Implementação de Testes Automatizados Mobile em Servidores Linux: Um Relato de Experiência em um Banco Público

Jonnathan Riquelmo[1], Maicon Bernardino[1], Elder de Macedo Rodrigues[1]

[1]Laboratoy of Empirical Studies in Software Engineering (LESSE),
Engenharia de Software, Universidade Federal do Pampa (UNIPAMPA)
Av. Tiarajú, 810 - Bairro Ibirapuitã - Alegrete, RS

jonnathan.riquelmo@gmail.com, bernardino@acm.org, eldermr@gmail.com

Abstract. *Test automation is crucial to ensuring software quality and security in financial institutions, particularly in a public bank where reliability is critical for transactions. Traditionally, mobile test execution was restricted to Windows servers due to infrastructure limitations. The study reports the experience of migrating to a Linux-based architecture using Selenium Grid and Appium, integrated with Jenkins. This shift enabled greater flexibility, scalability, and efficiency, significantly improving the test automation process. The study also addresses the technical challenges overcome, such as authentication system integration and security within the bank's network.*

Resumo. *A automação de testes é essencial para garantir a qualidade e a segurança do software em instituições financeiras, especialmente em um banco público, onde a confiabilidade é crítica para as transações. Tradicionalmente, a execução de testes mobile estava restrita a servidores Windows, devido a limitações de infraestrutura. O estudo relata a experiência de migração para uma arquitetura baseada em servidores Linux, usando Selenium Grid e Appium integrados ao Jenkins. Essa mudança permitiu maior flexibilidade, escalabilidade e eficiência, resultando em na melhoria no processo de automação de testes. Também aborda os desafios técnicos superados, como a integração com sistemas de autenticação e a segurança dentro da rede do banco.*

1. Introdução

A qualidade do software é um fator crítico em qualquer organização, especialmente em instituições financeiras, onde a confiabilidade e a segurança das aplicações são essenciais para a operação contínua e segura das transações. No contexto de um banco público, em que a demanda por qualidade e precisão é elevada, a automação dos testes funcionais surge como uma solução estratégica para assegurar software de alta qualidade com maior rapidez. Tradicionalmente, os testes funcionais de aplicações web e *mobile* foram realizados de forma manual ou parcialmente automatizada, limitando a cobertura de testes e expondo os sistemas a potenciais riscos de falhas no ambiente de produção.

Neste cenário, a implementação de uma solução de automação completa e integrada, capaz de lidar com a complexidade dos sistemas bancários, torna-se um objetivo estratégico. Este artigo apresenta a experiência de implementar uma arquitetura de testes automatizados para aplicações *mobile* em um banco público, utilizando o *Selenium*

Grid e o *Appium*, e integrando essa solução na esteira DevOps do *Jenkins*. A principal motivação para este trabalho foi a necessidade de superar as restrições da infraestrutura existente, a qual dependia de servidores Windows® para a execução dos testes *mobile* devido a limitações técnicas, como o uso de DLLs (*Dynamic-Link Library*) específicas. A nova abordagem, baseada em servidores Linux, possibilitou flexibilidade e escalabilidade, otimizando a performance dos testes ao reduzir a carga de dependências proprietárias [Mozgovoy and Pyshkin 2018].

Além disso, a integração com o *Jenkins* facilitou a automação contínua dos testes, permitindo que as equipes de QA (*Quality Assurance*) desenvolvessem e executassem os testes de forma mais ágil e eficaz. A implementação desta solução envolveu várias etapas, desde a PoC (*Proof of Concept* ou Prova de Conceito) inicial, que validou a viabilidade técnica, até a adaptação da arquitetura existente para suportar os novos requisitos. Este estudo detalha cada uma dessas etapas, discutindo as decisões de projeto, os desafios técnicos enfrentados e as soluções adotadas, além de apresentar uma análise dos resultados obtidos com a nova arquitetura de testes.

A estrutura do artigo é organizada da seguinte forma: na Seção 2, é apresentada a fundamentação teórica, que inclui uma visão geral sobre Selenium Grid, Appium e testes funcionais paralelos. Na Seção 3, são discutidos os trabalhos relacionados ao tema. A Seção 4 descreve as decisões de projeto tomadas durante a implementação. Na Seção 5, a arquitetura adotada é detalhada. Em seguida, a Seção 6 apresenta os resultados obtidos e a discussão das melhorias realizadas. A Seção 7 traz considerações sobre a infraestrutura utilizada, e, finalmente, a Seção 8 conclui o estudo e sugere trabalhos futuros.

2. Fundamentação Teórica

Esta seção explora o *Selenium Grid*, o *Appium*, os Testes Funcionais Paralelos e Distribuídos, assim como a dificuldades de integrar essas tecnologias em uma rede altamente segura e monitorada.

2.1. Selenium Grid

O *Selenium Grid* é uma ferramenta amplamente usada na execução distribuída de testes automatizados. Ele permite que os testes sejam executados em múltiplas máquinas simultaneamente, usando uma arquitetura *Hub-nodes*. O *Hub* é responsável por gerenciar os testes e distribuí-los para os nós, os quais são máquinas configuradas para executar esses testes em diferentes ambientes [Fischer et al. 2023]. Isso é útil em cenários em que a diversidade de plataformas e navegadores é crítica para garantir a qualidade da aplicação.

No contexto do banco público, o motivo da escolha pelo *Selenium Grid* foi pela necessidade de realizar testes em múltiplas versões de SOs e dispositivos móveis. No entanto, a implementação em uma rede de alta segurança apresentou desafios significativos. A intranet do banco, projetada para proteger dados sensíveis, impôs várias restrições de comunicação e acesso, o que complicou a configuração e a operação do *Selenium Grid*. A comunicação entre o *Hub* e os nós, por exemplo, precisou ser meticulosamente configurada para garantir que as conexões fossem seguras e que os testes não comprometessem a integridade da rede. Isso envolveu a implementação de medidas adicionais de segurança, como autenticações via protocolos HTTPS e o uso de túneis VPN, além de ajustes nas políticas de *firewall* para o tráfego de dados entre os componentes do *Selenium Grid*.

2.2. Appium

O *Appium* é uma ferramenta de código aberto que permite a automação de testes em aplicativos móveis, suportando tanto *Android*® quanto iOS. Ele utiliza uma arquitetura semelhante ao *Selenium WebDriver*, o que facilita a integração com o *Selenium Grid* e permite a execução de testes automatizados em uma ampla gama de dispositivos e sistemas operacionais móveis [Villanes et al. 2017].

No ambiente do banco, o *Appium* foi escolhido pela sua flexibilidade e compatibilidade com a infraestrutura existente, além de sua capacidade de realizar testes tanto em dispositivos físicos quanto em emuladores. Contudo, a integração do *Appium* dentro da intranet do banco também apresentou desafios. A necessidade de garantir que os testes não comprometessem a segurança dos dados exigiu uma análise minuciosa de como o *Appium* interagia com os dispositivos e emuladores.

A configuração dos emuladores *Android*®, *e.g.* precisou ser ajustada para operar dentro das limitações impostas pela rede do banco. Isso incluiu a restrição de acesso a certos recursos do sistema e a implementação de controles rigorosos de visibilidade e *logs* sobre a comunicação entre o *Appium* e os dispositivos de teste. Além disso, a execução dos testes em um ambiente de rede vigiada exigiu a adoção de práticas de segurança adicionais, como o isolamento de ambientes de teste para prevenir acessos não autorizados.

2.3. Testes Funcionais Paralelos e Distribuídos

Os Testes Funcionais Paralelos e Distribuídos são uma prática que envolve a execução simultânea de múltiplos testes em diferentes ambientes. Essa prática é fundamental para acelerar o processo de testes, especialmente em *pipelines* de CI/CD (Integração Contínua/Entrega Contínua), em que a rapidez e a eficiência são essenciais [Vila et al. 2017]. A possibilidade de distribuir testes em diferentes nós e executá-los em paralelo permite reduzir drasticamente o tempo total de testes, melhorando a cobertura e a velocidade de *feedback* [Alenzi et al. 2022].

No caso do contexto deste estudo, a implementação de testes paralelos e distribuídos foi crucial para lidar com a complexidade dos sistemas e a necessidade de testar em várias plataformas *mobile*. No entanto, a execução desses testes dentro de uma rede altamente monitorada trouxe desafios adicionais. Cada teste executado precisava ser cuidadosamente controlado para garantir que o tráfego gerado não violasse as políticas de segurança do banco. Isso incluiu a configuração de canais de comunicação seguros entre os nós do *Selenium Grid* e o *Hub*, bem como o monitoramento constante do uso de recursos de rede para evitar qualquer impacto negativo no ambiente produtivo.

Além disso, a escalabilidade dos testes distribuídos exigiu ajustes na arquitetura da rede do banco para acomodar a execução de múltiplos testes simultaneamente, sem comprometer a segurança ou o desempenho. A adoção de práticas como a segmentação de rede e o uso de VPNs internas - neste caso a *HPE Aruba Networking* - foram necessárias para garantir que os testes pudessem ser realizados de maneira eficiente e segura.

3. Trabalhos Relacionados

Esta seção apresenta uma revisão de estudos relacionados que abordam a implementação e os desafios da automação de testes *mobile*, especialmente em ambientes que utilizam

ferramentas como Appium e Selenium Grid. Ao apresentar esses trabalhos, buscamos contextualizar o relato da experiência apresentada neste artigo, destacando as práticas e técnicas aplicadas em diferentes cenários de testes automatizados. A comparação com trabalhos anteriores permite validar as abordagens empregadas e demonstrar a relevância da nossa solução implementada no banco público.

Costa e Miranda (2023) realizou uma análise comparativa de diversas ferramentas open-source voltadas para testes funcionais de interfaces gráficas (GUI) *mobile* em ambientes de Integração Contínua (CI). O estudo avaliou os frameworks Detox, Appium, Calabash e Maestro, levando em consideração critérios como popularidade, facilidade de uso, tempos de execução e integração com *pipelines* de CI. Através de testes práticos realizados em aplicativos Android, o Appium se destacou pela sua flexibilidade e compatibilidade com diferentes plataformas, facilitando a integração em diversos ambientes CI/CD. Embora não seja a ferramenta mais rápida, a pesquisa concluiu que o Appium é adequado para cenários que exigem alta flexibilidade e personalização, alinhando-se à necessidade de adaptação em ambientes complexos, como o de um banco público.

Outro estudo relevante é o de Li (2023), que explorou a implementação de testes automatizados em dispositivos móveis utilizando Appium e Selenium WebDriver. O foco do trabalho foi na validação de funcionalidades em diferentes versões do sistema operacional Android, abordando os desafios da fragmentação no ecossistema *mobile*. Os resultados indicaram que a combinação do Selenium Grid com Appium permitiu uma execução mais eficiente e paralela dos testes, o que se relaciona diretamente com a abordagem adotada neste artigo.

Yu *et al.* (2021) propõem um *framework* de automação de testes mobile que usa visão computacional para reconhecimento de layout e imagens, permitindo a execução de *scripts* de teste em várias plataformas. Esse *framework*, chamado LIRAT, gera *scripts* independentes de plataforma e melhora a flexibilidade dos testes em diferentes sistemas operacionais. Usando Appium e Selenium, a abordagem oferece uma solução robusta para testes funcionais de interface, superando limitações específicas de portabilidade entre plataformas, algo similar ao *framework* utilizado no banco.

Por fim, o trabalho de AbuSalim (2021) foca na análise de desempenho de testes *mobile* em uma infraestrutura baseada em Selenium Grid, comparando diferentes técnicas de paralelização. Os autores identificaram que a orquestração dos testes em múltiplos nós resulta em uma melhoria significativa nos tempos de execução. Este resultado é consistente com as melhorias observadas na nossa implementação, que utiliza um conjunto de nós Linux para executar testes *mobile* em paralelo, reduzindo o tempo de *feedback* para as equipes de desenvolvimento.

4. Decisões de Projeto

A decisão de migrar a execução dos testes *mobile* de servidores Windows® para Linux foi impulsionada por várias considerações técnicas e operacionais. A principal motivação foi a necessidade de superar as limitações impostas pela infraestrutura existente, que dependia de DLLs específicas desenvolvidas em C# para autenticação e comunicação com outros sistemas. Essas DLLs não eram compatíveis com o ambiente Linux, o que impedia a execução dos testes fora do ambiente Windows®.

Para validar a viabilidade da migração, foi realizada uma POC que envolveu a

configuração de um ambiente de teste com o *Selenium Grid* e *Appium* em servidores Linux. O fluxo original da POC é apresentado na Figura 1. A POC demonstrou que era possível configurar emuladores *Android*® em servidores Linux e executar testes automatizados de forma eficiente. Além disso, a utilização de Linux ofereceu vantagens adicionais, como maior flexibilidade na gestão de recursos e a possibilidade de escalabilidade horizontal, permitindo adicionar novos nós de teste conforme necessário.

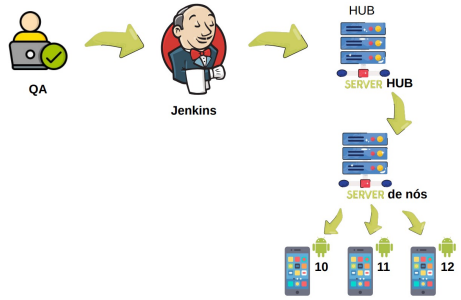

Figura 1. Fluxo em alto nível do POC.

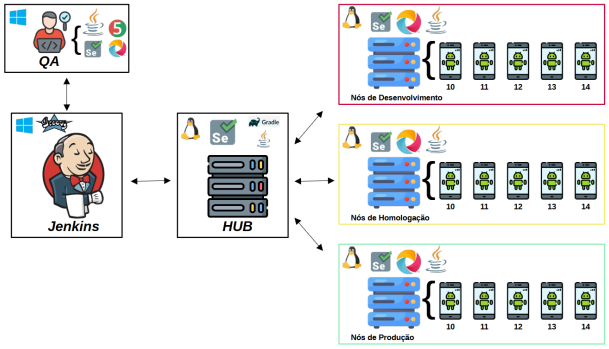

Figura 2. Arquitetura Atual.

Com base nos resultados da POC, foi decidido seguir em frente com a migração, adaptando a arquitetura do sistema para suportar a execução distribuída dos testes em Linux. Isso envolveu a reconfiguração do *Jenkins* para integrar-se ao novo ambiente e a implementação de soluções alternativas para as dependências de DLLs, como o desenvolvimento de APIs para autenticação na intranet do banco, além da reescrita de partes do código do *framework* interno proprietário de testes web e *mobile*.

5. Arquitetura

A Figura 2 apresenta a arquitetura da nova solução desenhada para maximizar a eficiência, escalabilidade e segurança dos testes automatizados em um ambiente bancário crítico. O núcleo da arquitetura é composto pelo *Hub* do *Selenium Grid*, configurado em um servidor Linux, que atua como ponto central de orquestração de testes. Os projetos de teste são desenvolvidos pelos QAs e versionados no SVN ou *GitLab* internos do banco. Nos *jobs* do *Jenkins* é indicado o repositório de onde deve ser feito o *clone*, bem como ambiente (desenvolvimento, homologação ou produção) e o AVD (*Android Virtual Device*, ou Dispositivo virtual Android). O *Hub* recebe as requisições de teste enviadas pelo *Jenkins*, e após o *clone* ele realiza basicamente um comando Gradle *buildFunctionalTest* do projeto e distribui essas tarefas para os nós disponíveis, que são servidores configurados com emuladores *Android*® de diferentes versões.

Cada nó é responsável por emular ambientes *Android*® específicos, variando de versões mais antigas às mais recentes do sistema operacional. Isso permite uma cobertura abrangente dos cenários de teste, garantindo que os aplicativos sejam validados em uma diversidade de configurações. Os nós também operam em modo *headless*, ou seja, sem interface gráfica, o que melhora a eficiência de processamento ao reduzir o consumo de recursos [Su et al. 2021]. O modo *headless* é particularmente importante para garantir que o uso de memória e CPU dos servidores seja otimizado, aumentando a capacidade de execução de múltiplos testes simultaneamente.

A distribuição de testes pelo *Selenium Grid* é um dos maiores ganhos de eficiência desta arquitetura. A orquestração inteligente realizada pelo *Hub* permite que testes sejam redirecionados para nós ociosos, mesmo que esses nós pertençam a diferentes ambientes nós. Isso ocorre de forma transparente, já que os emuladores *Android*® são idênticos, e a separação dos servidores como diferentes *labels* é semântica, destinada apenas a organizar a infraestrutura e garantir relatórios de testes agrupados por tipo de APK (*Android Application Package*) testado, ou seja, se são APKs de versão de desenvolvimento, homologação ou produção. Essa flexibilidade reduziu o tempo de espera em aproximadamente 40% para a execução dos testes, dando uma utilização mais eficiente dos recursos.

Outro aspecto importante da arquitetura é a integração contínua com o *Jenkins*, que atua como o coordenador dos *jobs* de teste. O *Jenkins* se conecta diretamente ao Hub do *Selenium Grid*, iniciando a execução de cada tarefa de teste e gerando os relatórios finais após a conclusão. A comunicação entre o *Jenkins* e o *Hub* é simplificada e direta, o que minimiza falhas e atrasos na execução dos testes, mesmo em grandes conjuntos de casos de teste (suítes). Após a conclusão dos testes, os resultados são agregados no *Hub*, processados e enviados de volta para o *Jenkins*, em que ficam disponíveis para as equipes de QA e desenvolvimento analisarem.

Além disso, a arquitetura foi desenhada com uma preocupação rigorosa em relação à segurança e integridade dos dados. Em um ambiente bancário, a segurança é uma prioridade, e a arquitetura adotada leva isso em consideração. A comunicação entre o *Hub* e os nós, bem como entre o *Jenkins* e o *Hub*, é protegida por mecanismos de autenticação (sistemas internos de geração de *tickets* para testes, produção de *logs* e rastreamento para auditoria) e criptografia SSL (*Secure Sockets Layer*), garantindo que dados sensíveis não sejam expostos durante a execução dos testes. Cada nó está isolado de maneira a evitar o acesso não autorizado, e a criptografia dos dados em trânsito assegura que informações confidenciais não sejam comprometidas.

Por fim, o *design* modular da arquitetura facilita a escalabilidade. Novos nós podem ser facilmente adicionados à estrutura existente, permitindo que a capacidade de execução de testes seja aumentada conforme necessário, sem a necessidade de reconfigurações complexas. Essa flexibilidade permite que o banco ajuste sua infraestrutura de acordo com a demanda, adicionando servidores quando o volume de testes aumenta, ou desativando nós em momentos de menor uso.

6. Resultados e Discussão de Melhorias

A implementação da nova arquitetura de testes automatizados trouxe melhorias significativas em comparação com a infraestrutura prévia. Antes da migração, os testes *mobile* eram executados exclusivamente em servidores Windows®, o que resultava em várias limitações, incluindo dependência de recursos proprietários, maior tempo de execução dos testes e dificuldades na integração contínua com o *Jenkins*. A nova solução, baseada em servidores Linux, eliminou essas restrições, proporcionando maior flexibilidade e eficiência na execução dos testes. Ainda, a visualização da execução dos testes era feita pelo *log* do *Jenkins*, o que dificultava o seu acompanhamento. Com o *Selenium Grid* possibilitou o acesso a uma visualização centralizada dos nós e seus emuladores, tornando mais intuitivo a visualização da atividades, a qual a Figura 3 apresenta essa visão.

A redução do tempo total de execução das suítes de testes é uma métrica impor-

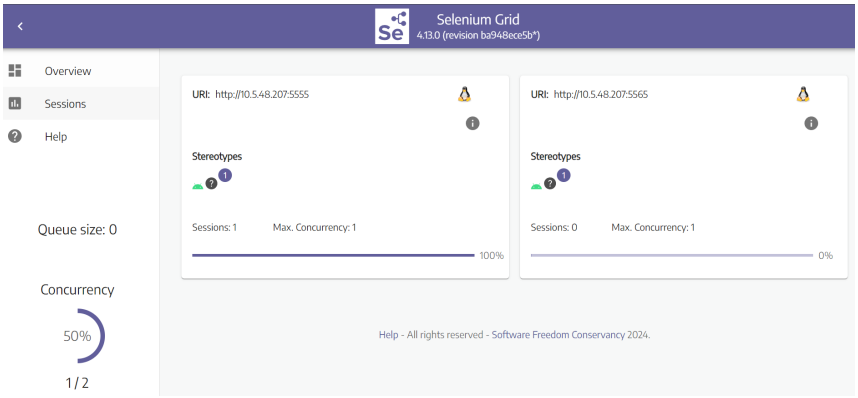

Figura 3. Visão geral fornecida pelo Hub do *Selenium Grid*.

tante, que consistem em uma média de 300 casos de teste em cada projeto por sistema. O ganho médio de tempo foi de aproximadamente 40%, graças ao alto poder de processamento dos servidores de nós e ao uso de emuladores em modo *headless* (sem interface gráfica), o que reduz a sobrecarga de execução. Além disso, o *Selenium Grid* orquestrou a distribuição dos testes para emuladores ociosos de nós diferentes, independente da segmentação semântica entre ambientes de desenvolvimento, homologação ou produção.

Sendo assim, uma das principais melhorias observadas foi a redução no tempo total de execução dos testes. Com a capacidade de executar testes em paralelo em múltiplos nós, o tempo necessário para testar uma versão completa das aplicações foi drasticamente reduzido. Isso permitiu que as equipes de QA entregassem resultados mais rapidamente, acelerando o ciclo de *feedback* para os desenvolvedores e permitindo a detecção precoce de problemas. A Figura 4 apresenta um dos testes realizados na POC sendo executados.

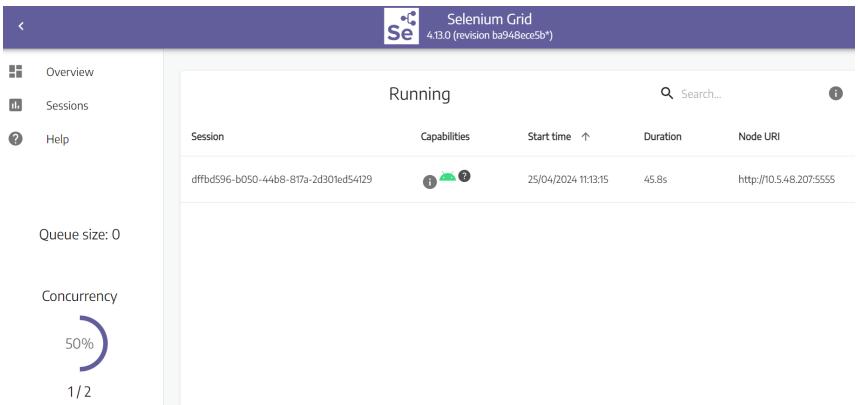

Figura 4. Teste sendo executado e visualizado no HUB.

KPIs (*Key Performance Indicators*, ou Indicadores-Chave de Desempenho) importantes utilizados no monitoramento da eficácia da solução foram definidos, como a cobertura de testes, que aumentou em 25% após a implementação da nova arquitetura, permitindo uma execução média de 150 casos de teste por nó. Ainda, a taxa de sucesso das execuções subiu para 98%, e o tempo médio do ciclo de *feedback*, que dentro do banco é quanto tempo deve levar para resolver *bugs* ou incidentes quando os mesmos são comunicados aos desenvolvedores, pôde ser redefinido para até 6 horas (tempo padrão do

expediente) após início dos testes, frente ao teto anterior que era de 18 horas, representando uma redução de 66,7%.

Além disso, a integração contínua com o *Jenkins* foi significativamente aprimorada. Com a nova arquitetura, os testes são executados automaticamente como parte do processo de CI/CD, garantindo que todas as mudanças no código sejam testadas antes de serem integradas ao *branch* principal. Isso não só melhora a qualidade do código, como também reduz o risco de introduzir falhas em produção, algo que é impossível de ser aceito no caso de grandes aplicações financeiras. A Figura 5 apresenta os dados de uma sessão iniciada com sucesso para testes funcionais de um aplicativo que faz parte da esteira de desenvolvimento.

Figura 5. Informações da sessão vistas no HUB.

Outra vantagem da nova solução foi a capacidade de escalabilidade. Em vez de depender de uma quantidade limitada de servidores Windows®, que no geral apresentam maiores dificuldades para configuração da infraestrutura, a nova arquitetura permite adicionar nós Linux conforme necessário, aumentando a capacidade de execução de testes sem comprometer o desempenho. Isso é especialmente importante em momentos de alta demanda, como durante a finalização dos ciclos de desenvolvimento antes dos períodos de *freezing*, ou antes de entregas importantes.

Como autor e colaborador no banco público, é importante ressaltar que, devido à natureza confidencial dos dados e relatórios gerados durante o processo de testes, não é possível apresentar esses resultados de forma concreta e detalhada. Esses dados são todos classificados como informações internas e sigilosas da instituição, e seu acesso é restrito às equipes autorizadas. No entanto, os resultados descritos neste artigo são baseados em observações e análises do desempenho geral da solução implementada.

7. Considerações sobre Infraestrutura e Desafios

Implementar uma arquitetura de testes automatizados distribuídos e paralelos dentro de uma rede bancária altamente segura é uma tarefa complexa que envolve superar diversos desafios técnicos e de infraestrutura. A migração dos testes *mobile* para servidores Linux, *e.g.* exigiu não apenas um conhecimento profundo das tecnologias utilizadas, mas também uma adaptação das políticas de segurança e dos processos de autenticação.

A configuração dos emuladores *Android*® como nós do *Selenium Grid*, por exemplo, foi um processo que exigiu a consideração de diversas restrições impostas pela

segurança da rede do banco. Isso incluiu a implementação de controles de acesso rigorosos, a criptografia de dados em trânsito, e a adaptação dos emuladores para operar de maneira isolada, garantindo que cada ambiente de teste fosse seguro e impenetrável.

A integração do *Selenium Grid* e do *Appium* com o *Jenkins* que opera em um ambiente Windows® é outro aspecto crítico. A comunicação entre esses diferentes SOs, dentro de uma rede altamente controlada, apresentou desafios significativos. Foi necessário desenvolver soluções que permitissem a comunicação segura e eficiente entre os servidores Linux que hospedavam o *Selenium Grid* e o servidor Windows® que operava o *Jenkins*, garantindo que a automação de testes funcionasse de maneira fluida e integrada.

Em resumo, a implementação de testes automatizados em um ambiente bancário não se limita apenas à configuração técnica das ferramentas. Ela também envolve uma adaptação às exigências de segurança e uma integração cuidadosa com a infraestrutura existente, tudo isso enquanto se garante que os testes possam ser executados de maneira eficiente e segura, sem comprometer a integridade dos dados ou a segurança da rede.

8. Conclusão e Trabalhos Futuros

A migração da execução dos testes *mobile* de servidores Windows® para Linux, utilizando *Selenium Grid* e *Appium*, representou um avanço significativo no processo de automação de testes no banco. A nova solução superou as limitações da infraestrutura, oferecendo maior flexibilidade, escalabilidade e eficiência na execução dos testes. A integração contínua com o *Jenkins* foi aprimorada, permitindo uma automação mais robusta e confiável, essencial para manter a qualidade do software em um ambiente crítico.

O relato do caso apresentado demonstra que, com as decisões de projeto corretas e uma abordagem estruturada, é possível superar desafios técnicos significativos e implementar uma solução de automação de testes eficaz. Os resultados alcançados mostram que a nova arquitetura não apenas melhorou a eficiência operacional, mas também contribuiu para a entrega de software de maior qualidade, com menor risco de falhas em produção.

Para trabalhos futuros, estão sendo realizados estudos preliminares com o objetivo de aprimorar a integração com os sistemas internos de autenticação e comunicação, incluindo os microsserviços desenvolvidos com um *framework* próprio em Java e as APIs desenvolvidas com *framework* proprietário em C#. Todos esses *frameworks*, criados pelos próprios colaboradores, possuem um legado de mais de 15 anos de existência e têm sido continuamente evoluídos para atender às crescentes demandas legais e da instituição.

Paralelamente, o *framework* de testes funcionais *web/mobile*, desenvolvido internamente em Java e implementado desde 2019, está em constante atualização para acompanhar as necessidades de qualidade e eficiência nos processos de desenvolvimento. Essas soluções foram projetadas para atender às necessidades específicas do banco, uma vez que nenhuma oferta de mercado cobria adequadamente o contexto da organização, seja por restrições de segurança ou por questões de controle e suporte contínuo.

Além disso, está sendo estudado o uso de *Containers Podman* em *runners* no novo *GitLab* interno do banco para evolução da solução, embora dificuldades estejam surgindo ao tentar executar emuladores *Android*® dentro de *containers*, uma vez que isso configura uma emulação dentro de outra. Ainda, estão sendo implementados *scripts* para gerenciamento de toda a infraestrutura utilizando o *OpenTofu*, a alternativa *open-*

source ao *Terraform*. O objetivo é a substituição completa dos *scripts .sh*, usados na construção e/ou destruição da infraestrutura, que hoje são invocados diretamente por meio de aplicações como o *PuTTy* e o *MobaXterm*.

Por fim, é importante destacar que o fluxo de testes web, o qual o *framework* de testes funcionais também suporta, está em processo de migração para ambientes Linux, o que permitirá uma maior uniformidade e eficiência na execução dos testes automatizados.

Referências

AbuSalim, S. W., Ibrahim, R., and Wahab, J. A. (2021). Comparative analysis of software testing techniques for mobile applications. In *Journal of Physics: Conference Series*, volume 1793, Kuala Lumpur, Malaysia. IOP Publishing.

Alenzi, A., Alhumud, W., Bryce, R., and Alshammari, N. (2022). A survey of software testing tools in the web development domain. *J. Comput. Sci. Coll.*, 38(2).

Costa, G. and Miranda, B. (2023). A comparative analysis of mobile ui testing frameworks in continuous integration environments. In *Proceedings of the 8th Brazilian Symposium on Systematic and Automated Software Testing*, SAST '23, New York, NY, USA. Association for Computing Machinery.

Fischer, S., Ramler, R., Assunção, W. K. G., Egyed, A., Gradl, C., and Auberger, S. (2023). Model-based testing for a family of mobile applications: Industrial experiences. In *Proceedings of the 27th ACM International Systems and Software Product Line Conference - Volume A*, SPLC '23, New York, NY, USA. ACM.

Li, J. and Cao, H. (2024). Design and implementation of api automation testing system for mobile hybrid mode based on appium technology. In *Proceedings of the 2023 7th International Conference on Electronic Information Technology and Computer Engineering*, EITCE '23, New York, NY, USA. Association for Computing Machinery.

Mozgovoy, M. and Pyshkin, E. (2018). Mobile farm for software testing. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, MobileHCI '18, New York, NY, USA. Association for Computing Machinery.

Su, T., Yan, Y., Wang, J., Sun, J., Xiong, Y., Pu, G., Wang, K., and Su, Z. (2021). Fully automated functional fuzzing of android apps for detecting non-crashing logic bugs. *Proc. ACM Program. Lang.*, 5(OOPSLA).

Vila, E., Novakova, G., and Todorova, D. (2017). Automation testing framework for web applications with selenium webdriver: Opportunities and threats. In *Proceedings of the International Conference on Advances in Image Processing*, ICAIP '17, New York, NY, USA. Association for Computing Machinery.

Villanes, I. K., Ascate, S. M., Gomes, J., and Dias-Neto, A. C. (2017). What are software engineers asking about android testing on stack overflow? In *Proceedings of the XXXI Brazilian Symposium on Software Engineering*, SBES '17, New York, NY, USA. Association for Computing Machinery.

Yu, S., Fang, C., Yun, Y., and Feng, Y. (2021). Layout and image recognition driving cross-platform automated mobile testing. In *Proceedings of the 43rd International Conference on Software Engineering (ICSE '21)*. IEEE.