figura 26 exemplifica o método da classe BasePage para a espera implícita por elementos da interface da aplicação:

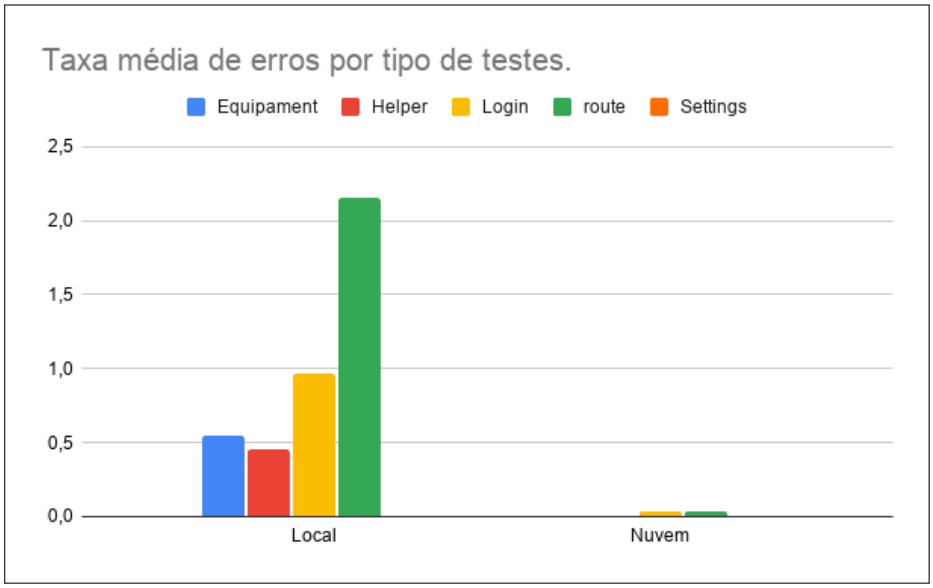
Figura 26 – Espera implícita de elementos.

```
public MobileElement findElement(String element){
    WebDriverWait wait = new WebDriverWait(DriverFactory.getDriver(), timeOutInSeconds: 15);
    wait.until(ExpectedConditions.presenceOfElementLocated(new By.ByXPath(element)));
    return DriverFactory.getDriver().findElementByXPath(element);
}
```

Fonte: elaborado pelo autor (2021).

É importante destacar que os erros gerados durante a execução do projeto se distribuíram entre os cenários cobertos da aplicação, levantando a hipótese mais um vez que os erros ocorridos em ambiente local tiveram como influência uma limitação nos recursos computacionais disponíveis. Na Figura 27 é possível observar que houve problemas de execução em 4 dos 5 cenários existentes, sendo o cenário de testes de *route* o com maior incidência de erros com base na média calculada. Já para o cenário de teste em nuvem não é possível visualizar a existência de erros entre todos os 5 cenários de testes.

Figura 27 – Média de erros por tipo de teste.



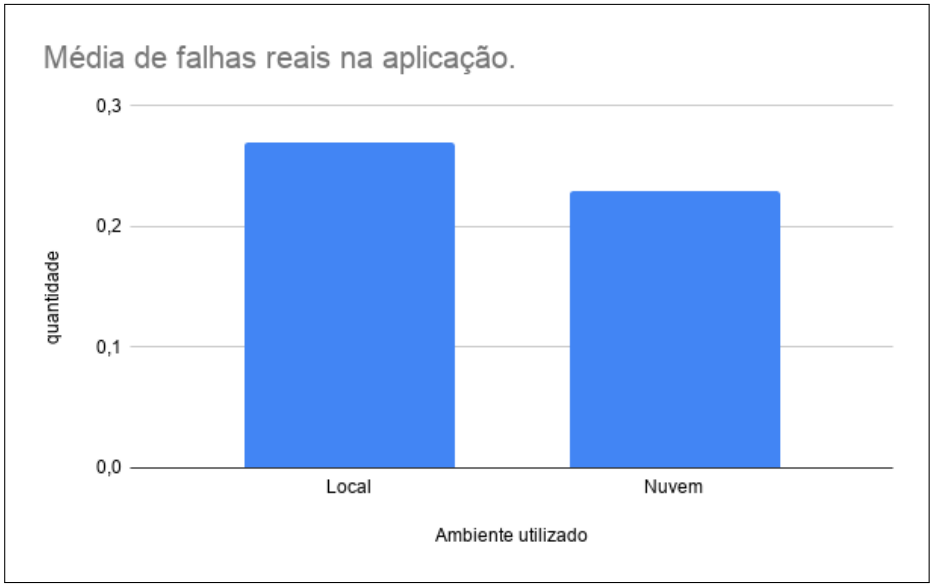
Fonte: elaborado pelo autor (2021).

Durante a etapa de execução do projeto de testes foi possível observar a presença de verdadeiros negativos, ou seja, falhas reais na aplicação, pois um resultado esperado não foi atingido. A verificação da presença dos erros foi possível devido a existência de assertivas em diferentes etapas dos testes desenvolvidos, como transição entre telas, presença de textos ou a mudança de estado da aplicação. Como existia a necessidade de criação de uma massa de dados para realização do teste, assertivas foram utilizadas para verificar se a criação e exclusão dos dados havia ocorrido de forma exata, caso algum problema fosse detectado, como um *status code* de uma requisição com código diferente de 200, essa falha também foi considerada como verdadeiro negativo, uma vez que o fluxo de criação de dados fazia parte do cenário de testes.

A Figura 28 apresenta a média de falhas capturadas durante os testes realizados utilizando os ambientes de teste local e em nuvem, é possível observar que diferente de todos os resultados apresentados até o momento, os verdadeiros negativos apresentaram a média mais próxima entre os ambientes utilizados na pesquisa sendo que o ambiente local apresentou uma

média de 0,27, já o ambiente de nuvem Browserstack apresentou uma média de 0,23 erros para 33 execuções.

Figura 28 – Média de falhas reais na aplicação.



Fonte: elaborado pelo autor (2021).

6 CONCLUSÃO E TRABALHOS FUTUROS

A proposta do presente trabalho foi verificar as melhorias obtidas no processo de teste de software quando adotado a utilização de ambientes de computação em nuvem para a realização de testes de software mobile baseado em dispositivos reais, bem como os ganhos e viabilidade de adoção desse modelo de testes por empresas que utilizam ambiente local de testes. Este trabalho tinha como um dos objetivos secundários a implementação de um projeto de testes mobile baseado em uma aplicativo móvel como forma de propor a adoção da infraestrutura de nuvem em um cenário real de uma empresa.

Discutido todo o trabalho e apresentados os resultados obtidos na pesquisa, bem como a modelagem do projeto, conclui-se que a utilização de ambientes de computação em nuvem que forneçam dispositivos reais para testes mobile se torna viável, visto que apresentam uma benéfica diferença dos resultados em relação a tempo de execução e quantidade de erros gerados quando comparado com um ambiente de teste local. É possível afirmar que a utilização de ambiente de computação em nuvem para testes mobile é de grande utilidade, pois facilita o processo de construção e implantação de um projeto de testes mobile, fornecendo uma série de informações de fácil acesso sobre os testes realizados bem como o fácil acesso a uma alta gama de dispositivos existentes no mercado, sendo estratégia chave de adoção por uma empresa ou organização.

Durante o desenvolvimento do trabalho foram encontradas dificuldades, principalmente nas etapas de criação de massa de dados para os testes, onde se optou por uma comunicação via API utilizando uma *basic auth* fornecida para comunicação e criação de massa de dados necessários para os testes. A etapa de escolha do ambiente de nuvem utilizado na pesquisa demandou uma longa análise de ambientes existentes, verificando como principais pontos: o suporte a dispositivos reais; suporte ao *Framework* Appium.

As limitações de poder computacional da máquina utilizada localmente para execução dos testes pode ter tido influência em uma série de resultados negativos para a execução de forma local, com uma alta taxa de erros. Devido às limitações de tempo para a pesquisa, não foi possível investigar a fundo qual componente do processo de execução local estava limitando os testes devido a poder computacional, sendo uma recomendação de trabalho futuro uma investigação sobre o gerenciamento de recursos e poder computacional durante a execução de um projeto de testes mobile.

A quantidade de dispositivos móveis disponíveis em diferentes sistemas e modelos

para a execução de forma local limitou que pudessem ser realizados experimentos multiplataforma, sendo realizados testes somente em dispositivos com o sistema Android e especificamente em um único modelo. Vê-se nos trabalhos futuros a necessidade de verificação de resultados comparativos quando utilizado dispositivos com o sistema IOS.

A principal contribuição do trabalho foi a melhoria relatada do tempo de execução com a adoção de ambientes de computação em nuvem existentes no mercado, suprindo uma ausência apresentada nos trabalhos relacionados. O trabalho torna-se relevante para pesquisas futuras, pois fornece uma confirmação de viabilidade de adoção destes ambientes para testes, e possibilitando a adaptabilidade a outras realidades.

6.1 Trabalhos futuros:

Como sugestão para trabalhos futuros a serem pesquisados:

- Viabilidade de utilização de dispositivos virtualizados por meio de computação em nuvem;
- Execução de forma distribuída utilizando ambiente de nuvem;
- Investigação sobre o consumo e disponibilidade de recursos computacionais em testes mobile;
- Testes para a plataforma IOS utilizando ambiente de computação em nuvem;
- Execução dos testes em nuvem utilizando ferramentas de integração contínua;
- Propor uma otimização dos testes utilizando algoritmos de agendamento de trabalhos.

REFERÊNCIAS

APPIUM. **Automation for Apps**. 2020. Disponível em: <http://appium.io/>.

APPLE. **Comprar iPhone 12 Pro e iPhone 12 Pro Max**. 2020. Disponível em: <https://www.apple.com/br/shop/buy-iphone/iphone-12-pro>.

APPLEGATE, D.; COOK, W. A computational study of the job-shop scheduling problem. **ORSA Journal on computing**, INFORMS, v. 3, n. 2, p. 149–156, 1991.

BARTIÉ, A. **Garantia da qualidade de software**. [S.l.]: Gulf Professional Publishing, 2002.

BROWSERSTACK. **Real Device Cloud**. 2020. Disponível em: <https://www.browserstack.com/live/features>.

BRUNS, A.; KORNSTADT, A.; WICHMANN, D. Web application tests with selenium. **IEEE software**, IEEE, v. 26, n. 5, p. 88–91, 2009.

CANDEA, G.; BUCUR, S.; ZAMFIR, C. Automated software testing as a service. In: **Proceedings of the 1st ACM symposium on Cloud computing**. [S.l.: s.n.], 2010. p. 155–160.

DEVICEFARM. **Melhore a qualidade de seus aplicativos móveis e da Web testando em navegadores de desktop e dispositivos móveis reais hospedados na Nuvem AWS**. 2020. Disponível em: <https://aws.amazon.com/pt/device-farm/>.

DUARTE, A. N. *et al.* Uma abordagem baseada em testes automáticos de software para diagnóstico de faltas em grades computacionais. Universidade Federal de Campina Grande, 2010.

EMERY, D. H. Writing maintainable automated acceptance tests. In: **Agile Testing Workshop, Agile Development Practices, Orlando, Florida**. [S.l.: s.n.], 2009.

FEWSTAR, M.; GRAHAM, D. **Software Testing Automation: Effective use of test execution tools**. [S.l.]: ACM Press, Addison Wesley, 1999.

FOWLER, M. **Test Pyramid**. 2012. Disponível em: <https://martinfowler.com/bliki/TestPyramid.html>.

GAMBI, A.; KAPPLER, S.; LAMPEL, J.; ZELLER, A. Cut: Automatic unit testing in the cloud. In: **Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis**. New York, NY, USA: Association for Computing Machinery, 2017. (ISSTA 2017), p. 364–367. ISBN 9781450350761. Disponível em: <https://doi-org.ez11.periodicos.capes.gov.br/10.1145/3092703.3098222>.

GAO, J.; TSAI, W.-T.; PAUL, R.; BAI, X.; UEHARA, T. Mobile testing-as-a-service (mtaas)–infrastructures, issues, solutions and needs. In: IEEE. **2014 IEEE 15th International Symposium on High-Assurance Systems Engineering**. [S.l.], 2014. p. 158–167.

GENYMOTION. **Android In the Cloud (AIC): A server edition of Android that is fully compatible with ADB & offers HTTP APIs & live streaming in web browsers.** 2020. Disponível em: <https://www.genymotion.com/cloud/>.

Guo, C.; Zhu, S.; Wang, T.; Wang, H. Fet: Hybrid cloud-based mobile bank application testing. In: **2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)**. [S.l.: s.n.], 2018. p. 21–26.

JOSEP, A. D.; KATZ, R.; KONWINSKI, A.; GUNHO, L.; PATTERSON, D.; RABKIN, A. A view of cloud computing. **Commun. ACM**, v. 53, n. 4, p. 50–58, 2010.

Jun, W.; Meng, F. Software testing based on cloud computing. In: **2011 International Conference on Internet Computing and Information Services**. [S.l.: s.n.], 2011. p. 176–178.

KILINÇ, N.; SEZER, L. *et al.* Cloud-based test tools: A brief comparative view. **Cybernetics and Information Technologies**, Sciendo, v. 18, n. 4, p. 3–14, 2018.

KIM, G.-H.; KIM, Y.-G.; CHUNG, K.-Y. Towards virtualized and automated software performance test architecture. **Multimedia Tools and Applications**, Springer, v. 74, n. 20, p. 8745–8759, 2015.

KOçU, A. **Appium Architecture for Android Testing**. 2017. Disponível em: <https://medium.com/mobile-application-test-automation/android-testleri-i%C3% A7in-appium-mimarisi-d4097d97626e>.

KUHN, E. **Selenium Grid Setup: The Complete Guide**. 2019. Disponível em: <https: //medium.com/maestral-solutions/selenium-grid-setup-the-complete-guide-cf000a2be50f>.

Lou, Y.; Zhang, T.; Yan, J.; Li, K.; Jiang, Y.; Wang, H.; Cheng, J. Dynamic scheduling strategy for testing task in cloud computing. In: **2014 International Conference on Computational Intelligence and Communication Networks**. [S.l.: s.n.], 2014. p. 633–636.

Ma, X.; Wang, N.; Xie, P.; Zhou, J.; Zhang, X.; Fang, C. An automated testing platform for mobile applications. In: **2016 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)**. [S.l.: s.n.], 2016. p. 159–162.

MENEZES, L. **Rodando testes em paralelo com Appium, Selenium Grid e Java: Parte 2**. 2018. Disponível em: <https://medium.com/assertqualityassurance/ rodando-testes-em-paralelo-com-appium-selenium-grid-e-java-parte-2-ae67977bdd92>.

MESEVAGE, T. G. **What's the Difference Between Cloud and Virtualization?** 2019. Disponível em: <https://www.datto.com/library/ whats-the-difference-between-cloud-and-virtualization>.

MOLINARI, L. **Testes de Software: Produzindo Sistemas melhores e mais confiáveis**. [S.l.]: Erica, 2008.

NAGANO, M. S.; MOCCELLIN, J. V.; LORENA, L. A. N. Programaçao da produçao flow shop permutacional com minimizaçao do tempo médio de fluxo. **XXXVI Simpósio Brasileiro de Pesquisa Operacional. Sao Joao Del-Rei**, v. 3, p. 4, 2004.

OR-TOOLS, G. **The Job Shop Problem**. 2020. Disponível em: <https://developers.google. com/optimization/scheduling/job_shop>.

PARVEEN, T.; TILLEY, S. When to migrate software testing to the cloud? In: IEEE. **2010 Third International Conference on Software Testing, Verification, and Validation Workshops**. [S.l.], 2010. p. 424–427.

RAFI, D. M.; MOSES, K. R. K.; PETERSEN, K.; MÄNTYLÄ, M. V. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In: IEEE. **2012 7th International Workshop on Automation of Software Test (AST)**. [S.l.], 2012. p. 36–42.

ROJAS, I. K. V.; MEIRELES, S.; DIAS-NETO, A. C. Cloud-based mobile app testing framework: Architecture, implementation and execution. In: **Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing**. New York, NY, USA: Association for Computing Machinery, 2016. (SAST). ISBN 9781450347662. Disponível em: <https://doi.org/10.1145/2993288.2993301>.

SATASIYA, P. **Top 6 Advantages of using Cloud Automation Testing for your Enterprise**. 2018. Disponível em: <https://dzone.com/articles/top-6-advantages-of-using-cloud-automation-testing>. Acesso em: 23 abr. 2020.

SKYONE. **4 tendências em tecnologia que vão transformar os negócios**. 2017. Disponível em: <https://skyone.solutions/pb/4-tendencias-em-tecnologia-que-vao-transformar-os-negocios/>.

SOMMERVILLE, I. **Engenharia de software**. [S.l.]: Person Prenticial Hall, 2011.

TAN, L. **Intro to Parallel Testing: How It Works, Benefits, Challenges: by Perforce**. 2020. Disponível em: <https://www.perfecto.io/blog/parallel-testing>.

TAURION, C. **Cloud computing-computação em nuvem**. [S.l.]: Brasport, 2009.

VIEIRA, L. S.; BARRETO, C. G. L.; SANTOS, E. B. dos; ARAGÃO, B. S.; SANTOS, I. de S.; ANDRADE, R. M. C. Test automation in a test factory: an experience report. In: **Proceedings of the XIV Brazilian Symposium on Information Systems**. [S.l.: s.n.], 2018. p. 1–8.

APÊNDICE A – QUADRO COMPARATIVO ENTRE OS AMBIENTES BROWSERSTACK, SAUCELABS E DEVICE FARM

Tabela 2 – Comparativo ambientes

Pontos analisados	Browserstack	Saucelabs	Devive Farm
Framework Suportados	Junit, TestNG, Cucumber JS, Protractor, MBUnit, Specflow, Behat, PHPUnit, Behave, Lettuce, RSpec etc.	Selenium, Appium, XCTest, Espresso	Appium Java Junit, Appium Java TestNG, Appium Python, Calabash, JUnit, Espresso, Robotium, XCTest
Devices Suportados	110+ devices	373 devices	373 devices
OS Suportados	Android, iOS, Windows	Android, iOS	iOS, Android and Fire OS
Faixa de preços	O preço começa a partir de $ 29 / mês.	O preço começa em $ 19 / mês.	0,17$ / minuto do dispositivo.
Compatibilidade de automação/DevOps	Jenkins, Travis, Bamboo, TeamCity, CircleCI, Bitbucket.	Visual Studio Team Services, Bitbucket Pipelines, Bamboo, Jenkins, TeamCity	Jenkins
Real mobile devices	SIM	SIM	SIM
Utilização sem custos	Licença para estudantes de 1 ano, sem custos e com acesso a todos os recursos	100 minutos	1000 minutos gratis
Facilidade de integração	Fácil de ser integrado, documentação demostra os passos necessários para configuração em diferentes linguagens e frameworks	Médio de ser integrado. inclui alguns exemplos na documentação para ser integrado.	Dificil de ser integrado, necessidade necessidade de um arquivo .yml onde se definir toda a configuração, de framework, linguagem, e configuração do ambiente.

APÊNDICE B – AUTORIZAÇÃO PARA REALIZAÇÃO DE TESTES AUTOMATIZADOS NO APP DRIVER7

Em 14 de outubro de 2020 6:28, Marlo Henrique <moliveira@greenmile.com> escreveu:

Dears, good morning!

My name is Marlo Henrique, I am currently completing a bachelor's degree in Software Engineering at UFC (Federal University of Ceará). I am developing my course conclusion work focused on the mobile testing area, with the title "CLOUD TEST: IMPROVED EFFICIENCY IN TEST EXECUTION USING CLOUD COMPUTING".

My proposal is to evaluate test automation tools, where possible schedule automated tests and run them on cloud services that are capable of running tests on different real or virtualized devices and compare real gains when compared to using devices in a real environment and when using devices offered by computing environments in the cloud.

Phase 1 of this research aimed to demonstrate the feasibility of the project, presenting a small set of tests that were performed in a generic application, using the cloud environment BrowserStack. The proposal was approved by a panel on September 8, 2020.

Phase 2 has as main objective to demonstrate the functioning of the tools chosen in phase 1, in addition to proving the real gains that were obtained using the execution of tests in a cloud environment. We need to run some tests on an application developed by a third party in phase 2.

For being part of the quality team of the Driver7 application, I believe that it would be an interesting application to carry out this research. Because it is an application available on the two main platforms on the market, IOS and Android, and because it has compatibility with a testing framework that the authors intend to use in this work.

We would like your authorization, so that if possible, carry out the tests in the Driver7 application version 7.12.32, with a strictly academic character, in order to evaluate the testing tools for mobile devices chosen for study.

Awaiting return! Att, Marlo Henrique.

Resposta: André Campos <acampos@greenmile.com.br> 14 de out. de 2020 14:35 para mim, Régis

Excellent, Marlo!

I absolutely approve such research using GreenMile's Driver 7 application. Please

before publicly available, please send us a copy of the document. This will help us understand and evolve our processes and product. Also we can quickly validate if there is any confidential information exposed. Besides that, you can absolutely proceed with your research.

Great work!

APÊNDICE C – CENÁRIOS DE TESTE UTILIZADOS.

Tabela 3 – Cenários de testes da aplicação automatizados

Casos de teste	**Cenário**
[CT_001] - Carregar equipamento inexistente	equipamento
[CT_002] - Equipamento sem rota atribuida	equipamento
[CT_003] - Carregar equipamento existente	equipamento
[CT_004] - Carregar equipamento e atribuir a rota	equipamento
[CT_005] - Ajudante não atribuido a rota na tela de helper	Helper
[CT_006] - Remover ajudante atribuido na tela de helper	Helper
[CT_007] - Login com driver realizado com sucesso	Login
[CT_008] - Login com driver com senha invalida	Login
[CT_009] - Login com ajudante.	Login
[CT_010] - Logout test	Login
[CT_011] - Login com driver do tipo smarttrack	Login
[CT_012] - Iniciando uma rota	Route
[CT_013] - Realizando saida do deposito	Route
[CT_014] - Realizar atendimento de uma parada	Route
[CT_015] - Realizar atendimento a parti da lista de paradas.	Route
[CT-016] - Clonando uma parada	Route
[CT_017] - Cancelar todas as paradas restantes.	Route
[CT_018] - Carregar rota de um dia anterior	Route
[CT_019] - Troca senha do driver	Route
[CT_020] - Preencher informações da tela de settings	Settings
[CT_021] - Preencher tela de setting com campo de servidor em branco	Settings
[CT_022] - Preencher tela de setting com servidor invalido	Settings