Cloud AutoDroid: uma Arquitetura de Backend para Executar Serviços de IA Generativa na Nuvem

Luiz Felipe Laviola, Angelo Gaspar Diniz Nogueira, Diego Kreutz, Rodrigo Brandão Mansilha

[1]Laboratório de Estudos Avançados em Computação (LEA)
Programa de Pós-Graduação em Engenharia de Software (PPGES)
Universidade Federal do Pampa (UNIPAMPA)

luiz@laviola.dev, {angelonogueira.aluno, diegokreutz, mansilha}@unipampa.edu.br

Resumo. *Apresentamos a Cloud AutoDroid: uma arquitetura de software distribuída, baseada em virtualização leve, que disponibiliza ferramentas de Inteligência Artificial (IA) como serviço de forma simplificada e escalável horizontalmente. A arquitetura é flexível, permitindo a execução e o monitoramento de serviços e infraestruturas de IA, tanto atuais quanto futuros. Demonstramos a viabilidade técnica da proposta por meio de uma implementação da Cloud AutoDroid e de um conjunto de testes funcionais. Além disso, avaliamos a aplicabilidade da Cloud AutoDroid através de um estudo de caso, aplicado no projeto Malware DataLab, fomentado pela Rede Nacional de Ensino e Pesquisa (RNP).*

Abstract. *We present Cloud AutoDroid: a distributed software architecture, based on lightweight virtualization, that provides Artificial Intelligence (AI) tools as a service in a simplified and horizontally scalable manner. The architecture is flexible, allowing the execution and monitoring of both current and future AI services and infrastructures. We demonstrate the technical feasibility of the proposal through an implementation of Cloud AutoDroid and a set of functional tests. Additionally, we assess the applicability of Cloud AutoDroid through a case study within the Malware DataLab project, supported by the Brazilian National Education and Research Network (RNP).*

1. Introdução

O Malware DataLab[1] é um projeto fomentado pela Rede Nacional de Ensino e Pesquisa (RNP)[2] no âmbito do Programa Hackers do Bem[3], que tem como objetivo contribuir para a formação de profissionais na área de cibersegurança. O projeto visa oferecer um serviço que reduz a curva de aprendizado e facilita a investigação de técnicas avançadas de geração de dados sintéticos, com o propósito de expandir *datasets* úteis na classificação de aplicativos Android (maligno ou benigno).

A rápida evolução dos *malwares* impulsionados por inteligência artificial exige soluções escaláveis, tornando essencial o uso de contramedidas também baseadas

[1]https://malwaredatalab.github.io/
[2]https://rnp.br
[3]https://hackersdobem.org.br

em IA, como modelos preditivos [Meijin et al. 2022]. O desempenho dessas soluções depende diretamente da disponibilidade de grandes volumes de dados de alta qualidade para treinamento [AI & Data Today 2023]. Entretanto, a obtenção de amostras representativas e atualizadas, especialmente no contexto das mutações frequentes de *malwares* Android, representa uma barreira significativa para o desenvolvimento de classificadores eficientes [Miranda et al. 2022, Kouliaridis et al. 2020, Wang et al. 2019]. Além disso, estudos recentes apontam que muitos *datasets* disponíveis para a detecção de *malwares* Android sofrem de problemas como obsolescência, baixa qualidade e limitações de abrangência [Miranda et al. 2022], agravando ainda mais esse desafio. Para mitigar esse problema, técnicas como as *conditional* GANs (cGANs)[Nogueira et al. 2024a] têm sido exploradas, como no caso do DroidAugmentor [Casola et al. 2023] e, mais recentemente, da MalSynGen [Nogueira et al. 2024b].

Nesse contexto, estamos desenvolvendo um ambiente de execução e experimentação para que futuros "*Hackers do Bem*" possam realizar atividades como expandir conjuntos de dados rotulados de maneira rápida e confiável (isto é, com base em métricas de desempenho) usando modelos pré-treinados ou investigar, de maneira sistemática, múltiplos modelos novos. O *backend* do Malware DataLab é desafiador considerando requisitos como escalabilidade, elasticidade e disponibilidade. O sistema deve oferecer capacidade para atender potencialmente centenas de usuários simultâneos que utilizam recursos custosos de processamento, incluindo *Graphic Processing Units* (GPUs), de armazenamento (*datasets* na ordem de gigabytes) e de comunicação. Como primeiro passo, em trabalhos anteriores, propomos uma solução para execução do sistema de *backend* encapsulado em virtualização leve (i.e., contêiners) [Laviola et al. 2023].

Neste trabalho, avançamos na direção de uma solução para a orquestração de execução de contêineres na nuvem, enfrentando desafios como o gerenciamento de máquinas disponíveis (i.e., *workers*) para executar os contêineres (e.g., MalSynGen) e a distribuição eficiente desses contêineres entre os *workers*. Apresentamos uma nova arquitetura flexível, denominada Cloud AutoDroid, que aborda esses desafios. Demonstramos a viabilidade técnica da arquitetura por meio de testes funcionais realizados com uma implementação. Além disso, validamos sua aplicabilidade através de um estudo de caso no contexto do projeto Malware DataLab.

Embora a literatura cinza apresente diversas soluções relacionadas, como Apache Airflow[4], Argo Workflows[5], e Metaflow[6], nenhuma delas atende completamente aos nossos requisitos específicos. Nossa principal necessidade é uma solução que permita o uso de *workers* em um ambiente de computação distribuída, sem exigir que esses *workers* tenham acesso direto à infraestrutura protegida. Além disso, a configuração dos *workers* deve ser simples e rápida, permitindo uma implementação ágil e sem dependências complexas de tecnologias externas.

Soluções como o Apache Airflow, apesar de robustas, exigem uma curva de aprendizado elevada e uma configuração inicial complexa para alcançar o mesmo

[4]`https://airflow.apache.org/`

[5]`https://argoproj.github.io/workflows/`

[6]`https://metaflow.org/`

objetivo. Já ferramentas como o Metaflow, embora mais intuitivas para processos de ciência de dados, não oferecem a flexibilidade e modularidade necessárias para serem usadas como blocos de construção em uma arquitetura de *workers* distribuída e de fácil configuração.

O restante deste trabalho está organizado da seguinte forma: a arquitetura Cloud AutoDroid é apresentada na Seção 2. Em seguida, na Seção 3, discutimos a avaliação funcional e o estudo de caso realizado. Finalmente, apresentamos as conclusões e as perspectivas para trabalhos futuros na Seção 4.

2. Cloud AutoDroid

A execução dos serviços da Malware DataLab envolve uma série de requisitos não funcionais, como gerenciamento de dependências, manutenabilidade, flexibilidade, disponibilidade, escalabilidade e elasticidade. Embora a virtualização para execução local (e.g., AutoDroid) possa atender adequadamente cenários com um único usuário ou máquina hospedeira, essa abordagem se torna inadequada para cenários mais complexos, que envolvem múltiplos usuários e máquinas, como os previstos no Malware DataLab.

Em relação aos requisitos de segurança, adotamos o seguinte modelo de ameaça: (a) consideramos seguras as primitivas criptográficas, como funções de *hash* (e.g., SHA256) e MAC (e.g., HMAC), além de protocolos como TLS 1.3; (b) presumimos que o ambiente de execução é seguro (e.g., *datacenter* ou nuvem privada), dispensando o uso de protocolos como TLS entre os componentes internos do sistema; (c) confiamos nos administradores do sistema e da infraestrutura, ou seja, não consideramos a ameaça de *insider threats*; e (d) assumimos que a aplicação (e.g., MalSynGen) é confiável, sendo auditada para garantir que não introduza ameaças ao ambiente, como *backdoors* ou *malwares*.

A arquitetura geral da Cloud AutoDroid é apresentada na Figura 2, utilizando a notação C4[7]. Em resumo, a Cloud AutoDroid é destinada aos chamados "Hackers do Bem", usuários interessados em executar serviços como o MalSynGen [Nogueira et al. 2024b] na nuvem. A Cloud AutoDroid oferece uma API Gateway (componente AutoDroid), responsável por comandar um conjunto de máquinas distribuídas (componente Worker). Múltiplas instâncias do componente Worker são responsáveis pela execução de sistemas de processamento de *datasets* (componente Data Processor), como o MalSynGen.

A arquitetura detalhada da Cloud AutoDroid é ilustrada na Figura 2, utilizando a notação C4 no nível de contêiner. No topo da figura, está a camada de persistência, responsável pelo armazenamento dos dados. Abaixo dessa camada, encontra-se a API, que intermedia a comunicação entre os diferentes componentes. Na base da figura, um exemplo apresenta duas instâncias de *workers*, que executam as tarefas distribuídas pelo sistema.

O *API Gateway* é o componente central da Cloud AutoDroid, expondo uma API REST[8]. Ele se conecta a diversos componentes de persistência e gerencia múl-

[7] `https://c4model.com/`

[8] `https://restfulapi.net/`

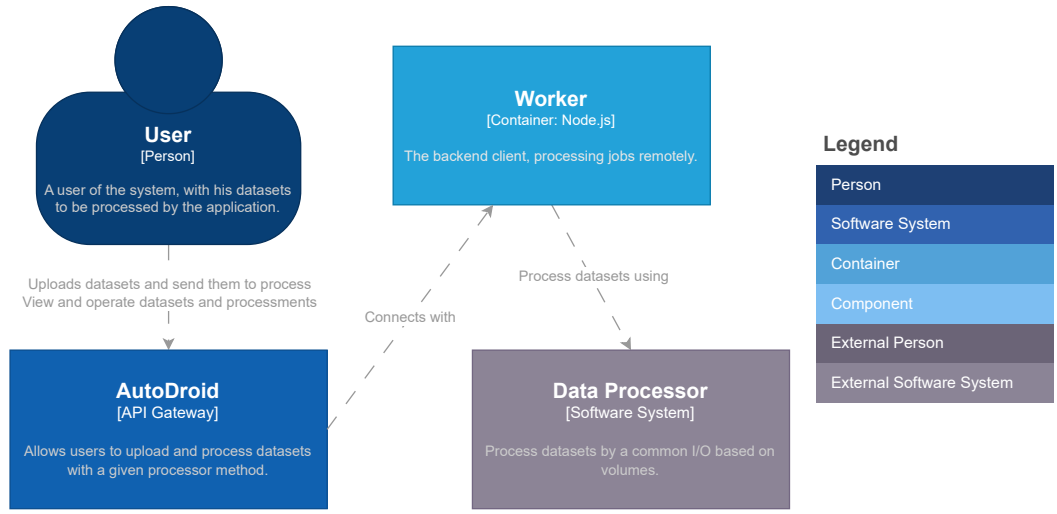

Figura 1. Arquitetura em nível de contexto de sistema.

tiplas instâncias de *Worker*, que executam instâncias de *Data Processor*. Essa organização possibilita a escalabilidade horizontal, permitindo que várias instâncias da API sejam executadas em paralelo, aumentando a disponibilidade da solução.

A camada de persistência é composta por três sistemas complementares. Um sistema gerenciador de banco de dados (SGBD) relacional, como o *PostgreSQL*[9], é usado para armazenar dados mestres, como informações de usuários e processos. Um SGBD não relacional baseado em memória, como o *Redis*[10], armazena informações transacionais, como o estado de execução dos *Workers*. Além disso, um SGBD não relacional orientado a documentos é utilizado para armazenar arquivos diversos, como entradas e saídas dos processamentos realizados pelos *Workers*. Foram mapeados quatro principais conceitos a serem persistidos: *User*, contendo atributos dos usuários; *Dataset*, que armazena informações sobre os arquivos de dados usados como entrada (dados reais) e saída (dados sintéticos); *Processment*, que representa a execução de aplicações de IA; e *Worker*, que se refere às máquinas (físicas ou virtuais) utilizadas no sistema.

O *Worker* atua como cliente da API e gerencia contêineres de virtualização leve (e.g., Docker). Após seu registro no sistema, o *Worker* recebe uma chave de segurança usada para garantir a comunicação segura com a API.

O *Data Processor* abstrai a aplicação desejada (e.g., DroidAugmentor), encapsulando-a em uma tecnologia de virtualização leve, como o Docker, configurada para padrões de entrada e saída, além de valores permitidos para cada parâmetro. Essa abordagem proporciona flexibilidade ao sistema: o administrador pode adicionar novos processadores de *datasets* (ou diferentes versões de um mesmo processador) simplesmente especificando uma imagem existente em um repositório,

[9] `https://www.postgresql.org/`
[10] `https://redis.io/`

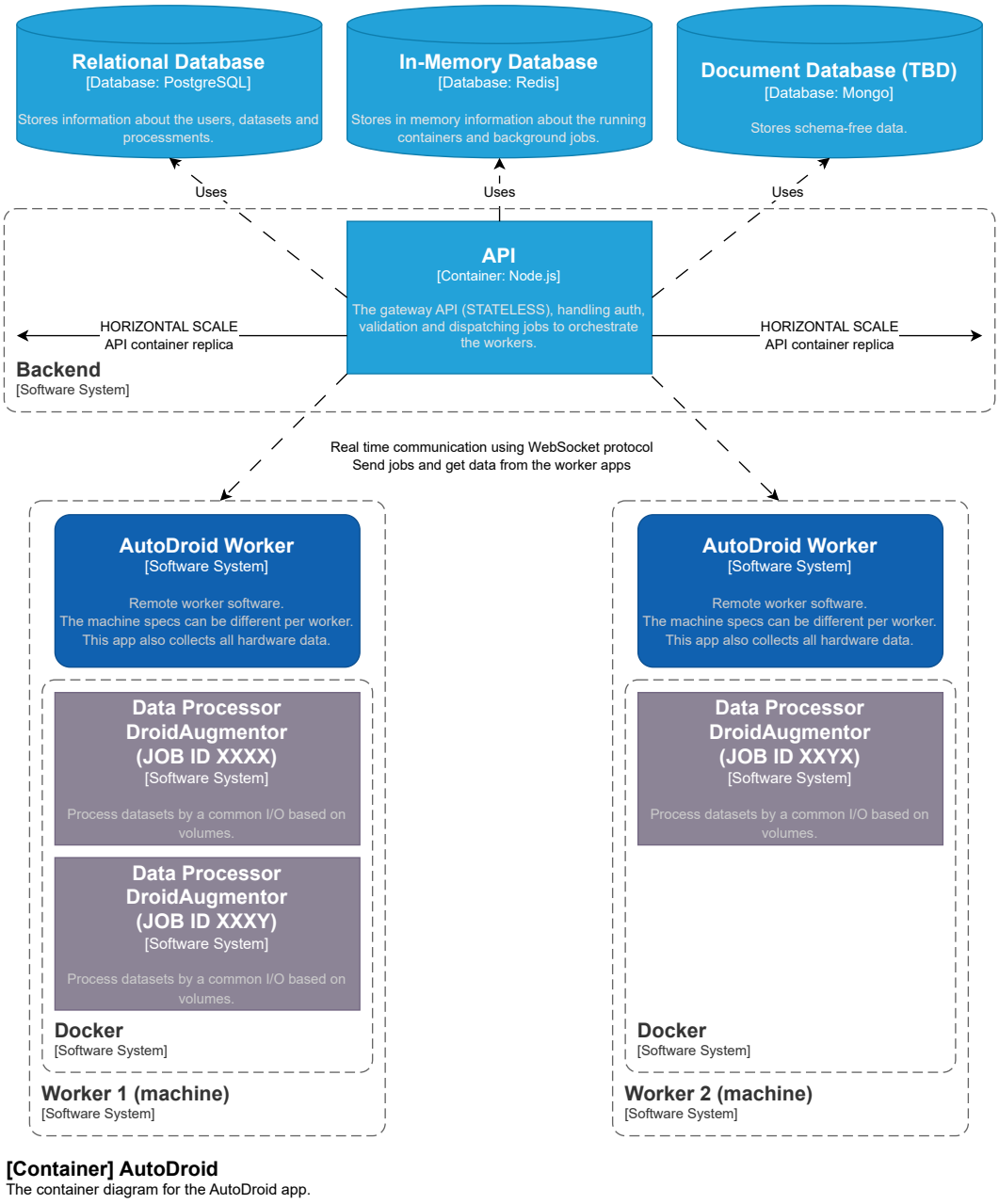

Figura 2. Arquitetura em nível de contêiner.

como o *Docker Hub*[11], no arquivo de configuração, geralmente no formato JSON. Esse arquivo é carregado na inicialização da aplicação, definindo os processadores disponíveis, suas configurações e a imagem correspondente, que é automaticamente carregada durante a inicialização da API.

Os conceitos de *User*, *Dataset* e *Processment* são processados em série através das camadas *Controller*, *Services* e *Repository*. Por exemplo, para o conceito *Proces-*

[11] `https://hub.docker.com/`

sment, existem os componentes *Processment Controller*, *Processment Service* e *Processment Repository*. Essa estrutura proporciona isolamento de responsabilidades, garantindo maior manutenabilidade e flexibilidade à arquitetura, alinhando-se aos princípios de *Domain Driven Design* e aos princípios SOLID (*Single-responsibility Principle, Open-closed Principle, Liskov Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle*).

A Figura 3 ilustra o diagrama de sequência da Cloud AutoDroid para a execução de um processamento. Inicialmente, são realizados testes e configuradas as etapas necessárias para a criação do usuário e do *dataset* de entrada. A chamada para iniciar o processamento é assíncrona, considerando que essa etapa pode variar significativamente em duração, desde minutos até dias. O sistema permite consultas periódicas sobre o status do processamento até sua conclusão. Após a execução, é possível verificar o resultado (sucesso ou falha) e acessar os arquivos de saída, como o *dataset* sintético gerado e gráficos de qualidade produzidos pela MalSynGen.

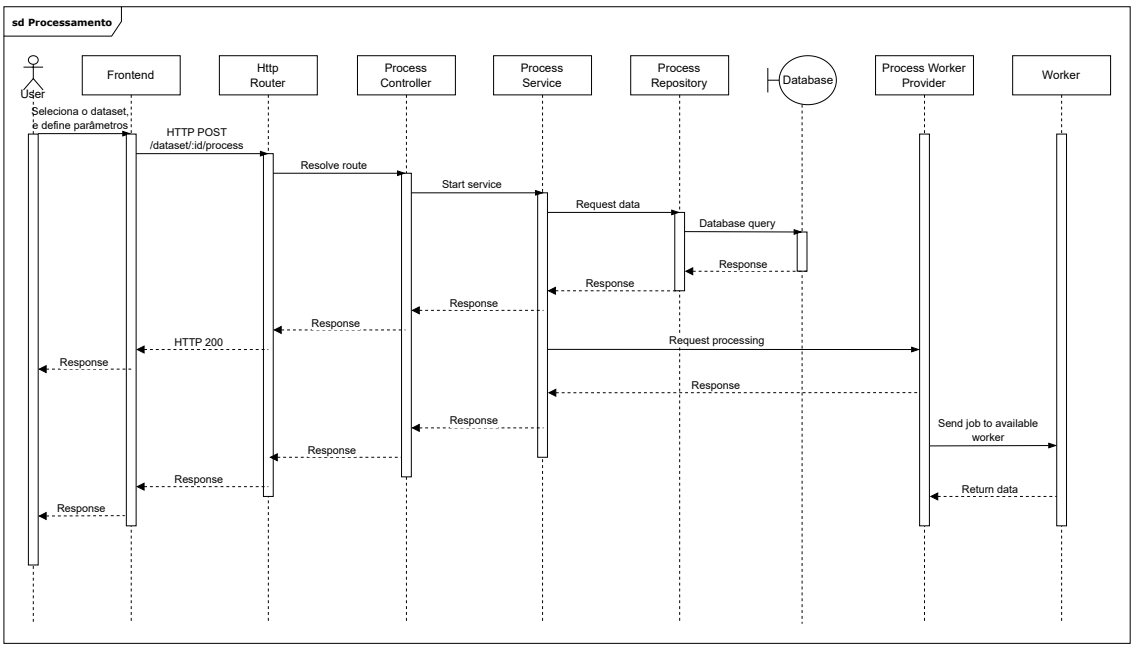

Figura 3. Diagrama de sequência para realização de processamento.

Para delimitar o escopo do projeto, adotamos duas simplificações principais. A primeira está relacionada ao balanceamento de carga entre os *workers*. Assumimos que todos os *Workers* possuem capacidades equivalentes e são capazes de executar qualquer instância do *Data Processor*, com a restrição de que cada *Worker* pode executar, no máximo, uma instância por vez. A política de escalonamento adotada é *round robin*, uma estratégia eficiente e amplamente utilizada em serviços de larga escala na Internet, como o DNS [Hong et al. 2006]. Não implementamos estratégias de tolerância a falhas (e.g., recuperação de nós ou transferência de estado parcial ou total).

A segunda simplificação envolve um esquema básico para o acompanhamento do processamento, com apenas quatro estágios: preparado, executando, pronto e falho. No futuro, pretendemos incorporar controles mais avançados, especialmente

para o treinamento de redes neurais (e.g., acompanhamento de épocas de treinamento) e monitoramento de uso de recursos, visando otimizar a eficiência.

3. Avaliação Preliminar

Nesta seção, apresentamos uma implementação em um ambiente de testes como prova de conceito, resultados de testes funcionais, e um estudo de caso.

3.1. Implementação e Ambiente de Testes

Utilizamos diversas tecnologias na implementação do Cloud AutoDroid, com base em critérios de popularidade, maturidade, suporte pela comunidade e o conhecimento técnico da equipe envolvida no projeto. A aplicação foi desenvolvida em TypeScript, executada no runtime Node.js, e utiliza o *framework* Express.js para lidar com requisições HTTP. O SGBD PostgreSQL é usado para armazenar dados mestres (usuários, *workers*, etc.), enquanto o banco de dados não relacional MongoDB é responsável pela persistência e representação de arquivos e processamentos. Para a execução de tarefas de forma assíncrona, adotamos o banco de dados chave-valor Redis, em conjunto com a biblioteca Bull.js para o gerenciamento de filas.

A implementação do sistema inclui o upload de arquivos por URL assinada, com a aplicação expondo tanto uma API REST, via Express.js[12], quanto uma API GraphQL utilizando Apollo Server[13]. A comunicação bidirecional entre o *backend* e os *workers* (clientes do *backend*) é realizada utilizando Socket.io. A autenticação dos usuários é feita através do Firebase, enquanto a autenticação dos *workers* ocorre inicialmente por meio de um *token* administrativo, que pode ser configurado para uso único ou ilimitado, conforme definido pelo administrador. Após a autenticação inicial, o *worker* realiza a troca do *token* por um *refresh token* e um *access token* JWT (JSON Web Token) [Jones 2015] para comunicação contínua com o *backend*. A biblioteca Dockerode[14] é utilizada para interação com o Docker da máquina hospedeira do *worker*, por meio do arquivo *docker.sock*, estabelecendo a comunicação necessária entre os componentes.

3.2. Testes

Os testes de software têm o objetivo de assegurar que a solução implementada funcione conforme o esperado e continue operando adequadamente durante todo o ciclo de evolução e manutenção do software. A instrumentação dos testes deve ser feita com rigor, escolhendo as ferramentas adequadas e estabelecendo uma base sólida para garantir que os testes possam ser implementados sem restrições tecnológicas.

A estrutura de testes da Cloud AutoDroid foi dividida em três tipos: testes unitários, testes de integração e testes de ponta a ponta[15]. O *framework* de testes escolhido foi o Vitest[16], que foi facilmente configurado e adequado para os três

[12] `https://expressjs.com/`

[13] `https://www.npmjs.com/package/apollo-server`

[14] `https://www.npmjs.com/package/dockerode/v/2.5.5`

[15] No cenário de uma aplicação *backend*, considera-se ponta a ponta a chamada da API na porta especificada, dispensando o teste de interface do usuário, que é de responsabilidade do projeto que consumirá os dados (frontend).

[16] `https://vitest.dev/`

tipos de testes realizados neste projeto. Ele atendeu a todas as necessidades básicas de teste, possibilitando a execução de ambientes isolados e paralelos, garantindo rapidez na execução e uma fácil análise dos resultados por meio do relatório gerado pelo próprio *framework*, juntamente com o Istanbul[17], também disponível no Vitest.

Os testes unitários foram focados em validar o escopo das funções, verificando se todas as ramificações do fluxo de código foram corretamente alcançadas para cada caso. Foram utilizadas estratégias como *Test Doubles* (Mocks, Stubs, Fakes, Spies e Dummies) para simular potenciais cenários reais, garantindo que a aplicação responderá corretamente, evitando interações indesejadas com serviços ou aplicações externas. Os dados apresentados refletem a cobertura de testes do sistema, destacando a eficácia dos testes em diferentes componentes. Atualmente, a cobertura de funções é de 56,34%, representando 302 de 536 funções testadas, selecionadas de modo a minimizar riscos e maximizar o retorno com o esforço de teste.

Os testes de integração foram desenvolvidos para verificar a comunicação entre os componentes da Cloud AutoDroid, incluindo o banco de dados. Para isso, utilizamos *Testcontainers*[18], que fornecem uma conexão real com os serviços de persistência, porém com isolamento e limpeza automática do ambiente, utilizando Docker internamente.

Por fim, os testes de ponta a ponta (E2E) executam uma instância completa do serviço, disponibilizando a interface HTTP para testar a entrada e saída (*I/O*) da aplicação como um todo, de forma isolada. Utilizamos a biblioteca *supertest*[19] para as chamadas internas e asserções, além da já mencionada *Testcontainers*, garantindo o isolamento de cada suíte de teste.

3.3. Estudo de Caso

A Cloud AutoDroid está sendo implantada como solução de *backend* do Malware DataLab, que apresenta requisitos avançados nas áreas de segurança e sistemas distribuídos. Testes iniciais indicam que a arquitetura é adequada para atender a essas demandas específicas. No entanto, identificamos também outras possíveis aplicações para a Cloud AutoDroid, como em fluxos de trabalho de ciência de dados e no provisionamento de IA *as a Service* (IAaaS).

4. Considerações Finais

Neste trabalho, apresentamos a Cloud AutoDroid, uma solução de *backend* para a execução de serviços de inteligência artificial como serviço, utilizando virtualização leve distribuída. Descrevemos uma implementação, realizamos uma série de testes como prova de conceito, e apresentamos um estudo de caso sobre sua aplicação no Malware DataLab. Esperamos que a Cloud AutoDroid contribua positivamente para a investigação e disponibilização de *datasets* de *malware*, facilitando a execução de processos de maneira distribuída, proporcionando escalabilidade, elasticidade e tolerância a falhas. Além disso, planejamos explorar o potencial da Cloud AutoDroid em outros contextos de processamento distribuído de serviços de IA.

[17] `https://istanbul.js.org/`
[18] `https://testcontainers.com/`
[19] `https://www.npmjs.com/package/supertest`

Identificamos oportunidades para pesquisa e desenvolvimento futuros. Por exemplo, pretendemos ampliar a cobertura dos testes de *functions*, além de avaliar quantitativamente os ganhos proporcionados pela solução em comparação com alternativas de execução existentes. Também esperamos analisar o desempenho da Cloud AutoDroid em escala nacional, no uso com o Malware DataLab, bem como a usabilidade da API por equipes de outros projetos envolvendo soluções de IA. Os resultados dessas avaliações devem motivar a consideração de novos requisitos, como estratégias mais sofisticadas de balanceamento de carga e métodos de acompanhamento em grão fino da execução.

Agradecimentos. A pesquisa contou com apoio parcial da RNP (Programa Hackers do Bem - GT Malware DataLab), da CAPES (Código de Financiamento 001) e da FAPERGS, por meio dos editais 02/2022 (processo 22/2551-0000841-0), 08/2023 e 09/2023 e do termo de outorga 24/2551-0001368-7.

Referências

AI & Data Today (2023). Top 10 reasons why ai projects fail. `https://www.aidatatoday.com/top-10-reasons-why-ai-projects-fail`.

Casola, K., Paim, K., Mansilha, R., and Kreutz, D. (2023). DroidAugmentor: uma ferramenta de treinamento e avaliação de cGANs para geração de dados sintéticos.

Hong, Y. S., No, J., and Kim, S. (2006). DNS-based load balancing in distributed web-server systems. In *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and the Second International Workshop on Collaborative Computing, Integration, and Assurance (SEUS-WCCIA'06)*, pages 4–pp. IEEE.

Jones, M. (2015). Json web token (jwt). *Internet Engineering Task Force (IETF) RFC*, 7519.

Kouliaridis, V., Kambourakis, G., and Peng, T. (2020). Feature importance in android malware detection. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 1449–1454. IEEE.

Laviola, L., Paim, K., Kreutz, D., and Mansilha, R. (2023). AutoDroid: disponibilizando a ferramenta DroidAugmentor como serviço. In *Anais da XX Escola Regional de Redes de Computadores*, pages 145–150, Porto Alegre, RS, Brasil. SBC.

Meijin, L., Zhiyang, F., Junfeng, W., Luyu, C., Qi, Z., Tao, Y., Yinwei, W., and Jiaxuan, G. (2022). A systematic overview of android malware detection. *Applied Artificial Intelligence*, 36(1):2007327.

Miranda, T. C., Gimenez, P.-F., Lalande, J.-F., Tong, V. V. T., and Wilke, P. (2022). Debiasing android malware datasets: How can i trust your results if your dataset is biased? *IEEE Transactions on Information Forensics and Security*, 17:2182–2197.

Nogueira, A., Paim, K., Bragança, H., Mansilha, R., and Kreutz, D. (2024a). Geração de dados sintéticos tabulares para detecção de malware android: um estudo

de caso. In *Anais do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 808–814, Porto Alegre, RS, Brasil. SBC.

Nogueira, A., Paim, K., Bragança, H., Mansilha, R., and Kreutz, D. (2024b). Malsyngen: redes neurais artificiais na geração de dados tabulares sintéticos para detecção de malware. In *Anais Estendidos do XXIV Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 129–136, Porto Alegre, RS, Brasil. SBC.

Wang, H., Si, J., Li, H., and Guo, Y. (2019). RmvDroid: Towards a reliable android malware dataset with app metadata. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 404–408.