

PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA
NÚCLEO DE COMPUTAÇÃO



**SEGURANÇA EM COMPUTAÇÃO *SERVERLESS*: ESTUDO DE CASO NO
PROJETO *PENSE BEM***

CAROLINE PEREIRA DE SOUZA

Manaus – AM

2026

CAROLINE PEREIRA DE SOUZA

**SEGURANÇA EM COMPUTAÇÃO *SERVERLESS*: ESTUDO DE CASO NO
PROJETO *PENSE BEM***

Trabalho de Conclusão de Curso apresentado
à banca avaliadora do Curso de Engenharia
de Computação, da Escola Superior de
Tecnologia, da Universidade do Estado do
Amazonas, como pré-requisito para obtenção
do título de Engenheiro de Computação.

Orientador(a): Prof. Dr. Raimundo Corrêa de Oliveira

Manaus – AM

2026

Ficha Catalográfica

Ficha catalográfica elaborada automaticamente de acordo com os dados fornecidos pelo(a) autor(a).
Sistema Integrado de Bibliotecas da Universidade do Estado do Amazonas.

S719s	Souza, Caroline Pereira de Segurança em computação serverless: Estudo de caso no projeto Pense Bem / Caroline Pereira de Souza. Manaus : [s.n], 2026. 94 f.: color.; 21.0 cm. TCC - Graduação em Engenharia de Computação- Universidade do Estado do Amazonas, Manaus, 2026. Inclui Bibliografia. Inclui Apêndice. Orientador: Oliveira, Raimundo Corrêa de. 1. Computação Serverless. 2. FaaS. 3. Segurança em Nuvem. 4. Análise de Vulnerabilidades. 5. AWS. I. Oliveira, Raimundo Corrêa de (Orient.) II. Universidade do Estado do Amazonas. III. Título CDU(1997)62:004
-------	---

Caroline Pereira de Souza

Segurança em Computação *Serverless*: Estudo de Caso no Projeto *Pense Bem*

Trabalho de Conclusão de Curso apresentado à banca avaliadora do Curso de Engenharia de Computação, da Escola Superior de Tecnologia, da Universidade do Estado do Amazonas, como pré-requisito para obtenção do título de Engenharia de Computação.

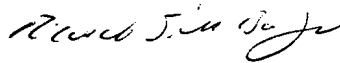
Trabalho aprovado. Manaus, 15 de junho de 2026, pela seguinte banca examinadora:



Prof. Dr. Raimundo Corrêa de Oliveira

Universidade do Estado do Amazonas (UEA)

Orientador



Prof. Dr. Ricardo da Silva Barboza

Universidade do Estado do Amazonas (UEA)



Esp. Paulo Vitor Dácio Teixeira

Universidade do Estado do Amazonas (UEA)

Manaus – AM

2026

Dedicatória

Dedico este trabalho ao Prof. Raimundo, cuja orientação encorajadora guiou meus passos com sabedoria e humanidade. Sua confiança no meu potencial foi essencial para que eu chegasse até aqui.

Dedico também aos meus colegas de classe, que tornaram a jornada mais leve, dividiram desafios, celebraram conquistas e ofereceram apoio nos momentos mais exigentes.

Dedico, sobretudo, aos meus pais, Cléa e Marcos, que fizeram das próprias batalhas a estrada por onde pude caminhar. À eles, que me ofereceram estudo, coragem e afeto; que sustentaram meus passos quando o mundo parecia grande demais; e que permaneceram ao meu lado em cada conquista.

A vocês, minha imensa gratidão.

Resumo

Este trabalho apresenta uma análise de segurança aplicada ao modelo *serverless* em aplicações em nuvem, utilizando o *back-end* da plataforma educacional *Pense Bem* como estudo de caso. O objetivo foi identificar vulnerabilidades associadas ao modelo, analisar sua manifestação na arquitetura da aplicação e validar experimentalmente riscos relacionados ao isolamento de funções, ao controle de permissões, ao monitoramento, à auditoria e à rastreabilidade distribuída. A metodologia envolveu revisão bibliográfica, análise de relatórios técnicos, mapeamento da arquitetura do sistema, identificação de ativos críticos e execução de testes experimentais em rotas e serviços selecionados. Os resultados demonstraram a reutilização de ambientes Lambda entre invocações, a persistência de arquivos no diretório temporário `/tmp`, a recorrência de *warm starts* sob carga, a existência de permissões IAM excessivas e limitações de observabilidade, como ausência de logs de acesso configurados, falta de identificadores de correlação, inexistência de auditoria persistente no DynamoDB e baixa padronização dos logs. Com isso, o estudo evidencia que a segurança em arquiteturas *serverless* depende não apenas dos recursos oferecidos pelo provedor de nuvem, mas também de decisões adequadas de implementação, configuração e monitoramento.

Palavras-chave: Computação Serverless; FaaS; Segurança em Nuvem; Análise de Vulnerabilidades; AWS.

Abstract

This work presents a security analysis applied to the *serverless* model in cloud-based applications, using the *back-end* of the educational platform *Pense Bem* as a case study. The objective was to identify vulnerabilities associated with this model, analyze their manifestation in the application architecture, and experimentally validate risks related to function isolation, permission control, monitoring, auditing, and distributed traceability. The methodology involved a literature review, analysis of technical reports, mapping of the system architecture, identification of critical assets, and execution of experimental tests on selected routes and services. The results demonstrated the reuse of Lambda execution environments across invocations, persistence of files in the temporary `/tmp` directory, recurrence of *warm starts* under load, excessive IAM permissions, and observability limitations, such as the absence of configured access logs, lack of correlation identifiers, absence of persistent auditing in DynamoDB, and low log standardization. Thus, the study shows that security in *serverless* architectures depends not only on the resources provided by the cloud provider, but also on appropriate implementation, configuration, and monitoring decisions.

Keywords: Serverless Computing; FaaS; Cloud Security; Vulnerability Analysis; AWS.

Lista de Figuras

2.1	Representação genérica de uma arquitetura serverless baseada em funções FaaS, mostrando o fluxo entre clientes, gateway de API, funções e serviços auxiliares. .	13
4.1	Tela inicial da plataforma <i>Pense Bem</i> , que apresenta uma experiência totalmente gamificada ao usuário.	19
4.2	Localização das ameaças e desafios de segurança no framework serverless.	23
4.3	Distribuição de identidades em ambientes de nuvem.	30
4.4	Classificação de higiene de vulnerabilidades IAM.	30
4.5	Organizações que realizam avaliação de risco financeiro relacionado a FaaS.	31
4.6	Arquitetura geral do <i>Pense Bem</i>	33
4.7	Modelo de classes do sistema <i>Pense Bem</i>	34
5.1	Distribuição de <i>warm</i> e <i>cold starts</i> por contêiner durante o Teste A1.	51
5.2	Evolução da quantidade de arquivos pré-existent no diretório <code>/tmp</code> durante o Teste A2.	54
5.3	Resultados do stress test com K6 para avaliação de <i>warm starts</i> e reuso de contêiner no Teste A3.	56
5.4	Resultado da simulação de permissões IAM para a role <code>games-api-role</code>	58
5.5	Avaliação de <i>access logs</i> e tracing distribuído no API Gateway durante o Teste B1.	61
5.6	Avaliação de <i>Correlation IDs</i> e rastreamento distribuído no Teste B2.	63
5.7	Cobertura de auditoria do DynamoDB avaliada durante o Teste B3.	65

5.8	Avaliação da qualidade e estruturação dos logs das funções Lambda durante o	
	Teste B4.	67

Lista de Quadros

4.1	Ativos críticos do backend do <i>Pense Bem</i> , serviços associados e principais ameaças.	37
4.2	Matriz de riscos para as vulnerabilidades analisadas.	43
5.1	Componentes considerados no ambiente experimental.	47
5.2	Rotas selecionadas para os testes experimentais.	48
5.3	Ferramentas utilizadas na validação experimental.	48
5.4	Resultado do Teste A1: persistência de variáveis entre execuções.	51
5.5	Resultado do Teste A2: persistência de arquivos em <code>/tmp</code>	54
5.6	Configuração do teste de stress com K6 no Teste A3.	55
5.7	Resultado do Teste A3: stress de <i>warm start</i> com K6.	57
5.8	Resultado do Teste A4: permissões avaliadas na role <code>games-api-role</code>	59
5.9	Resultado do Teste B1: auditoria no API Gateway.	62
5.10	Resultado do Teste B2: rastreamento distribuído entre serviços.	63
5.11	Resultado do Teste B3: auditoria de operações no DynamoDB.	65
5.12	Resultado do Teste B4: qualidade dos logs das funções Lambda.	67
5.13	Matriz de testes por vulnerabilidade.	68
5.14	Comparação entre achados experimentais e configuração segura de referência. . .	72

Sumário

1	Introdução	1
1.1	Contextualização	1
1.2	Justificativa	2
1.3	Objetivo Geral	4
1.4	Objetivos Específicos	4
1.5	Organização do Trabalho	5
2	Fundamentação Teórica	7
2.1	Computação em Nuvem	7
2.1.1	Modelos de Serviço: IaaS, PaaS e SaaS	8
2.1.2	Serverless e o Modelo FaaS	8
2.2	Modelo Function-as-a-Service (FaaS)	9
2.2.1	Benefícios Operacionais do FaaS	9
2.2.2	Desafios do FaaS	10
2.3	Arquitetura e Modelos de Execução Serverless	11
2.3.1	Modelo de execução: contêineres, cold start e warm start	11
2.3.2	Estrutura e fluxo de execução	12
2.4	Síntese crítica da revisão	14
3	Metodologia	15
3.1	Definição da Metodologia	15
3.2	Etapas Metodológicas	16

4	Análise de Segurança do <i>Pense Bem</i>	18
4.1	Contextualização da Análise	18
4.2	O <i>Pense Bem</i>	19
4.3	Escopo da Análise	20
4.4	Panorama de Segurança em <i>Serverless</i>	21
4.4.1	Desafios estruturais de segurança	21
4.4.2	Ataques de canal lateral, condições de corrida e vulnerabilidades emergentes	24
4.4.3	Ataques de negação de serviço e negação de carteira	25
4.4.4	Implicações para o sistema <i>Pense Bem</i>	25
4.5	Normas e Boas Práticas de Segurança	26
4.5.1	Normas internacionais aplicáveis à computação em nuvem	26
4.5.2	Boas práticas recomendadas para ambientes <i>serverless</i>	27
4.5.3	Modelos complementares de referência	28
4.6	Cenário Atual e Estatísticas Relevantes	29
4.7	Arquitetura do <i>Pense Bem</i>	32
4.7.1	Componentes Principais do Back-end	33
4.7.2	Fluxo de Dados e Execução	35
4.8	Identificação de Ativos, Ameaças e Vulnerabilidades	35
4.8.1	Ativos Críticos Identificados	35
4.8.2	Seleção das Vulnerabilidades Analisadas	37
4.8.3	Ameaças Associadas	39
4.9	Análise dos Pontos de Exposição do Ambiente	40
4.9.1	Critérios de Risco Utilizados	41
4.9.2	Avaliação da Vulnerabilidade A — Falhas de Isolamento	41
4.9.3	Avaliação da Vulnerabilidade B — Falhas de Monitoramento e Rastrea- bilidade Distribuída	42
4.9.4	Matriz de Riscos	42

5	Validação Experimental das Vulnerabilidades	44
5.1	Contextualização da Validação Experimental	44
5.2	Objetivos dos Testes	45
5.3	Ambiente Experimental	46
5.4	Ferramentas Utilizadas	48
5.5	Testes Relacionados à Vulnerabilidade A: Falhas de Isolamento	49
5.5.1	Teste A1: Persistência de Variáveis entre Execuções	49
5.5.2	Teste A2: Persistência de Arquivos no Diretório Temporário <code>/tmp</code>	52
5.5.3	Teste A3: Stress de <i>Warm Start</i> com K6	55
5.5.4	Teste A4: Avaliação de Controle de Acesso e Permissões Excessivas na Role <code>games-api-role</code>	57
5.6	Testes Relacionados à Vulnerabilidade B: Falhas de Monitoramento, Auditoria e Rastreabilidade Distribuída	60
5.6.1	Teste B1: Ausência de Access Logs e Tracing no API Gateway	60
5.6.2	Teste B2: Ausência de Correlation IDs entre Serviços	62
5.6.3	Teste B3: Cobertura de Auditoria no DynamoDB	64
5.6.4	Teste B4: Qualidade e Estruturação dos Logs	66
5.7	Síntese dos Testes Executados	68
5.8	Discussão dos Resultados Experimentais	69
5.9	Contramedidas Indicativas	71
6	Resultados e Considerações Finais	75
6.1	Síntese dos Resultados Obtidos	75
6.2	Limitações do Estudo	77
6.3	Trabalhos Futuros	78
6.4	Considerações Finais	79

Capítulo 1

Introdução

Este capítulo introdutório tem como finalidade contextualizar o tema, apresentar o problema de pesquisa e situar os objetivos que norteiam o desenvolvimento deste trabalho.

1.1 Contextualização

A computação em nuvem pode ser compreendida como um ambiente formado por uma ampla rede de servidores, físicos ou virtuais, que oferece, pela Internet, um conjunto de recursos como processamento, armazenamento, conectividade, plataformas, aplicações e serviços. Conforme interpretado de Taurion (TAURION, 2009), a nuvem representa a evolução do conceito de virtualização, deixando de abranger apenas máquinas isoladas para virtualizar todo o data center, abstraindo sua complexidade e disponibilizando seus recursos de forma dinâmica, elástica e sob demanda.

Nesse cenário, a computação *serverless* emergiu como uma das abordagens mais relevantes da nuvem. Diferentemente dos modelos tradicionais — que serão discutidas mais adiante — o *serverless* permite que funções sejam executadas apenas quando requisitadas, com escalabilidade automática, cobrança baseada em uso e abstração completa da gestão de servidores. Essa forma de execução é viabilizada principalmente pelo *Function-as-a-Service* (FaaS), no qual pequenas unidades de código são acionadas de maneira efêmera, isolada e orientada a eventos.

Embora esse modelo traga benefícios significativos — como redução de custos operacio-

nais, simplificação do desenvolvimento e maior resiliência — ele também introduz desafios de segurança específicos, decorrentes de características como efemeridade das execuções, compartilhamento de recursos, fragmentação do fluxo de controle e forte dependência de permissões e serviços gerenciados.

Pesquisas recentes têm demonstrado que, apesar de seus benefícios operacionais, o modelo *serverless* introduz desafios de segurança que diferem substancialmente daqueles observados em modelos tradicionais. Trabalhos como os de Li, Leng e Chen (LI; LENG; CHEN, 2021) e Marin, Perino e Di Pietro (MARIN; PERINO; PIETRO, 2021) destacam que características estruturais do *serverless* — como execução efêmera, acionamento por múltiplos tipos de evento, fragmentação do fluxo de controle e dependência intensiva de serviços gerenciados — ampliam a superfície de ataque e dificultam mecanismos clássicos de auditoria, isolamento e gestão de permissões. Esses estudos evidenciam que, embora os provedores de nuvem ofereçam mecanismos de proteção avançados, ainda subsistem lacunas relacionadas à observabilidade distribuída, ao isolamento entre funções e à configuração adequada de identidades e permissões, tornando indispensável uma análise criteriosa dos riscos específicos desse modelo.

Diante desse cenário, torna-se essencial compreender de forma sistemática como essas vulnerabilidades se manifestam na prática e quais medidas podem ser adotadas para mitigá-las. Assim, o presente trabalho propõe uma análise aplicada de segurança em ambientes *serverless*, tendo como estudo de caso o sistema educacional *Pense Bem*, cuja arquitetura é baseada em APIs com funções implementadas no modelo FaaS. Ao investigar a superfície de ataque, identificar vulnerabilidades e propor contramedidas, busca-se contribuir para práticas mais seguras de desenvolvimento e operação em computação em nuvem, bem como para o avanço acadêmico na área de segurança em ambientes sem servidor.

1.2 Justificativa

A crescente migração de sistemas para soluções em nuvem tem ampliado o uso de arquiteturas que adotam execução orientada a eventos, caso das aplicações desenvolvidas no modelo *serverless*. Entretanto, à medida que esse tipo de solução se torna mais comum em ambientes

produtivos, também se intensifica o surgimento de incidentes relacionados a falhas de configuração, permissões mal definidas e dependência excessiva de serviços gerenciados. Esses fatores têm evidenciado limitações nos mecanismos tradicionais de segurança, que não foram concebidos para lidar com execuções efêmeras, fluxos distribuídos entre múltiplos serviços e ausência de visibilidade centralizada — características inerentes ao *serverless*.

Estudos recentes apontam que essas limitações se manifestam predominantemente em quatro áreas: isolamento entre funções, monitoramento distribuído, gestão granular de permissões e proteção de dados (LI; LENG; CHEN, 2021). A combinação desses elementos, somada ao uso crescente de identidades automatizadas, torna o modelo mais suscetível a problemas de configuração e a ataques que exploram acessos indevidos ou comportamentos inesperados. Esse cenário é agravado por evidências empíricas: análises da Sysdig indicam que cerca de 90% das permissões concedidas em ambientes de nuvem não são utilizadas, revelando um quadro recorrente de privilégios excessivos, enquanto identidades não humanas já representam 58% das credenciais ativas, ampliando a complexidade de controle em aplicações distribuídas (Sysdig, 2023).

A relevância dessa problemática também se expressa no impacto financeiro associado a falhas de segurança. De acordo com o relatório *Cost of a Data Breach 2025*, da IBM, o custo médio de uma violação de dados no Brasil atingiu R\$ 7,19 milhões, sendo o *phishing*, o comprometimento de terceiros e a exploração de vulnerabilidades as causas mais frequentes (IBM Brasil, 2025). Esses vetores estão diretamente relacionados a configurações inadequadas, permissões amplas e falta de monitoramento — fragilidades típicas de arquiteturas distribuídas.

Além dos riscos operacionais, ambientes *serverless* também estão expostos a ameaças econômicas emergentes, como os ataques do tipo *Denial of Wallet* (DoW). Dorsett et al. (DORSETT et al., 2025) demonstram que cargas de trabalho FaaS podem ser manipuladas para gerar elevações abruptas de custo, chegando a milhares de dólares por dia mesmo sem violação de dados. Em um dos casos analisados, requisições rejeitadas foram suficientes para inflar a fatura da AWS em aproximadamente USD 1.300 diários, evidenciando a fragilidade financeira do modelo. Apesar disso, o estudo indica que 64% das organizações que utilizam o *serverless* não

realizam avaliação de risco financeiro, e apenas 13,6% dos profissionais afirmam conhecer esse tipo de ataque, o que demonstra uma lacuna significativa de conscientização.

Considerados em conjunto, esses dados ressaltam que a segurança em ambientes *serverless* envolve desafios que extrapolam a camada técnica, exigindo atenção sobre governança, monitoramento, permissões e impacto financeiro. Dessa forma, justifica-se a necessidade de estudos aplicados que analisem como tais vulnerabilidades se manifestam na prática e quais medidas podem ser adotadas para mitigá-las de maneira eficaz, contribuindo para o desenvolvimento de aplicações mais seguras e resilientes em computação em nuvem.

1.3 Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma análise de segurança de aplicações baseadas em computação *serverless*, tomando como estudo de caso o sistema *Pense Bem*. Busca-se compreender como vulnerabilidades estruturais desse modelo se manifestam na prática, validar experimentalmente os riscos identificados e analisar seus impactos no contexto do sistema avaliado.

1.4 Objetivos Específicos

Para alcançar o objetivo geral, estabelecem-se os seguintes objetivos específicos:

- **Consolidar um panorama teórico sobre segurança em ambientes *serverless***, reunindo e organizando resultados de estudos, normas e relatórios técnicos que abordam vulnerabilidades, classes de ataques e desafios estruturais desse modelo.
- **Mapear e analisar a implementação *serverless* do sistema *Pense Bem***, identificando seus componentes, fluxos de dados, integrações com serviços gerenciados, mecanismos de autenticação, identidades IAM e demais elementos relevantes para a superfície de ataque.
- **Identificar, classificar e contextualizar vulnerabilidades** presentes na implementação do sistema, relacionando-as aos desafios estruturais apresentados na literatura, como falhas

de isolamento, monitoramento fragmentado, exposição de eventos, permissões excessivas e problemas decorrentes da execução distribuída.

- **Avaliar os riscos associados às vulnerabilidades mapeadas**, considerando sua probabilidade de ocorrência, os pontos de exposição da arquitetura, as evidências obtidas nos testes experimentais e sua relevância no contexto específico do *Pense Bem*.
- **Executar testes experimentais direcionados às vulnerabilidades priorizadas**, coletando evidências sobre reutilização de ambientes Lambda, persistência de dados temporários, permissões IAM, cobertura de auditoria, rastreabilidade e qualidade dos logs.
- **Discutir os resultados obtidos** a partir das evidências coletadas, relacionando os achados experimentais aos riscos de segurança observados na arquitetura analisada e às boas práticas aplicáveis a ambientes *serverless*.

A execução dos testes respeitou restrições éticas e operacionais previamente delimitadas. Todas as verificações foram realizadas em ambiente autorizado, com foco na coleta de evidências técnicas e sem exploração ofensiva, exfiltração de dados, alteração destrutiva de recursos ou tentativa de indisponibilizar o sistema. Os testes foram conduzidos de forma controlada, utilizando rotas e serviços selecionados, com o objetivo de validar condições de risco associadas ao *serverless*, e não comprometer a aplicação ou seus usuários.

1.5 Organização do Trabalho

Além deste capítulo introdutório, o presente trabalho está estruturado em mais cinco capítulos, organizados da seguinte forma:

- **Capítulo 2 – Fundamentação Teórica:** apresenta os conceitos essenciais relacionados à computação em nuvem e ao *serverless computing*, incluindo definições, características gerais, modelos de serviço, funcionamento do modelo *serverless* e desafios de segurança associados a esse paradigma. O capítulo fornece a base conceitual necessária para a compreensão dos elementos técnicos analisados posteriormente.

- **Capítulo 3 – Metodologia:** descreve o delineamento metodológico adotado na pesquisa, sua natureza e as etapas que estruturaram o desenvolvimento do trabalho. São apresentadas as atividades de revisão bibliográfica, análise documental, estudo de caso, mapeamento da arquitetura, identificação de vulnerabilidades e definição dos testes experimentais.
- **Capítulo 4 – Análise de Segurança do Sistema *Pense Bem*:** apresenta o estudo de caso, com a descrição do sistema *Pense Bem*, o escopo da análise, o mapeamento da arquitetura do *back-end*, a identificação dos ativos críticos, o levantamento das ameaças e a seleção das vulnerabilidades priorizadas. O capítulo também discute os principais pontos de exposição do ambiente analisado.
- **Capítulo 5 – Validação Experimental das Vulnerabilidades:** descreve os testes executados para verificar, na prática, as vulnerabilidades priorizadas no capítulo anterior. São apresentados os experimentos relacionados a falhas de isolamento, persistência de dados temporários, reutilização de ambientes Lambda, permissões IAM, ausência de *access logs*, falhas de correlação entre serviços, auditoria no DynamoDB e qualidade dos logs.
- **Capítulo 6 – Resultados e Considerações Finais:** apresenta a síntese dos resultados obtidos, as limitações do estudo, os direcionamentos para trabalhos futuros e as considerações finais. O capítulo consolida as evidências observadas ao longo da análise e discute sua relevância para a segurança de aplicações baseadas em *serverless computing*.

Além dos capítulos mencionados, o trabalho inclui as referências bibliográficas utilizadas e os materiais complementares necessários para apoiar a análise realizada, como diagramas, descrições técnicas, tabelas e evidências dos testes experimentais.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta os conceitos fundamentais que embasam o presente trabalho. São descritos os princípios da computação em nuvem e do modelo serverless, com ênfase em suas vantagens, limitações e desafios de segurança. Também são relacionadas as principais ameaças, vulnerabilidades e práticas recomendadas documentadas na literatura e em normas internacionais. Os conceitos aqui discutidos fornecem o referencial teórico necessário para o estudo de caso do sistema. Pense Bem nos capítulos seguintes.

2.1 Computação em Nuvem

A computação em nuvem (*cloud computing*) constitui um modelo de computação que transforma a forma como recursos tecnológicos são disponibilizados e consumidos. Conforme descrito por Taurion (TAURION, 2009), trata-se de um ambiente sustentado por uma ampla rede de servidores físicos ou virtuais que disponibilizam, pela Internet, recursos como processamento, armazenamento, conectividade, plataformas e aplicações. Esse modelo representa a evolução do conceito de virtualização, expandindo-o para a virtualização completa do data center. Entre suas características fundamentais destacam-se a elasticidade dos recursos, o acesso sob demanda, a eliminação do provisionamento antecipado e o modelo de cobrança baseado no uso (*pay-per-use*). Tais propriedades tornam a computação em nuvem um elemento estratégico para organizações modernas, permitindo maior flexibilidade, eficiência e capacidade de adaptação.

2.1.1 Modelos de Serviço: IaaS, PaaS e SaaS

A evolução da computação em nuvem permitiu o surgimento de diferentes modelos de fornecimento de serviços, cada um oferecendo níveis distintos de abstração da infraestrutura. Classicamente, esses modelos são organizados em *Infrastructure-as-a-Service* (IaaS), *Platform-as-a-Service* (PaaS) e *Software-as-a-Service* (SaaS). De acordo com o Google Cloud (GOOGLE CLOUD, 2025), esses modelos são complementares e podem ser combinados conforme as necessidades específicas de desenvolvimento, implantação e operação de sistemas.

No modelo IaaS, o provedor fornece recursos de infraestrutura virtualizada — como máquinas virtuais, redes e armazenamento — enquanto o usuário é responsável pelos sistemas operacionais e demais camadas de software. O PaaS amplia esse nível de abstração ao oferecer um ambiente de desenvolvimento e execução gerenciado, permitindo que equipes se concentrem no código sem lidar com a configuração de sistemas operacionais ou middleware. Já o SaaS disponibiliza aplicações completas acessíveis diretamente via navegador ou APIs, eliminando a necessidade de instalação e manutenção local.

O avanço progressivo desses modelos ampliou o nível de automação e abstração oferecido pelos provedores de nuvem, abrindo caminho para abordagens ainda mais simplificadas de execução computacional. É nesse contexto que surge a computação sem servidor (*serverless*), discutida na subseção seguinte, que representa um salto evolutivo ao permitir que aplicações sejam executadas sem que o usuário gerencie qualquer infraestrutura subjacente.

2.1.2 Serverless e o Modelo FaaS

A computação sem servidor (*serverless*) representa um modelo avançado de entrega de serviços em nuvem no qual o provedor assume integralmente as tarefas relacionadas ao gerenciamento da infraestrutura, como alocação de recursos, implantação de serviços, escalonamento automático e monitoramento operacional. Segundo Li, Leng e Chen (LI; LENG; CHEN, 2021), esse modelo caracteriza-se por eliminar a necessidade de provisionamento e manutenção de servidores por parte do usuário, permitindo que aplicações sejam cobradas apenas pelo tempo efetivo de execução. Essa combinação de eficiência operacional e redução de custos contribuiu para a rápida

expansão do *serverless*, que tem atraído crescente interesse da indústria e da academia.

Inserido nesse contexto, o modelo *Function-as-a-Service* (FaaS) assume papel central na computação sem servidor. De acordo com a IBM (IBM, 2025), o FaaS constitui um subconjunto do *serverless* no qual unidades independentes de código são executadas exclusivamente em resposta a eventos, sem a manutenção de servidores persistentes. Esse mecanismo altera profundamente a superfície de ataque tradicional, pois a execução fragmentada, o isolamento granular entre funções e a necessidade de correlacionar eventos distribuídos impactam diretamente os mecanismos de segurança, monitoramento e rastreabilidade. Assim, compreender o comportamento e as implicações do modelo FaaS torna-se essencial para analisar arquiteturas baseadas em *serverless*, especialmente quando tais ambientes processam dados sensíveis ou dependem de múltiplos serviços gerenciados.

2.2 Modelo Function-as-a-Service (FaaS)

No centro de ambientes *serverless* encontra-se o modelo Function-as-a-Service (FaaS), o qual está concentrado no paradigma de computação baseada em eventos, onde o código da aplicação ou os containers são executados somente em resposta a eventos ou solicitações (IBM, 2025). No sistema Pense Bem, funções AWS Lambda, que fazem parte desse modelo, desempenham papel central na execução de tarefas específicas, o que torna essencial compreender como sua natureza efêmera, distribuída e acionada por eventos influencia diretamente o isolamento das execuções, o monitoramento contínuo e a proteção de dados em um ambiente dinâmico.

2.2.1 Benefícios Operacionais do FaaS

Além de introduzir um novo modelo operacional, o FaaS oferece vantagens práticas que aumentam sua adoção no desenvolvimento de aplicações em nuvem. Entre seus principais benefícios estão a capacidade de concentrar o esforço de desenvolvimento exclusivamente no código, sem necessidade de gerenciar servidores, bem como o modelo de cobrança baseado apenas no tempo efetivo de execução das funções, o que reduz custos em cargas de trabalho dinâmicas ou inter-

mitentes.

Suas funções também podem escalar automaticamente e de forma independente, acompanhando variações na demanda, enquanto se apoiam na infraestrutura altamente disponível da nuvem. Em termos de aplicação, o FaaS é adequado para processamento de dados, criação de back-ends orientados a eventos e tarefas paralelizáveis (IBM, 2025).

2.2.2 Desafios do FaaS

Apesar das vantagens do modelo citado, seu funcionamento distribuído, efêmero e altamente escalável introduz um conjunto de desafios específicos de segurança. Conforme apontado por Li et al. (LI; LENG; CHEN, 2021), as vulnerabilidades em ambientes *serverless* podem ser organizadas em quatro eixos principais: isolamento de recursos, monitoramento, gerenciamento de segurança e proteção de dados.

O primeiro desafio ocorre devido ao isolamento fraco entre funções *serverless* não ser tão forte quanto o de máquinas virtuais. Isso ocorre porque elas executam em contêineres que compartilham o mesmo *kernel* — o núcleo do sistema operacional responsável por gerenciar memória, processos e comunicação com o hardware. Neste contexto, o isolamento reduzido possibilita abertura para ataques de canal lateral (*side-channel attacks*). Esses ataques, que serão aprofundados em outra seção, exploram características do ambiente para adquirir informações sobre outras funções executadas no mesmo host físico. Assim, uma função maliciosa pode, em certas condições, observar ou deduzir o comportamento de outras pertencentes a diferentes usuários.

Outro importante desafio destacado por Li et al. (LI; LENG; CHEN, 2021) refere-se às limitações de monitoramento nestes ambientes. Por conta do ciclo de vida curto, distribuído e acionados de forma independente dessas funções, torna-se complexo rastrear o fluxo completo de execução e correlacionar eventos para a identificação de anomalias. A ausência de instâncias persistentes impede o uso de ferramentas tradicionais de auditoria e dificulta a detecção de comportamentos suspeitos, especialmente quando há interação com múltiplos serviços ou componentes externos.

No contexto do Pense Bem, essa limitação se manifesta no fluxo de dados entre o gateway de API, as funções e os serviços complementares, onde cada interação gera pontos de execução que exigem rastreamento cuidadoso. Essa característica reforça a necessidade de compreender, no tópico seguinte, como a própria arquitetura sem servidor, com seus mecanismos de execução, isolamento, compartilhamento de recursos e comunicação entre funções, influencia diretamente problemas práticos de isolamento, monitoramento e proteção de dados, e de que forma tais aspectos podem se traduzir em vetores de ataque dentro de fluxos reais de APIs e funções.

2.3 Arquitetura e Modelos de Execução Serverless

A arquitetura por trás do *serverless computing* é construída sobre o princípio de executar aplicações nativas, independentes e orientadas a eventos. Nesse modelo, estruturado sobre o paradigma de *Function-as-a-Service* (FaaS), os provedores de nuvem assumem a responsabilidade pelo provisionamento, escalonamento e operação da infraestrutura, enquanto o desenvolvedor concentra-se apenas na lógica da aplicação. Como consequência, a execução ocorre de forma distribuída e fragmentada, com funções iniciadas, escaladas e encerradas conforme a demanda, sem a existência de servidores persistentes alocados ao usuário.

2.3.1 Modelo de execução: contêineres, cold start e warm start

O modelo de execução do FaaS baseia-se na utilização de contêineres leves, criados e gerenciados dinamicamente pelo provedor de nuvem. Nesse modelo, cada função é executada dentro de um contêiner isolado, que é inicializado somente quando necessário. Segundo a documentação oficial da Amazon Web Services, quando uma função é invocada pela primeira vez, ou após certo período de inatividade, o provedor cria um novo ambiente de execução, carrega o *runtime* e inicializa o código da função, caracterizando o processo conhecido como *cold start* (Amazon Web Services, 2025c).

Por outro lado, quando o ambiente de execução permanece ativo após uma chamada recente, o provedor pode reutilizá-lo, reduzindo consideravelmente o tempo de resposta; esse processo é

conhecido como *warm start*. A coexistência desses dois processos reforça o caráter efêmero das funções *serverless*, que não mantêm estado persistente nem servidores dedicados. Além disso, o comportamento já citado anteriormente sobre o isolamento entre as funções — que ocorre apenas em nível de contêineres que compartilham o mesmo *kernel* do sistema operacional do provedor — torna a proteção do ambiente um fator crítico e destaca a importância de compreender o impacto do modelo de execução no comportamento e na segurança das aplicações sem servidor.

2.3.2 Estrutura e fluxo de execução

A típica arquitetura de um ambiente *serverless* organiza-se a partir de componentes desacoplados que interagem entre si conforme a demanda. De maneira geral, esse modelo é composto por três elementos centrais: um ponto de entrada responsável por receber solicitações de clientes, um ou mais funções FaaS que implementam unidades independentes de lógica de negócio e um conjunto de serviços gerenciados que podem oferecer armazenamento, monitoramento, banco de dados e outros recursos auxiliares. Cada elemento opera de maneira autônoma, compondo um fluxo de execução fragmentado que opera a partir da combinação desses componentes.

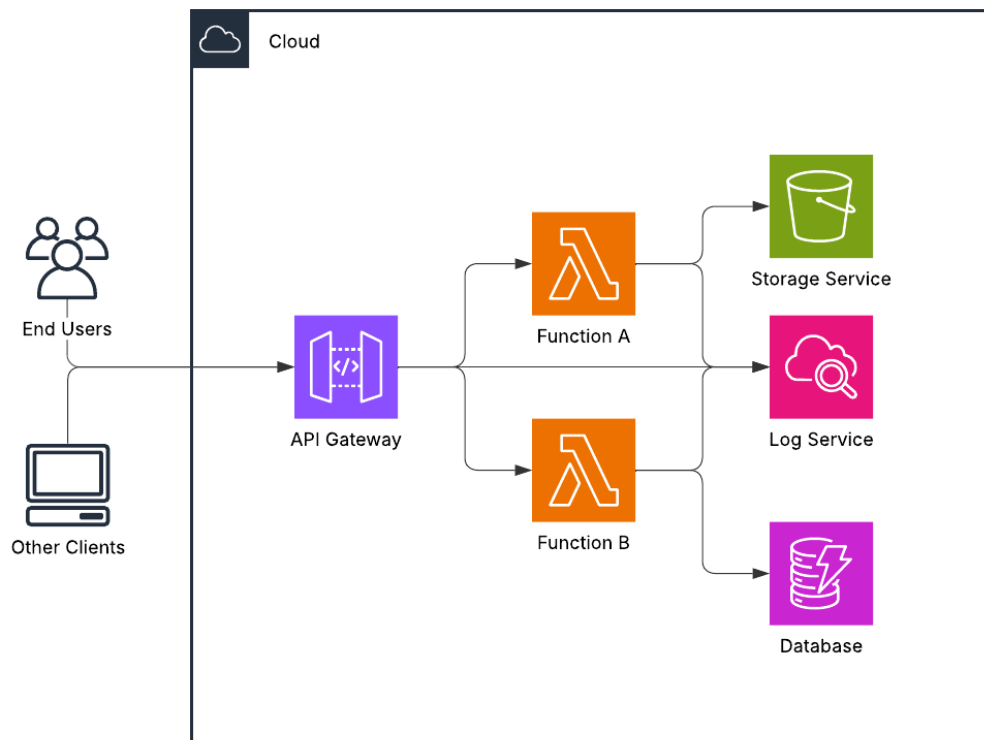


Figura 2.1: Representação genérica de uma arquitetura serverless baseada em funções FaaS, mostrando o fluxo entre clientes, gateway de API, funções e serviços auxiliares.

O diagrama apresentado na Figura 2.1 ilustra um exemplo de arquitetura típica para implementações sem servidor e FaaS. Nessa representação, usuários finais e outras aplicações enviam requisições para um serviço de gerenciamento de chamadas — frequentemente implementado por um gateway de API — que atua como intermediário entre o cliente e as funções. Cada solicitação aciona funções individuais, que processam a lógica necessária e interagem com serviços adicionais. Esses serviços representam abstrações comuns em provedores de nuvem, independentemente da tecnologia específica utilizada.

Esse fluxo evidencia a natureza modular e distribuída do *serverless*, onde cada função executa uma parte específica da aplicação, enquanto os serviços de backend fornecem suporte para persistência de dados, auditoria, monitoramento e processamento complementar. A comunicação entre componentes ocorre por meio de eventos ou chamadas diretas, compondo uma cadeia de execução que não depende de servidores persistentes, mas sim dos contêineres efêmeros.

Essa estrutura, embora ofereça benefícios significativos de escalabilidade, modularidade e redução de custos operacionais, também introduz desafios importantes de segurança. O fluxo distribuído entre clientes, gateway, funções e serviços gerenciados cria múltiplos pontos de interação suscetíveis a falhas de isolamento, perda de visibilidade, erros de configuração e exploração de permissões, temas aprofundados na próxima subseção.

2.4 Síntese crítica da revisão

A literatura analisada evidencia que os desafios de segurança em serverless não decorrem apenas de vulnerabilidades pontuais, mas de características estruturais do próprio modelo. Li, Leng e Chen organizam esses desafios em categorias amplas, como isolamento de recursos, monitoramento, gerenciamento de segurança e proteção de dados. Essa abordagem é útil para compreender o problema de forma sistêmica, pois relaciona a execução efêmera, a fragmentação dos fluxos e a dependência de serviços gerenciados a classes recorrentes de risco.

Marin, Perino e Di Pietro, por outro lado, enfatizam vulnerabilidades emergentes e cenários de ataque mais específicos, como ataques de canal lateral, condições de corrida e problemas decorrentes da execução distribuída. Enquanto Li, Leng e Chen fornecem uma taxonomia mais abrangente dos desafios de segurança, Marin, Perino e Di Pietro aprofundam aspectos técnicos associados ao isolamento e à comunicação entre funções.

As duas abordagens convergem ao apontar que o modelo serverless altera a superfície de ataque tradicional e exige mecanismos específicos de observabilidade, isolamento e controle de permissões. A principal diferença está no nível de análise: enquanto uma abordagem organiza os desafios em categorias estruturais, a outra destaca vetores técnicos mais específicos e ainda pouco explorados empiricamente.

No contexto deste trabalho, essa comparação justifica a escolha das vulnerabilidades analisadas. As falhas de isolamento e as falhas de monitoramento/rastreabilidade representam pontos de convergência entre os autores e também se manifestam diretamente na arquitetura do *Pense Bem*. Por esse motivo, a validação experimental concentrou-se nesses dois grupos, buscando relacionar o panorama teórico a evidências observáveis em um sistema real.

Capítulo 3

Metodologia

*Este capítulo descreve a metodologia utilizada no desenvolvimento deste trabalho, abrangendo o delineamento da pesquisa, os procedimentos adotados para a análise teórica e empírica e as etapas necessárias para a validação experimental das vulnerabilidades identificadas. O sistema *Pense Bem* é utilizado como estudo de caso para fundamentar a investigação e apoiar a análise prática dos riscos associados ao modelo *serverless*.*

3.1 Definição da Metodologia

A metodologia adotada neste trabalho caracteriza-se como um estudo de caso de natureza **aplicada**, com abordagem **exploratória-descritiva**. A natureza aplicada se justifica pela intenção de produzir conhecimento voltado à análise de um problema concreto: a segurança de um sistema real baseado em computação em nuvem e em serviços *serverless*. A abordagem exploratória permitiu investigar vulnerabilidades associadas a esse modelo, enquanto a abordagem descritiva possibilitou caracterizar como essas vulnerabilidades se manifestam na arquitetura analisada e quais impactos podem apresentar no contexto do sistema.

O procedimento técnico adotado, fundamentado em estudo de caso, mostrou-se adequado para observar em profundidade a arquitetura, os componentes e o comportamento operacional do sistema *Pense Bem*. Essa escolha metodológica permite relacionar os achados teóricos à realidade prática do sistema, fortalecendo a consistência da análise e possibilitando a validação experimental das vulnerabilidades priorizadas.

3.2 Etapas Metodológicas

A metodologia foi estruturada em etapas que integram levantamento teórico, análise arquitetural, identificação de vulnerabilidades e validação experimental. As atividades desenvolvidas ao longo do trabalho incluem:

- **Revisão bibliográfica e documental:** foi conduzida uma revisão sistematizada de artigos científicos, relatórios técnicos, normas internacionais e publicações especializadas sobre vulnerabilidades, ataques e desafios de segurança em ambientes *serverless*. O resultado dessa revisão consolidou o panorama teórico que fundamenta a análise realizada nos capítulos seguintes.
- **Levantamento e análise de dados sobre vulnerabilidades:** a partir da literatura e de relatórios técnicos de organizações e empresas como Datadog, Sysdig, IBM, AWS e OWASP, foi construído um panorama das vulnerabilidades e desafios mais recorrentes em computação sem servidor. Esses dados orientaram a seleção das vulnerabilidades estudadas no sistema analisado.
- **Mapeamento da arquitetura do sistema *Pense Bem*:** realizou-se a análise detalhada da arquitetura do sistema, contemplando suas funções Lambda, fluxos do Amazon API Gateway, estruturas de armazenamento no DynamoDB, uso do Amazon S3, mecanismos de autenticação, identidades IAM, registros no CloudWatch e comunicação com a instância EC2 utilizada para o modelo de recomendação. Esse mapeamento permitiu compreender a superfície de ataque e as áreas de maior exposição.
- **Identificação e classificação de vulnerabilidades:** com base nas etapas anteriores, foram identificadas vulnerabilidades estruturais relevantes ao ambiente estudado. A classificação considerou categorias relacionadas ao isolamento das funções, monitoramento distribuído, gestão de permissões, auditoria e proteção de dados, além das categorias de ameaças discutidas no referencial teórico.
- **Análise dos pontos de exposição do ambiente:** os resultados do mapeamento foram

integrados a fim de identificar pontos críticos da arquitetura, analisando como as vulnerabilidades selecionadas poderiam se manifestar no sistema *Pense Bem* e quais riscos apresentariam para o funcionamento, a segurança e a rastreabilidade da aplicação.

- **Definição e execução dos testes experimentais:** foram definidos testes direcionados às duas vulnerabilidades priorizadas no trabalho: a Vulnerabilidade A – Falhas de isolamento e compartilhamento de ambiente de execução e a Vulnerabilidade B – Falhas de monitoramento e rastreabilidade distribuída. Para a Vulnerabilidade A, foram avaliadas a reutilização de ambientes Lambda, a persistência de arquivos temporários, o comportamento sob carga e as permissões IAM associadas à função analisada. Para a Vulnerabilidade B, foram avaliados os mecanismos de logs, *tracing*, *correlation IDs*, auditoria no DynamoDB e estruturação dos registros no CloudWatch.
- **Discussão dos resultados experimentais:** os resultados obtidos foram analisados considerando as evidências coletadas nos testes, os riscos observados na arquitetura e as boas práticas de segurança aplicáveis a ambientes *serverless*. Essa etapa permitiu relacionar os achados práticos às vulnerabilidades discutidas ao longo do trabalho.

A metodologia adotada integra bases teóricas consolidadas, evidências empíricas e estudo aplicado, resultando em um processo coerente com os objetivos da pesquisa. A combinação entre revisão crítica, análise detalhada de um ambiente real e validação experimental permitiu compreender de forma aprofundada as vulnerabilidades estruturais do modelo *serverless* e discutir seus impactos no contexto do sistema *Pense Bem*.

Capítulo 4

Análise de Segurança do *Pense Bem*

Este capítulo apresenta a análise de segurança realizada sobre a implementação serverless do sistema Pense Bem. São descritos a arquitetura avaliada, seus componentes principais, os ativos críticos envolvidos e as vulnerabilidades identificadas e classificadas com base na literatura técnica e nos dados discutidos no Capítulo 2. Trata-se de uma etapa de caráter diagnóstico, cuja finalidade é estabelecer a base necessária para a validação experimental apresentada no capítulo seguinte.

4.1 Contextualização da Análise

A análise desenvolvida neste capítulo corresponde à aplicação prática da metodologia apresentada anteriormente, buscando compreender como vulnerabilidades amplamente documentadas em ambientes *serverless* se manifestam no contexto específico do sistema *Pense Bem*. O objetivo é identificar ativos críticos, mapear fluxos de execução e classificar vulnerabilidades relevantes para o ambiente estudado, oferecendo uma visão diagnóstica da superfície de ataque e dos pontos de exposição da arquitetura.

Nesta etapa, não são implementadas contramedidas, pois o foco recai sobre a caracterização das vulnerabilidades e dos elementos estruturais que podem influenciar a segurança do sistema. Essa caracterização fornece a base necessária para a validação experimental conduzida no Capítulo 5, na qual as vulnerabilidades priorizadas são avaliadas por meio de testes práticos e coleta de evidências.

4.2 O *Pense Bem*

O *Pense Bem* é uma plataforma educacional desenvolvida no contexto de um projeto de Pesquisa e Desenvolvimento (P&D), voltada ao apoio pedagógico de estudantes do Ensino Fundamental, seguindo as diretrizes estabelecidas pela Base Nacional Comum Curricular (BNCC). O sistema oferece jogos educativos, tais como *Quiz*, *Criptograma*, *Palavras Cruzadas* e *Caça-Palavras*, com o objetivo de avaliar os conhecimentos dos alunos e recomendar conteúdos personalizados com base em seu desempenho.

A plataforma realiza recomendações automáticas considerando parâmetros como pontuação obtida, número de tentativas e grau de dificuldade enfrentado em cada atividade. Além disso, disponibiliza funcionalidades de gerenciamento de e-books, capítulos, seções e questões, que integram o ecossistema pedagógico da aplicação.

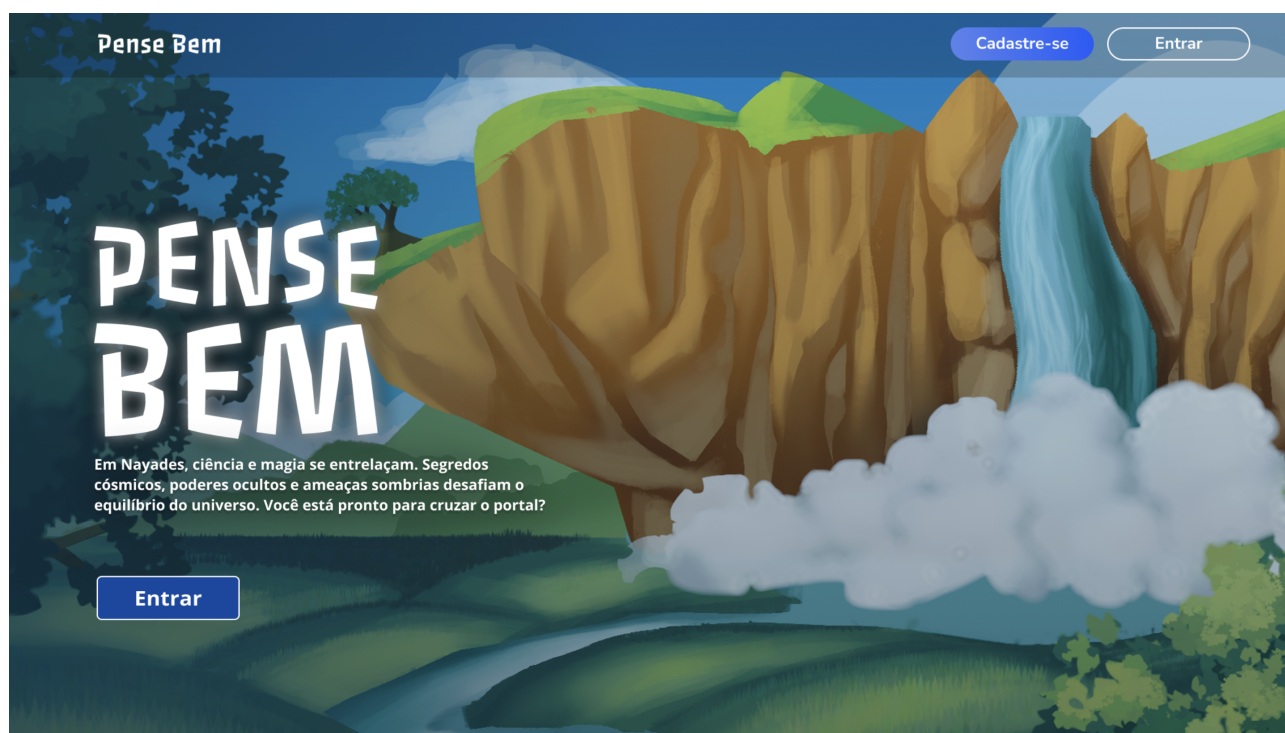


Figura 4.1: Tela inicial da plataforma *Pense Bem*, que apresenta uma experiência totalmente gamificada ao usuário.

O site do *Pense Bem*, que apresenta sua proposta pedagógica por meio de uma experiência imersiva e gamificada — na qual o próprio ambiente de navegação funciona como um jogo — pode ser acessado em: <https://pense-bem-gamma.vercel.app>

4.3 Escopo da Análise

A análise desenvolvida neste capítulo concentra-se na camada de **back-end** do sistema *Pense Bem*, responsável pela execução das regras de negócio e pela coordenação das operações realizadas pelos serviços em nuvem utilizados pela aplicação. Essa camada é construída majoritariamente segundo o modelo *serverless*, empregando funções Lambda, representantes do modelo *Function-as-a-Service* (FaaS), integradas a serviços como Amazon API Gateway, Amazon DynamoDB, Amazon S3, Amazon Cognito e Amazon CloudWatch, além de uma instância Amazon EC2 que hospeda o modelo de recomendação consumido pelas APIs. Todos esses componentes integram o ambiente alvo da análise de segurança realizada neste capítulo.

As APIs analisadas são expostas por meio do Amazon API Gateway e consumidas pela aplicação mediante requisições HTTPS. De forma geral, o acesso segue o padrão de endpoint disponibilizado pelo próprio serviço:

```
https://{api-id}.execute-api.{region}.amazonaws.com/{stage}/{rota}
```

Por questões de segurança, este trabalho não expõe os identificadores completos das URLs utilizadas no ambiente avaliado. As rotas consideradas nos testes experimentais foram acessadas de forma autorizada, utilizando token JWT válido de usuário autenticado no Amazon Cognito. Assim, a autenticação foi tratada como uma premissa operacional do ambiente, não sendo objetivo deste trabalho testar bypass de autenticação, expiração de tokens, manipulação de credenciais ou validação exaustiva de autorização em todos os endpoints da aplicação.

Não fazem parte do escopo deste trabalho o *front-end* da aplicação, a experiência do usuário e os aspectos pedagógicos associados às atividades gamificadas. Do mesmo modo, não é objetivo deste trabalho avaliar em detalhe a lógica interna do modelo de inteligência artificial executado na instância EC2, mas apenas os fluxos de comunicação entre essa instância e o restante dos componentes.

Também não faz parte do escopo a realização de uma auditoria completa sobre o conjunto total de endpoints do sistema, nem a classificação exaustiva de todas as rotas como públicas, privadas ou protegidas por autorização. As rotas avaliadas são apresentadas posteriormente no

capítulo de validação experimental e foram selecionadas por sua relação com os fluxos analisados e com as vulnerabilidades priorizadas neste trabalho.

Dessa forma, a análise concentra-se em rotas acessadas de maneira autorizada, com o objetivo de verificar vulnerabilidades estruturais relacionadas ao modelo *serverless*, especialmente isolamento de execução, permissões, monitoramento, auditoria e rastreabilidade distribuída. Assim, o foco deste capítulo recai sobre o comportamento do *back-end* atual, sua arquitetura, seus fluxos de dados e as vulnerabilidades estruturais mais relevantes identificadas nesse ambiente.

4.4 Panorama de Segurança em *Serverless*

Antes de analisar especificamente o sistema *Pense Bem*, torna-se necessário consolidar um panorama teórico aplicado sobre segurança em ambientes *serverless*, reunindo evidências, estatísticas, vulnerabilidades e desafios estruturais identificados na literatura e em relatórios técnicos recentes. Esta seção tem caráter intermediário: não se trata ainda da análise direta do objeto de estudo, mas de uma sistematização dos principais riscos e fragilidades documentados em implementações baseadas no modelo estudado. Esse panorama fundamenta a classificação e a seleção das vulnerabilidades examinadas nas seções posteriores deste capítulo.

A arquitetura e o modelo de execução apresentados na Seção 2.3 evidenciam que aplicações *serverless* se apoiam em funções efêmeras, contêineres gerenciados pelo provedor e um conjunto de serviços distribuídos que interagem por meio de eventos. Essa combinação, embora ofereça benefícios de escalabilidade e flexibilidade, introduz uma nova superfície de ataque e desloca parte significativa da responsabilidade de segurança para a camada de configuração, permissões e fluxos de dados.

4.4.1 Desafios estruturais de segurança

Os desafios de segurança em ambientes *serverless* vão além de vulnerabilidades pontuais, pois decorrem de propriedades fundamentais do próprio modelo. Li et al. (LI; LENG; CHEN, 2021)

propõem quatro categorias centrais para organizar esses desafios — *isolamento de recursos*, *monitoramento de segurança*, *gerenciamento de segurança* e *proteção de dados*. Tais eixos refletem limitações diretamente relacionadas ao uso de contêineres leves, à execução efêmera das funções e à intensa dependência de serviços gerenciados, características já introduzidas na Seção 2.3.

No que diz respeito ao isolamento, o emprego de contêineres que compartilham o mesmo *kernel* reduz a separação entre funções quando comparado a máquinas virtuais tradicionais. Esse cenário favorece ataques de canal lateral (*side-channel attacks*), aprofundados posteriormente. A literatura aponta que tais ataques ainda são pouco investigados no contexto *serverless*, indicando a necessidade de estudos adicionais (MARIN; PERINO; PIETRO, 2021).

O monitoramento também é afetado pela execução fragmentada e distribuída. Como cada função possui ciclo de vida curto e executa de forma independente, torna-se difícil reconstruir o fluxo de uma requisição e correlacionar eventos para fins de auditoria ou detecção de anomalias. Ferramentas tradicionais, baseadas em servidores persistentes, mostram limitações nesse cenário, exigindo mecanismos de rastreamento distribuído e correlação de logs — desafios amplamente discutidos por Li et al. (LI; LENG; CHEN, 2021).

O gerenciamento de segurança enfrenta dificuldades decorrentes da alta granularidade das permissões, uma vez que cabe ao desenvolvedor configurar políticas de acesso, papéis IAM (*Identity and Access Management*) e regras de interação entre serviços. A grande quantidade de componentes e a natureza dinâmica das funções contribuem para erros de configuração, como permissões excessivas ou papéis mal segmentados, ampliando a superfície de ataque.

Por fim, a proteção de dados envolve riscos associados tanto à comunicação entre funções e serviços BaaS (*Backend as a Service*) quanto ao armazenamento de informações sensíveis. Configurações inadequadas podem comprometer confidencialidade, integridade e disponibilidade — pilares diretamente influenciados pelas características arquiteturais discutidas anteriormente.

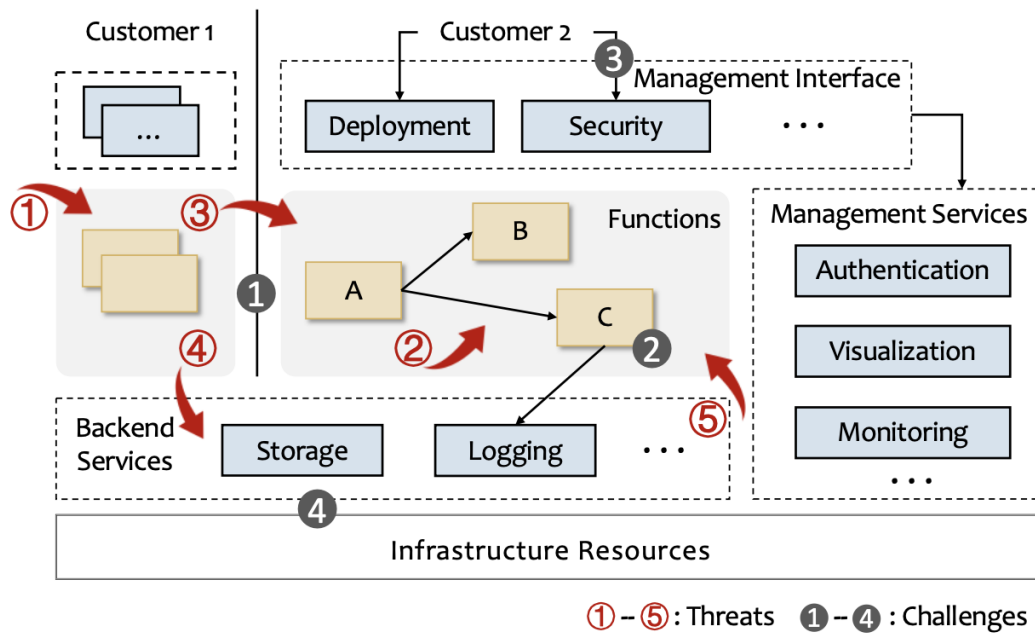


Figura 4.2: Localização das ameaças e desafios de segurança no framework serverless.

Fonte: Li, X.; Leng, X.; Chen, Y. (2021).

Além de organizarem os desafios estruturais da computação sem servidor, Li et al. (LI; LENG; CHEN, 2021) também classificam as ameaças segundo o ponto do ambiente em que se originam. Essa taxonomia complementa a análise anterior ao evidenciar que a superfície de ataque em *serverless* é distribuída e se estende por múltiplas camadas — desde o usuário final até a própria infraestrutura do provedor.

A primeira categoria refere-se a **ataques externos**, conduzidos por agentes sem privilégios, tais como injeções em APIs expostas ou explorações de falhas como *cross-site scripting*. A segunda engloba **ataques internos**, nos quais atores com algum nível de acesso podem observar padrões de execução ou manipular dados, explorando a comunicação entre funções. As três categorias seguintes refletem o caráter **multi-tenant** das plataformas sem servidor — modelo no qual múltiplos clientes compartilham a mesma infraestrutura física, ainda que isolados logicamente. Nesse contexto, surgem **ataques horizontais entre clientes** (como *side-channel attacks*, discutidos posteriormente), **ataques verticais à infraestrutura**, nos quais um adversário tenta escapar do ambiente isolado do contêiner, e **ataques verticais contra a aplicação**, decorrentes de comportamentos indevidos ou configurações inadequadas da própria plataforma.

Essa classificação reforça a necessidade de compreender tanto os desafios estruturais quanto as vulnerabilidades específicas, discutidas nas subseções seguintes.

4.4.2 Ataques de canal lateral, condições de corrida e vulnerabilidades emergentes

Além das ameaças já sistematizadas, estudos recentes destacam vulnerabilidades emergentes associadas a particularidades do modelo *serverless*. Marin et al. (MARIN; PERINO; PIETRO, 2021) enfatizam três classes relevantes: ataques de canal lateral, condições de corrida (*race conditions*) e inconsistências decorrentes da natureza distribuída das funções.

Os ataques de canal lateral exploram diretamente o modelo de isolamento baseado em contêineres, que compartilham o mesmo *kernel* e recursos físicos. Ao monitorar elementos como padrões de uso de CPU, comportamento de cache, dados de *cgroups* ou até consumo de energia, um atacante pode inferir informações sobre execuções paralelas. A literatura aponta que essa temática ainda carece de investigações mais aprofundadas, sobretudo no modelo estudado, indicando um campo aberto para pesquisas.

As condições de corrida surgem quando múltiplas funções acessam ou atualizam dados replicados simultaneamente, sem mecanismos adequados de sincronização. Isso pode resultar em estados inconsistentes, perda de integridade ou sobrescrita indevida de informações, especialmente em arquiteturas orientadas a eventos que utilizam cadeias de funções acionadas de forma autônoma. Além disso, a possibilidade de alterar parâmetros em tempo de execução amplia a complexidade para validar fluxos e garantir coerência dos dados.

Embora ataques de canal lateral sejam relevantes no contexto *serverless*, este trabalho não executa experimentos ofensivos dessa natureza, pois tais testes exigiriam técnicas de exploração de baixo nível, coleta de sinais indiretos de infraestrutura e potenciais impactos éticos e operacionais fora do escopo autorizado. Assim, os ataques de canal lateral são tratados como parte do panorama teórico, enquanto a validação experimental concentra-se em outros riscos apresentados posteriormente.

4.4.3 Ataques de negação de serviço e negação de carteira

A computação *serverless* também altera a dinâmica de ataques de negação de serviço. Embora DoS (*Denial of Service*, ataque que busca tornar um serviço indisponível) e DDoS (*Distributed Denial of Service*, mesma modalidade distribuída entre múltiplas máquinas) continuem tentando provocar indisponibilidade por exaustão de recursos, a escalabilidade automática característica desse modelo tende a absorver parte do impacto e manter o serviço funcional. Consequentemente, emergem ataques cujo objetivo principal não é derrubar o sistema, mas sim explorar o modelo de cobrança baseado em uso.

Dorsett et al. (DORSETT et al., 2025) caracterizam essa classe como *Denial of Wallet* (DoW, ataque voltado para gerar custos financeiros excessivos), um tipo de exploração no qual o adversário força execuções contínuas ou discretas de funções para elevar artificialmente o custo operacional da aplicação. Diferentemente do DoS tradicional, o DoW mira a sustentabilidade econômica do serviço, não sua disponibilidade. Os autores categorizam variantes como *Blast DDoW*, *Continual Inconspicuous DDoW* e *Background Chained DDoW*, que se diferenciam pela intensidade e padrão temporal das invocações maliciosas. Pesquisas recentes também investigam o uso de técnicas de aprendizado de máquina para identificar padrões anômalos de uso e propor modelos de cobrança mais adaptativos.

4.4.4 Implicações para o sistema Pense Bem

A análise das categorias de ameaças e vulnerabilidades emergentes evidencia que a segurança no modelo estudado demanda um equilíbrio entre responsabilidades do provedor e do desenvolvedor. Modelos multi-tenant, isolamento limitado de funções, desafios de observabilidade distribuída e a complexidade do gerenciamento de permissões tornam o ambiente suscetível tanto a ataques tradicionais quanto a novas formas de exploração.

Nesse contexto, os elementos discutidos nesta seção fundamentam a seleção das vulnerabilidades que serão analisadas ao decorrer deste capítulo. Esses elementos permitem relacionar os desafios teóricos aos riscos concretos observados na implementação real do sistema, preparando o terreno para a avaliação de segurança que será conduzida a seguir.

4.5 Normas e Boas Práticas de Segurança

A segurança em ambientes sem servidor não depende apenas da mitigação técnica de vulnerabilidades, mas também da adoção de normas, diretrizes e boas práticas amplamente reconhecidas na indústria. Esses referenciais fornecem princípios que orientam a definição de controles, a gestão de riscos e o alinhamento a requisitos legais e regulatórios. Assim, complementam os desafios discutidos na Seção 4.4 ao oferecer parâmetros objetivos para avaliação e mitigação em sistemas reais, incluindo o *Pense Bem*.

4.5.1 Normas internacionais aplicáveis à computação em nuvem

Entre os principais referenciais normativos aplicáveis ao contexto deste trabalho destaca-se a ISO/IEC 27017, que estabelece diretrizes específicas de segurança para serviços em nuvem. A norma define controles voltados ao gerenciamento de acesso, segregação de ambientes, proteção de dados e responsabilidades entre provedor e cliente (INTERNATIONAL ORGANIZATION FOR STANDARDIZATION, 2015). Sua abordagem complementa diretamente os desafios de isolamento e gerenciamento de permissões apontados anteriormente, reforçando a necessidade de políticas granulares de acesso, segregação por função e proteção contra erros de configuração.

No cenário brasileiro, a Lei Geral de Proteção de Dados Pessoais (LGPD), instituída pela Lei nº 13.709/2018, constitui o principal marco regulatório para o tratamento adequado e seguro de dados pessoais. A LGPD estabelece princípios como finalidade, necessidade, transparência, segurança e prevenção, além de exigir mecanismos que assegurem a integridade, confidencialidade e rastreabilidade das operações envolvendo dados pessoais (BRASIL, 2018). Em arquiteturas baseadas em *serverless*, tais requisitos são especialmente relevantes devido ao fluxo distribuído de dados entre múltiplos serviços, à efemeridade das execuções e à dependência de permissões adequadamente definidas.

Por fim, a OWASP publica diretrizes específicas para arquiteturas em nuvem e para funções FaaS, com destaque para o *OWASP Serverless Top 10* (OWASP Foundation, 2023), que reúne vulnerabilidades recorrentes, como configuração incorreta de permissões, funções excessivamente permissivas, exposição de endpoints e uso inseguro de eventos externos. Essas dire-

trizes são particularmente relevantes para o *Pense Bem*, uma vez que sua arquitetura se apoia fortemente em triggers, identidades (IAM) e comunicação entre serviços.

4.5.2 Boas práticas recomendadas para ambientes serverless

Além das normas formais, provedores de nuvem e organizações de segurança publicam recomendações específicas para aplicações baseadas em FaaS. Entre as referências mais consolidadas estão o *AWS Well-Architected Framework – Serverless Applications Lens* (Amazon Web Services, 2023) e o *OWASP Serverless Top 10* (OWASP Foundation, 2023), que apresentam princípios fundamentais para reduzir riscos estruturais e mitigar vulnerabilidades típicas desse modelo.

Com base nessas diretrizes, destacam-se as seguintes boas práticas:

- **Princípio do menor privilégio:** cada função deve receber apenas as permissões estritamente necessárias, conforme recomendado pelo AWS Well-Architected Lens, reduzindo a probabilidade de escalonamento de privilégios em caso de comprometimento.
- **Segmentação de funções e ambientes:** separar funções com responsabilidades distintas e evitar papéis IAM compartilhados, mitigando o impacto de falhas ou acessos indevidos entre componentes.
- **Observabilidade distribuída:** adotar ferramentas de rastreamento, correlação de eventos e monitoramento ponta a ponta, suprimindo a ausência de servidores persistentes e a fragmentação do fluxo de execução.
- **Proteção de dados em trânsito e repouso:** utilizar TLS, criptografia nativa dos provedores e mecanismos seguros de gerenciamento de chaves, conforme sugerido tanto pela AWS quanto pela OWASP.
- **Validação e filtragem de eventos:** aplicar filtros de eventos e sanitização de entradas para reduzir riscos de execuções indevidas disparadas por gatilhos controlados por atacantes, prática destacada no *OWASP Serverless Top 10*.

Essas práticas convergem diretamente com os desafios discutidos por Li et al. (LI; LENG; CHEN, 2021), sobretudo aqueles relacionados ao isolamento limitado entre funções, ao monitoramento distribuído e à necessidade de configurações de segurança altamente granulares no ambiente estudado.

4.5.3 Modelos complementares de referência

Além das recomendações específicas para *serverless*, outros modelos de referência podem apoiar a análise de segurança em arquiteturas distribuídas e ambientes AWS. Embora não tenham sido utilizados como metodologia principal deste trabalho, esses referenciais oferecem critérios complementares para comparação dos achados experimentais e para contextualização dos riscos observados no *Pense Bem*.

O NIST SP 800-204 apresenta estratégias de segurança para sistemas baseados em microsserviços, abordando aspectos como autenticação, autorização, comunicação entre serviços, monitoramento e proteção de APIs (CHANDRAMOULI, 2019). Apesar de seu foco não ser exclusivamente *serverless*, suas recomendações são aplicáveis a arquiteturas compostas por componentes desacoplados, como aquelas baseadas em API Gateway, funções Lambda e serviços gerenciados.

O CIS AWS Foundations Benchmark, por sua vez, estabelece um conjunto de recomendações de configuração segura para contas e serviços AWS (Center for Internet Security, 2026). Esse referencial é especialmente relevante para avaliar controles de identidade, logs, auditoria, monitoramento e configurações fundamentais de segurança, aproximando-se diretamente dos achados relacionados à ausência de *trails* do CloudTrail, logs de acesso e permissões IAM excessivas.

Já o MITRE ATT&CK for Cloud fornece uma base de conhecimento sobre táticas e técnicas utilizadas por adversários em ambientes de nuvem (MITRE ATT&CK, 2026). Esse modelo pode ser utilizado para relacionar os achados deste trabalho a comportamentos adversários, como abuso de permissões, descoberta de recursos, persistência, evasão de defesa e impacto sobre serviços em nuvem.

Dessa forma, os resultados obtidos no *Pense Bem* podem ser comparados não apenas com diretrizes específicas para *serverless*, como o OWASP Serverless Top 10 e o AWS Well-Architected Framework, mas também com referenciais amplos de segurança em nuvem e arquiteturas distribuídas. Essa comparação reforça a relevância dos achados experimentais, especialmente nos pontos relacionados a controle de acesso, auditoria, rastreabilidade e observabilidade.

4.6 Cenário Atual e Estatísticas Relevantes

O avanço de arquiteturas baseadas em *serverless* tem impulsionado sua adoção em ambientes corporativos. De acordo com a Datadog (Datadog, 2023), mais de 70% das organizações que utilizam serviços gerenciados em nuvem já empregam o modelo *serverless* como parte integrante de suas aplicações. Esse crescimento ampliou significativamente a superfície de ataque, sobretudo em áreas como gerenciamento de identidades, configurações inadequadas e exploração de eventos externos — fatores que se destacam nos principais relatórios de segurança em nuvem.

Um dos achados mais consistentes diz respeito ao mau gerenciamento de permissões. A Sysdig (Sysdig, 2023) demonstra que aproximadamente 90% das permissões concedidas em ambientes cloud não são utilizadas, revelando um cenário crítico de privilégios excessivos. Além disso, a mesma análise revela que identidades não humanas — associadas a funções, serviços, aplicações e integrações — já representam 58% das identidades em nuvem, superando identidades humanas tradicionais. A Figura 4.3 demonstra que funções automatizadas e serviços gerenciados constituem a maior parte do ecossistema de permissões, aumentando a complexidade de controle em arquiteturas *serverless*.

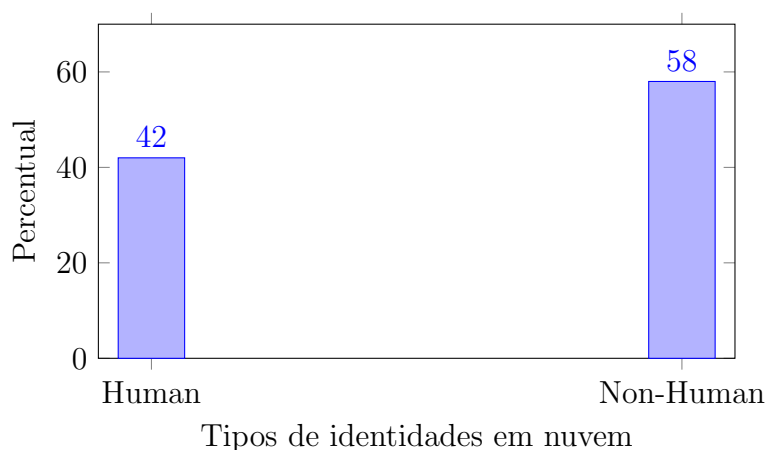


Figura 4.3: Distribuição de identidades em ambientes de nuvem.
Fonte: Sysdig (2023).

Esse aumento de identidades automatizadas reforça a necessidade de mecanismos de revisão contínua de permissões. A classificação de higiene de vulnerabilidades IAM, apresentada na Figura 4.4, indica que grande parte das organizações ainda se encontra nos níveis mais baixos de maturidade: 39% praticam apenas entre 0% e 20% de boas práticas efetivas.

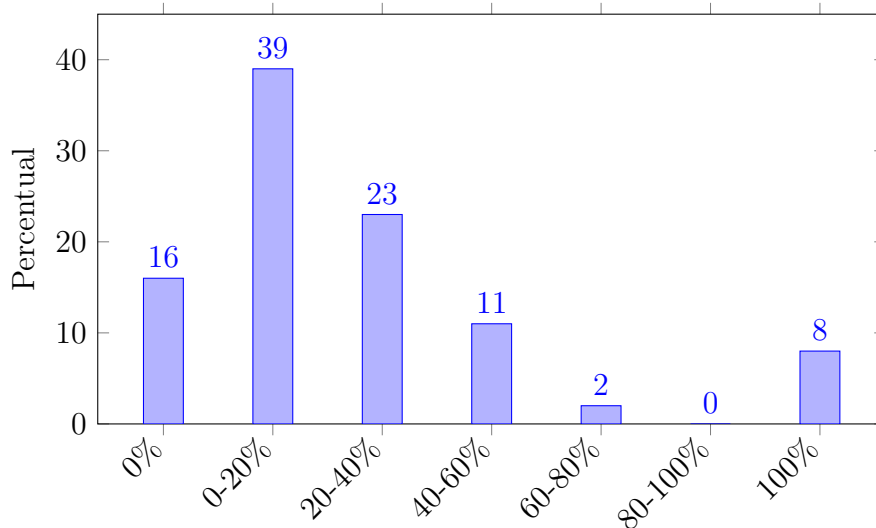


Figura 4.4: Classificação de higiene de vulnerabilidades IAM.
Fonte: Sysdig (2023).

A combinação desses fatores — permissões excessivas, predominância de identidades não humanas e baixa maturidade em IAM — compõe um cenário particularmente sensível para ambientes *serverless*. Como funções FaaS operam de forma distribuída e são frequentemente acionadas por eventos externos, qualquer permissão configurada de forma inadequada pode

resultar em escalonamento de privilégios, execução indevida de funções ou comprometimento de fluxos de dados.

No que diz respeito a ameaças econômicas, ataques de *Denial of Wallet* (DoW) têm recebido atenção crescente pela sua capacidade de explorar diretamente o modelo de cobrança *pay-as-you-go*. Dorsett et al. (DORSETT et al., 2025) demonstram que workloads *serverless* podem ser induzidos a gerar custos da ordem de milhares de dólares por dia, mesmo sem qualquer violação de segurança ou interrupção do serviço. Em um dos exemplos analisados pelos autores, um atacante foi capaz de inflar a fatura de um serviço AWS em aproximadamente USD 1.300 por dia apenas com requisições rejeitadas, evidenciando a fragilidade financeira desse modelo.

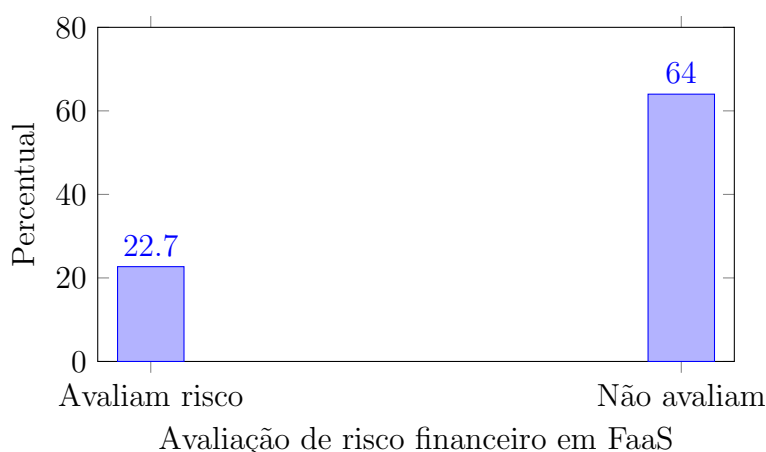


Figura 4.5: Organizações que realizam avaliação de risco financeiro relacionado a FaaS.
Fonte: Dorsett et al. (2025).

O mesmo estudo revela ainda um baixo nível de conscientização organizacional sobre esse tipo de ameaça: embora 100% dos profissionais entrevistados afirmassem conhecer ataques DDoS tradicionais, apenas 13,6% tinham familiaridade com DoW, e nenhum soube confirmar se sua organização já havia sido alvo desse tipo de abuso. Além disso, apresentado na Figura 4.5, 64% das organizações que utilizam funções FaaS não realizam qualquer avaliação de risco financeiro relacionada ao seu uso, indicando uma lacuna entre modelos tradicionais e os riscos econômicos emergentes. Mesmo entre aquelas que conduzem avaliações, não foram identificadas metodologias ou ferramentas padronizadas para estimar impactos financeiros — o que reforça a dificuldade de prever, mitigar ou mesmo detectar ataques dessa natureza.

Assim, o panorama estatístico atual evidencia que vulnerabilidades relacionadas ao isola-

mento entre funções, observabilidade distribuída e gestão granular de permissões, discutidas nas seções anteriores, são as mais relevantes para o FaaS. Essa consolidação quantitativa fundamenta a escolha das vulnerabilidades que serão aprofundadas ao longo deste capítulo, alinhando a revisão teórica ao estudo de caso do sistema *Pense Bem*.

4.7 Arquitetura do *Pense Bem*

A arquitetura do *Pense Bem* é composta por um conjunto de serviços gerenciados da AWS (*Amazon Web Services*) que, em conjunto, sustentam o funcionamento da aplicação e possibilitam a execução das funcionalidades educacionais e gamificadas da plataforma. Sua estrutura geral segue o padrão típico de arquiteturas *serverless* baseadas em funções FaaS, semelhante ao modelo apresentado na Figura 2.1, no qual um gateway de API atua como ponto de entrada, funções são acionadas sob demanda e serviços auxiliares fornecem persistência, autenticação e suporte operacional. Esse arranjo resulta em um ambiente altamente escalável, distribuído e orientado a eventos, características essenciais para o funcionamento dinâmico do sistema.

O fluxo de autenticação e gerenciamento de usuários é realizado pelo Amazon Cognito, serviço responsável pela emissão e validação de tokens JWT utilizados em todas as operações protegidas. As requisições enviadas pelos usuários são encaminhadas ao Amazon API Gateway, que atua como camada intermediária entre o front-end e o back-end, organizando rotas, validando credenciais e direcionando cada operação à função Lambda correspondente. A lógica de negócio principal é executada por meio de funções FaaS hospedadas no AWS Lambda, responsáveis por manipular questões, registrar jogadas, processar avaliações iniciais, consultar dados do aluno e interagir com demais serviços da infraestrutura.

O armazenamento das informações persistentes é realizado pelo Amazon DynamoDB, banco de dados NoSQL que contém registros de usuários, sessões, avaliações, atividades, questões, capítulos, seções e e-books, além dos dados relacionados ao desempenho do aluno. Arquivos estáticos, como imagens, mídias e conteúdos associados, são mantidos no Amazon S3, que também armazena elementos utilizados no fluxo operacional da aplicação. O envio de e-mails transacionais, como confirmações de cadastro e recuperação de senha, é executado pelo Amazon

Simple Email Service (SES), enquanto o AWS Amplify facilita a integração entre o front-end em React e os serviços Cognito e API Gateway.

Além dos serviços mencionados, a arquitetura inclui uma instância Amazon EC2 destinada à hospedagem do modelo de inteligência artificial utilizado para recomendar conteúdos ao aluno com base em seu histórico de desempenho. A instância integra o escopo desta análise, uma vez que suas interfaces de comunicação são acessadas diretamente pelo back-end do sistema.

A Figura 4.6 apresenta o diagrama geral da arquitetura utilizada pela plataforma, destacando a interligação entre os serviços e o fluxo operacional da aplicação.

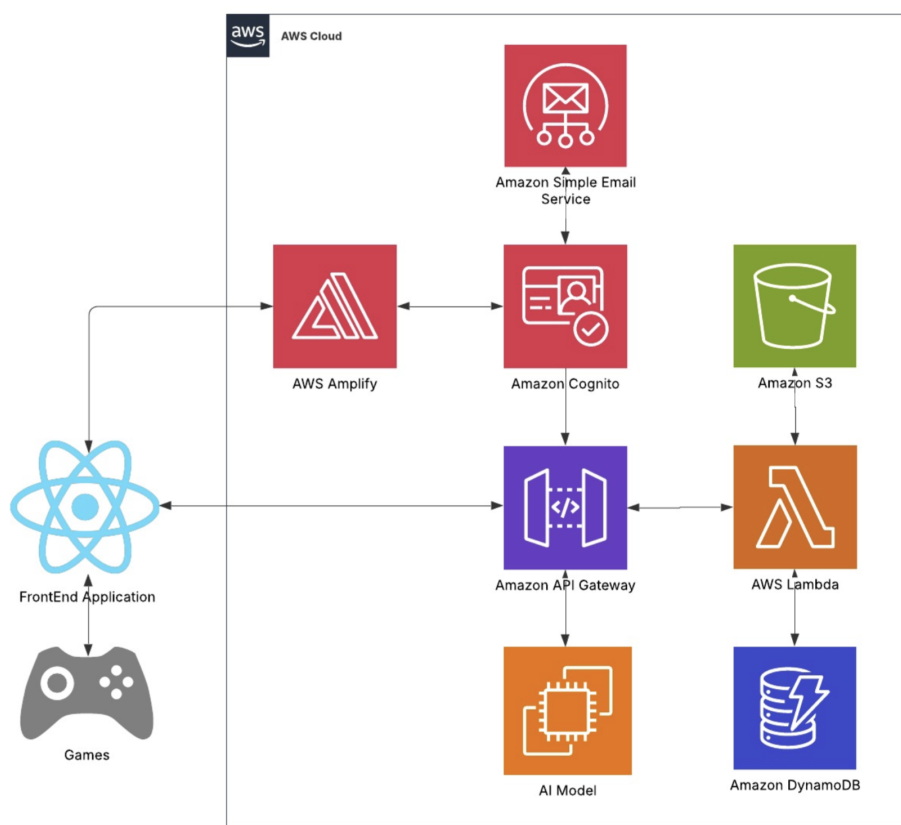


Figura 4.6: Arquitetura geral do *Pense Bem*.

4.7.1 Componentes Principais do Back-end

O back-end do *Pense Bem* é composto majoritariamente por funções AWS Lambda, que executam a lógica responsável pelo processamento das funcionalidades centrais do sistema. Essas funções são ativadas a partir de eventos provenientes do API Gateway e respondem por ativi-

dades como criação e atualização de questões, registro de partidas, avaliação de desempenho e comunicação com o modelo de recomendação. A execução ocorre de forma efêmera e isolada, conforme preconiza o modelo *Function-as-a-Service*, o que influencia diretamente os requisitos de segurança associados a isolamento, permissões e monitoramento distribuído.

O Amazon API Gateway opera como ponto de entrada para o back-end, intermediando todas as chamadas realizadas pela aplicação. Além da validação das credenciais enviadas pelos usuários, o serviço também organiza a estrutura de rotas, controla limites de requisição e realiza a integração direta com funções Lambda, garantindo o fluxo seguro e eficiente das operações.

O armazenamento das entidades do sistema é responsabilidade do Amazon DynamoDB, que contém dados referentes a usuários, sessões, avaliações, jogos, jogadas, questões, capítulos, seções e e-books. A estrutura dessas entidades corresponde ao modelo apresentado na Figura 4.7, derivada do diagrama de classes que fundamenta o funcionamento da aplicação e orienta a organização dos dados.

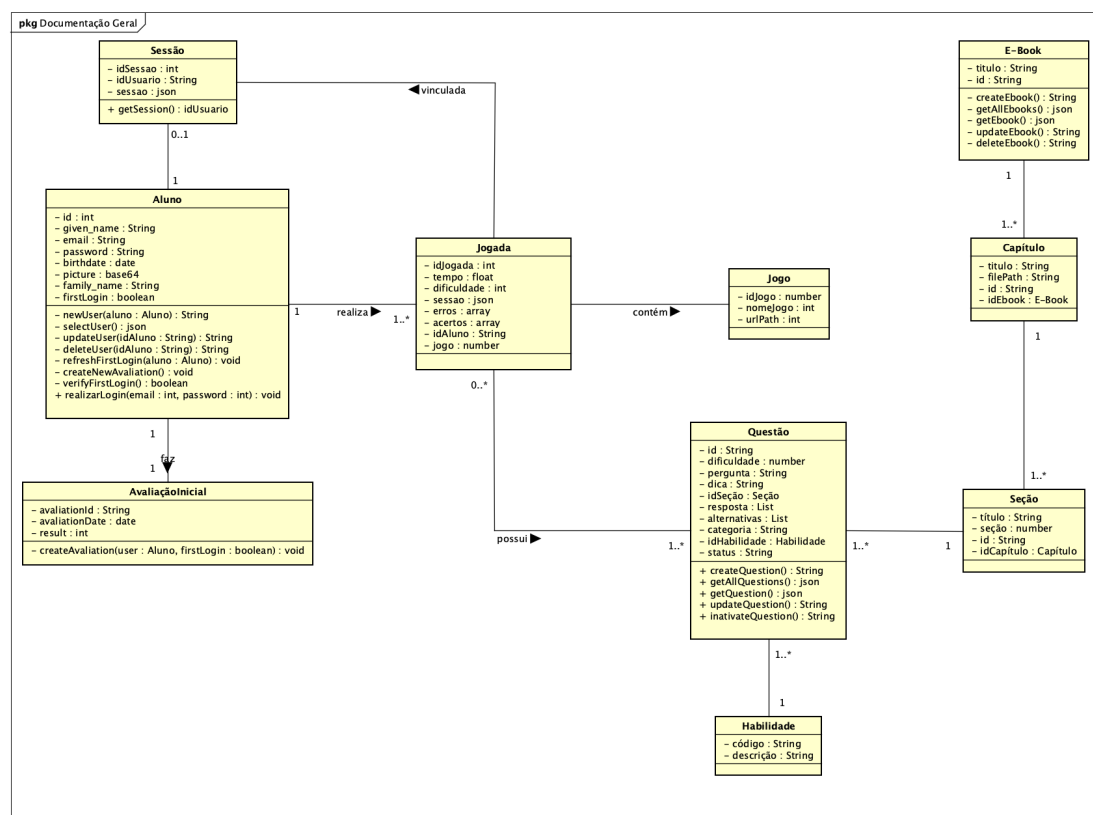


Figura 4.7: Modelo de classes do sistema *Pense Bem*.

4.7.2 Fluxo de Dados e Execução

O fluxo operacional do back-end segue o padrão característico de arquiteturas baseadas em funções FaaS. O usuário acessa a aplicação por meio do front-end desenvolvido em React, que utiliza o AWS Amplify para realizar o processo de autenticação no Amazon Cognito. Após a autenticação, o usuário inicia requisições direcionadas ao Amazon API Gateway, que valida o token JWT anexado às solicitações e encaminha cada operação para a função Lambda apropriada.

Uma vez acionada, a função Lambda executa a lógica necessária e pode interagir com diferentes serviços da infraestrutura. Dependendo da natureza da operação, a função pode realizar consultas ou atualizações no DynamoDB, acessar arquivos armazenados no S3 ou comunicar-se diretamente com o modelo de inteligência artificial hospedado na instância EC2. Após o processamento, a resposta é devolvida ao API Gateway e, em seguida, ao usuário.

Todos os eventos e execuções relacionados ao back-end são registrados no Amazon CloudWatch Logs, o que possibilita acompanhar o comportamento das funções, identificar falhas e monitorar padrões de execução. Esse fluxo evidencia a interdependência entre os componentes e destaca a importância de mecanismos eficientes de autorização, isolamento e rastreamento para garantir a segurança.

4.8 Identificação de Ativos, Ameaças e Vulnerabilidades

A seguir, são apresentados os ativos de maior relevância para a análise de segurança, bem como as vulnerabilidades selecionadas para validação experimental neste trabalho.

4.8.1 Ativos Críticos Identificados

A análise da arquitetura do *Pense Bem* permitiu identificar um conjunto de ativos cuja exposição representa riscos significativos sob a ótica do modelo de ameaças discutido no Capítulo 2, especialmente aquele representado na Figura 4.2. Conforme o framework de Li, Leng e Chen (LI; LENG; CHEN, 2021), esses ativos se relacionam diretamente às categorias de ameaças externas,

internas, horizontais e verticais próprias de ambientes *serverless*.

Assim, os principais ativos críticos observados no *backend* do sistema são:

- **Funções Lambda:** unidades centrais de execução da lógica de negócio, vulneráveis tanto a ataques externos quanto internos, além de possíveis interferências horizontais entre funções em cenários de *multi-tenancy*.
- **Permissões IAM:** definem o escopo de atuação das funções e demais serviços. Permissões excessivas configuram risco de escalonamento de privilégios, associado aos desafios de gestão de segurança.
- **Endpoints expostos pelo Amazon API Gateway:** constituem a porta de entrada do sistema e estão sujeitos a ataques externos, como injeções, manipulação de requisições e acionamento indevido de funções.
- **Dados armazenados no Amazon DynamoDB:** incluem registros sensíveis de usuários, sessões, jogos e questões. Comprometimentos nessa camada podem refletir ataques verticais, que atravessam diferentes níveis da infraestrutura.
- **Eventos externos que acionam funções:** representam potenciais vetores para *event injection*, associados a ataques externos e internos.
- **Logs e mecanismos de monitoramento:** essenciais para rastreamento e auditoria. Falhas nessa camada se relacionam diretamente ao desafio de monitoramento, ampliando a superfície de ataque não observada.
- **Fluxo financeiro de execução das funções e da instância EC2:** tanto as funções Lambda quanto a instância EC2 estão sujeitas à exploração econômica do modelo de cobrança, como em ataques de *Denial of Wallet*.

O quadro 4.1 sintetiza esses ativos, relacionando-os aos serviços utilizados pelo *Pense Bem*, aos fluxos mais diretamente impactados e às categorias de ameaças predominantes segundo o modelo de Li et al. (LI; LENG; CHEN, 2021).

Quadro 4.1: Ativos críticos do backend do *Pense Bem*, serviços associados e principais ameaças.

Ativo crítico	Serviços associados	Principais ameaças (Li et al., 2021)
Funções Lambda	AWS Lambda	Ataques externos; ataques internos; interferência entre funções; exploração econômica (<i>Denial of Wallet</i>).
Permissões IAM	AWS IAM	Escalonamento de privilégios; acesso indevido; mau gerenciamento de políticas.
Endpoints da API	Amazon API Gateway	Ataques externos como injeção e manipulação de requisições; acionamento malicioso de funções.
Dados no DynamoDB	Amazon DynamoDB	Vazamento ou manipulação de dados; ataques verticais à infraestrutura; exposição de informações sensíveis.
Eventos externos	Triggers do API Gateway e integrações	<i>Event injection</i> ; execução maliciosa via eventos externos ou internos.
Logs e monitoramento	Amazon CloudWatch Logs	Falhas de auditoria; invisibilidade de incidentes; desafios de monitoramento distribuído.
Custos de execução	AWS Lambda e Amazon EC2	Ataques econômicos (<i>Denial of Wallet</i>) decorrentes de execuções forçadas ou tráfego malicioso.

4.8.2 Seleção das Vulnerabilidades Analisadas

A definição das vulnerabilidades aprofundadas neste capítulo baseia-se no modelo conceitual apresentado no Capítulo 2, especialmente na taxonomia de ameaças e desafios estruturais proposta por Li, Leng e Chen (LI; LENG; CHEN, 2021). O framework dos autores identifica cinco classes de ameaças — externas, internas, horizontais, verticais e oriundas da própria plataforma

— e quatro desafios estruturais recorrentes em ambientes *serverless*: isolamento de recursos, monitoramento distribuído, gestão de segurança e proteção de dados. Esses elementos, combinados às análises de Marin, Perino e Di Pietro (MARIN; PERINO; PIETRO, 2021), evidenciam que fragilidades relacionadas ao isolamento de execução e à falta de observabilidade estão entre os riscos mais críticos e inerentes ao modelo FaaS.

A literatura técnica foi complementada pelo OWASP Serverless Top 10 (OWASP Foundation, 2023) e pelos dados empíricos apresentados na Seção 4.6, que apontam para um cenário de baixa maturidade em gestão de permissões e forte predominância de identidades não humanas em ambientes em nuvem. Tais dados reforçam que falhas de isolamento e monitoramento tendem a amplificar riscos já existentes, como permissões excessivas e acionamento indevido de funções.

Considerando o panorama teórico e estatístico, bem como as características específicas da arquitetura analisada, foram selecionadas duas vulnerabilidades centrais para este trabalho:

Vulnerabilidade A *Falhas de isolamento e compartilhamento de ambiente de execução.* Essa vulnerabilidade envolve riscos associados à reutilização de contêineres, exposição de dados residuais, interferência entre funções e ataques *side channel*. Trata-se de uma fragilidade estrutural amplamente documentada em FaaS e altamente relevante para o back-end do *Pense Bem*, que processa dados sensíveis de usuários e atividades.

Vulnerabilidade B *Falhas de monitoramento e rastreabilidade distribuída.* No *Pense Bem*, a execução envolve diversos serviços acionados por eventos — como API Gateway, Lambda, DynamoDB, S3 e EC2 — o que fragmenta o fluxo de requisições e dificulta sua reconstrução. Diferentemente de servidores tradicionais, onde logs ficam centralizados, ambientes *serverless* distribuem registros entre serviços, tornando mais complexa a correlação de eventos, mesmo com ferramentas como o CloudWatch. Essa limitação reduz a visibilidade operacional e favorece falhas não detectadas.

Essas vulnerabilidades foram priorizadas por representarem riscos intrínsecos ao modelo *serverless*, por serem diretamente aplicáveis ao ambiente estudado e por permitirem validação

prática no contexto do *Pense Bem*. A partir dessa seleção, os testes experimentais apresentados no Capítulo 5 foram estruturados para observar evidências relacionadas ao isolamento de ambientes Lambda, ao uso de permissões IAM, à cobertura de auditoria e à rastreabilidade distribuída.

4.8.3 Ameaças Associadas

As vulnerabilidades selecionadas se conectam diretamente às categorias de ameaças discutidas no modelo de Li, Leng e Chen (LI; LENG; CHEN, 2021), abrangendo riscos externos, internos, horizontais e verticais característicos de arquiteturas *serverless*. No entanto, nem todas as ameaças descritas na literatura se manifestam de forma homogênea em todos os ambientes; por isso, esta subseção diferencia entre ameaças inerentes ao modelo e aquelas plausíveis no contexto específico do *Pense Bem*.

De acordo com essa distinção, as principais ameaças consideradas são:

- **Escalonamento de privilégios.** Trata-se de uma ameaça estrutural do modelo *serverless*, frequentemente associada a permissões excessivas. No caso do *Pense Bem*, esse risco é plausível, pois a análise preliminar de IAM indica que algumas funções possuem escopos amplos de acesso a DynamoDB e S3, o que pode ampliar o impacto de uma função comprometida.
- **Vazamento de dados entre funções.** É uma ameaça documentada na literatura como decorrente da reutilização de contêineres e da possível presença de dados residuais. Embora o *Pense Bem* não opere em ambiente multi-tenant, essa ameaça permanece relevante, pois a plataforma processa dados sensíveis e a integridade do isolamento entre funções Lambda não é controlada diretamente pelo desenvolvedor.
- **Execução maliciosa a partir de eventos externos.** No modelo *serverless*, funções podem ser acionadas a partir de múltiplas fontes de eventos. No *Pense Bem*, as ativações ocorrem principalmente via Amazon API Gateway e, em menor escala, por eventos internos de aplicação. Assim, o risco de *event injection* é menor quando comparado a sistemas com

múltiplos gatilhos (como S3, IoT ou filas), mas ainda é relevante, pois uma requisição manipulada pode acionar funções críticas ou causar consumo excessivo.

- **Falhas de configuração IAM.** Risco destacado tanto na literatura quanto nas estatísticas apresentadas na Seção 4.6. No contexto do *Pense Bem*, esse vetor é particularmente plausível, pois o uso de múltiplas funções Lambda e interações com DynamoDB e S3 aumenta a chance de permissões demasiadamente amplas ou mal segmentadas.
- **Ataques baseados em eventos e exploração financeira (*Denial of Wallet*).** Esses ataques são característicos de sistemas *pay-as-you-go* e consistem no acionamento massivo e indevido de funções com o objetivo de gerar custos. No *Pense Bem*, esse risco é plausível, pois o API Gateway é público e pode ser alvo de uso abusivo, mesmo quando as requisições não culminam em operações bem-sucedidas. A dependência de Lambda e de uma instância EC2 dedicada ao modelo de recomendação torna o impacto financeiro ainda mais significativo.

Em conjunto, essas ameaças evidenciam como falhas de isolamento e monitoramento distribuído podem amplificar riscos operacionais, econômicos e de privacidade no ambiente *serverless* do *Pense Bem*. Essa relação justifica a escolha das vulnerabilidades priorizadas e orienta os testes experimentais realizados no capítulo seguinte.

4.9 Análise dos Pontos de Exposição do Ambiente

A identificação dos pontos de exposição do *Pense Bem* permite compreender onde as vulnerabilidades estruturais do modelo *serverless* se manifestam de forma mais evidente. Essa análise aprofunda aspectos diretamente relacionados às vulnerabilidades de isolamento (Vulnerabilidade A) e de monitoramento distribuído (Vulnerabilidade B), destacando trechos específicos do *back-end* que apresentam maior probabilidade de impacto sobre a segurança do sistema.

4.9.1 Critérios de Risco Utilizados

A avaliação dos pontos de exposição considerou quatro critérios principais:

- **Probabilidade:** estimativa de ocorrência com base no funcionamento das funções, integrações e permissões observadas.
- **Impacto:** avaliação dos efeitos sobre os pilares de segurança (confidencialidade, integridade e disponibilidade).
- **Exposição:** grau de acesso ou visibilidade que o componente apresenta diante de atores externos ou internos.
- **Contexto do *Pense Bem*:** características próprias da aplicação, como suas rotas críticas, uso de funções Lambda e dependência de múltiplos serviços gerenciados.

Esses critérios orientam a análise das vulnerabilidades apresentadas a seguir.

4.9.2 Avaliação da Vulnerabilidade A — Falhas de Isolamento

A Vulnerabilidade A refere-se à possibilidade de interferência entre funções Lambda ou exposição de dados residuais durante a reutilização de ambientes de execução. No *Pense Bem*, esse risco se manifesta principalmente em operações que exigem processamento encadeado, como a geração de recomendações ou a avaliação inicial do usuário, nas quais uma mesma função realiza consultas ao DynamoDB, acessa arquivos no S3 e comunica-se com a instância EC2.

Do ponto de vista técnico, falhas de isolamento podem ocorrer sempre que múltiplas execuções compartilham o mesmo contêiner subjacente, conforme documentado na literatura. Essa condição permite que variáveis, arquivos temporários ou estados da execução anterior permaneçam acessíveis na nova chamada, favorecendo vazamento de dados ou inconsistências entre execuções. Ambientes com alta taxa de requisições, como rotas de jogos e avaliações, apresentam maior probabilidade de ocorrência devido ao reaproveitamento frequente de contêineres.

Em termos de impacto, essa vulnerabilidade pode comprometer diretamente a confidencialidade dos dados do aluno, uma vez que a manipulação de informações sensíveis faz parte

do fluxo habitual das funções Lambda. No contexto do *Pense Bem*, embora não haja multi-tenancy entre clientes diferentes, as funções processam perfis distintos de usuários e sessões, o que torna o isolamento essencial para evitar exposição indevida.

4.9.3 Avaliação da Vulnerabilidade B — Falhas de Monitoramento e Rastreabilidade Distribuída

A Vulnerabilidade B envolve a incapacidade de correlacionar eventos distribuídos entre Lambda, API Gateway, DynamoDB, S3 e EC2. No *Pense Bem*, essa limitação é evidente devido ao volume de funções independentes e à ausência de um mecanismo unificado que permita rastrear, em um único fluxo, o caminho percorrido por cada operação.

Essa falha afeta principalmente rotas críticas, como as que tratam da lógica de jogos e das recomendações. Nessas rotas, uma única requisição pode originar múltiplos registros de log distribuídos e sem correlação direta, dificultando a detecção de anomalias, a identificação de chamadas indevidas ou a observação de comportamentos que possam indicar abuso de permissões. Essa condição amplia a superfície invisível do sistema e intensifica o risco de ataques silenciosos ou de execuções maliciosas não detectadas.

No contexto específico do *Pense Bem*, a dependência de uma instância EC2 torna a falta de rastreamento ainda mais relevante. Como não há um mecanismo unificado de monitoramento entre Lambda e EC2, o acionamento indevido do modelo pode permanecer oculto, dificultando a identificação de uso abusivo e contribuindo para riscos financeiros associados a *Denial of Wallet*.

4.9.4 Matriz de Riscos

A partir dos critérios definidos e da análise contextual, o Quadro 4.2 apresenta a matriz de riscos correspondente às vulnerabilidades avaliadas.

Quadro 4.2: Matriz de riscos para as vulnerabilidades analisadas.

Vulnerabilidade		Probabilidade	Impacto (CIA)	Nível de Risco
Vulnerabilidade A: Falhas de isolamento		Média	Alta (confidencialidade e integridade)	Alta
Vulnerabilidade B: Falhas de monitoramento		Alta	Média-Alta (de- tecção e disponi- bilidade)	Alta

Com base nessa matriz, ambas as vulnerabilidades apresentam risco elevado, justificando sua priorização na etapa de validação experimental. A combinação entre rotas críticas, funções com múltiplas integrações, permissões sensíveis, dependência de serviços gerenciados e ausência de rastreamento unificado reforça a necessidade de avaliar, na prática, como essas fragilidades se manifestam no ambiente analisado.

Capítulo 5

Validação Experimental das Vulnerabilidades

Este capítulo apresenta a validação experimental das vulnerabilidades identificadas na análise de segurança do sistema Pense Bem. A partir dos pontos de exposição mapeados anteriormente, foram definidos e executados testes direcionados às duas vulnerabilidades priorizadas neste trabalho: falhas de isolamento e falhas de monitoramento e rastreabilidade distribuída. Os testes têm como objetivo complementar a análise teórica com evidências práticas obtidas a partir da execução de requisições, observação de logs, análise de métricas e verificação de recursos de monitoramento da arquitetura.

5.1 Contextualização da Validação Experimental

A análise realizada no capítulo anterior permitiu identificar que o ambiente *serverless* do *Pense Bem* apresenta dois grupos de vulnerabilidades prioritárias. A primeira, denominada Vulnerabilidade A, refere-se às falhas de isolamento e ao risco de persistência de dados entre execuções de funções Lambda. A segunda, denominada Vulnerabilidade B, refere-se às falhas de monitoramento e rastreabilidade distribuída em fluxos que envolvem múltiplos serviços, como Amazon API Gateway, AWS Lambda, Amazon DynamoDB, Amazon S3, Amazon CloudWatch e Amazon EC2.

Essas vulnerabilidades foram selecionadas por estarem diretamente relacionadas aos desafios estruturais de segurança em *serverless*, conforme discutido por Li, Leng e Chen (LI; LENG;

CHEN, 2021) e Marin, Perino e Di Pietro (MARIN; PERINO; PIETRO, 2021). Além disso, sua relevância é reforçada pelo OWASP Serverless Top 10, que destaca riscos associados à exposição de dados sensíveis, configuração inadequada, monitoramento insuficiente e superfície de ataque orientada a eventos (OWASP Foundation, 2023).

A validação experimental foi conduzida com o objetivo de verificar como essas vulnerabilidades podem se manifestar no contexto específico do *Pense Bem*. Diferentemente de uma análise somente conceitual, os testes descritos neste capítulo buscam observar o comportamento do sistema diante de cenários práticos, incluindo chamadas consecutivas entre usuários, uso do diretório temporário das funções Lambda, carga artificial sobre rotas críticas, análise de logs e verificação da existência de mecanismos de rastreamento distribuído.

5.2 Objetivos dos Testes

O objetivo geral desta etapa foi validar empiricamente as vulnerabilidades previamente identificadas, relacionando-as aos fluxos reais do *back-end* do *Pense Bem*.

Os testes buscaram:

- verificar se dados podem permanecer em memória ou em armazenamento temporário entre invocações sucessivas de uma mesma função Lambda;
- avaliar se a reutilização do ambiente de execução pode ampliar riscos de exposição de dados de usuários;
- observar o comportamento da aplicação sob carga, considerando métricas de latência, taxa de erro, volume de requisições e reutilização de contêineres;
- verificar permissões associadas à função Lambda analisada, considerando o princípio do menor privilégio;
- verificar a cobertura de logs nos principais componentes;
- avaliar a existência ou ausência de identificadores de correlação entre serviços;

- analisar a cobertura de auditoria sobre operações realizadas no DynamoDB;
- relacionar os resultados obtidos aos critérios de probabilidade, impacto, exposição e contexto do *Pense Bem*.

5.3 Ambiente Experimental

Os testes foram conduzidos no *back-end* do sistema *Pense Bem*, composto por serviços gerenciados da AWS e por uma instância Amazon EC2 responsável pelo modelo de recomendação. O ambiente analisado segue a arquitetura descrita no capítulo anterior, na qual o Amazon API Gateway atua como ponto de entrada das requisições, as funções AWS Lambda executam a lógica de negócio, o Amazon DynamoDB armazena dados persistentes, o Amazon S3 armazena arquivos associados à aplicação e a instância EC2 executa o modelo de inteligência artificial.

O Quadro 5.1 apresenta os principais componentes considerados na validação experimental.

Quadro 5.1: Componentes considerados no ambiente experimental.

Componente	Função no ambiente	Relevância para os testes
Amazon API Gateway	Exposição das rotas REST do sistema.	Ponto de entrada das requisições e origem dos registros iniciais de acesso.
AWS Lambda	Execução da lógica de negócio.	Principal componente avaliado nos testes de isolamento e persistência de estado.
Amazon DynamoDB	Armazenamento de dados persistentes.	Contém dados de usuários, sessões, jogos, pontuações e recomendações.
Amazon S3	Armazenamento de arquivos e recursos associados.	Pode ser acessado por funções Lambda durante fluxos de processamento.
Amazon EC2	Hospedagem do modelo de recomendação.	Componente presente na arquitetura analisada, considerado no mapeamento dos fluxos distribuídos, embora os testes experimentais tenham se concentrado nos mecanismos de isolamento, auditoria e rastreabilidade das APIs e funções Lambda.
Amazon CloudWatch	Registro de logs e métricas.	Base para análise de observabilidade, alarmes e rastreamento de execuções.
AWS CloudTrail	Registro de eventos de serviços AWS.	Apoio à auditoria de chamadas e operações realizadas nos serviços.
AWS X-Ray	Rastreamento distribuído.	Permite verificar se há visibilidade de ponta a ponta entre serviços.

As rotas selecionadas para os experimentos foram escolhidas por manipularem dados relevantes da aplicação e por envolverem múltiplos serviços durante sua execução. O Quadro 5.2 sintetiza essas rotas.

Quadro 5.2: Rotas selecionadas para os testes experimentais.

Rota	Justificativa da escolha
POST /nivelPlayer	Persiste pontuação e recomendação do jogador, manipulando dados sensíveis de desempenho.
GET /nivel-Player/latest/{sub}	Consulta o nível mais recente de um jogador, com suporte a filtros opcionais por capítulo e seção.
GET /chapter	Consulta a lista de capítulos disponíveis na API de questões, sendo utilizada nos testes de observabilidade e correlação da API de questões.

5.4 Ferramentas Utilizadas

A execução dos testes envolveu ferramentas de requisição HTTP, geração de carga, auditoria e observabilidade. A seleção dessas ferramentas teve como objetivo permitir tanto a execução controlada de cenários específicos quanto a coleta de evidências operacionais.

Quadro 5.3: Ferramentas utilizadas na validação experimental.

Ferramenta	Categoria	Uso nos testes
curl	Requisições HTTP manuais.	Execução controlada de chamadas às rotas críticas.
K6	Teste de carga.	Geração de carga, medição de latência, taxa de erro e volume de requisições.
AWS CLI	Auditoria e coleta de dados.	Consulta a logs, métricas, alarmes e configurações dos serviços AWS.
Amazon CloudWatch Logs	Observabilidade.	Verificação dos logs das funções Lambda e análise de eventos de execução.
AWS CloudTrail	Auditoria de eventos.	Verificação de trails configurados e cobertura de data events no DynamoDB.
AWS X-Ray	Rastreamento distribuído.	Verificação da configuração de <i>tracing</i> no API Gateway — ausência confirmada nos três gateways.

O uso do K6 foi relevante para observar o comportamento do sistema sob carga. Essa

análise permitiu coletar métricas relacionadas à latência, erros e estabilidade da aplicação, complementando os testes funcionais de segurança. Já o CloudWatch, o CloudTrail e o X-Ray foram utilizados para avaliar a capacidade de monitoramento e rastreabilidade entre os componentes da arquitetura.

5.5 Testes Relacionados à Vulnerabilidade A: Falhas de Isolamento

A Vulnerabilidade A está relacionada à possibilidade de persistência de dados ou interferência entre execuções de funções Lambda. Em arquiteturas *serverless*, funções podem reutilizar ambientes de execução previamente inicializados, comportamento do *warm start*. Embora essa reutilização seja uma característica prevista e útil para reduzir latência, ela exige atenção para que dados sensíveis não sejam armazenados indevidamente fora do escopo da requisição (Amazon Web Services, 2025e, 2025b).

No contexto do *Pense Bem*, esse risco é relevante porque as funções Lambda processam informações associadas a usuários, sessões, jogos, pontuações e recomendações. Assim, falhas de implementação envolvendo variáveis globais, cache local ou arquivos temporários podem afetar a confidencialidade e a integridade dos dados dos alunos.

5.5.1 Teste A1: Persistência de Variáveis entre Execuções

O primeiro teste teve como objetivo verificar se variáveis declaradas fora do manipulador principal da função Lambda poderiam manter estado entre execuções consecutivas. Esse cenário foi analisado por meio de requisições sequenciais com usuários distintos, observando se informações de uma execução anterior poderiam permanecer acessíveis em chamadas posteriores.

O procedimento experimental consistiu no envio de requisições à rota `POST /nivelPlayer` em padrão alternado entre dois usuários distintos ($A \rightarrow B \rightarrow A \rightarrow B$), com intervalo de cinco a dez segundos entre cada chamada. O intervalo foi utilizado para demonstrar que a reutilização do contêiner não dependia necessariamente de cenários de sobrecarga, mas poderia ocorrer

como parte do comportamento natural de reutilização de ambientes (*warm start*) do AWS Lambda (Amazon Web Services, 2025e).

Para tornar observável o comportamento do ambiente de execução durante o teste, foi utilizada uma instrumentação temporária e controlada por meio da variável de ambiente `SECURITY_TEST_MODE=true`. Essa configuração habilitou, apenas no contexto experimental, a inclusão dos cabeçalhos `X-Container-Id`, `X-Warm-Start` e `X-Invocation-Count` nas respostas da API. Esses cabeçalhos permitiram identificar o ambiente lógico responsável por cada requisição, diferenciar execuções iniciadas por *cold start* daquelas processadas em *warm start* e observar a contagem de invocações realizadas no mesmo ambiente.

Como alternativa menos invasiva, seria possível inferir a ocorrência de *cold starts* e *warm starts* a partir dos logs padrão do AWS Lambda no Amazon CloudWatch, especialmente pela presença do campo *Init Duration* nas linhas de relatório. A presença desse campo indica a inicialização de um novo ambiente de execução, enquanto sua ausência em invocações subsequentes sugere o reaproveitamento de um ambiente já existente. Entretanto, essa abordagem não permite identificar com precisão se o mesmo contêiner lógico foi reutilizado entre usuários distintos, nem associar cada requisição a um identificador específico de ambiente. Por esse motivo, optou-se pela instrumentação controlada, restrita ao ambiente experimental e sem execução de ações destrutivas ou exploração ofensiva.

Após as chamadas, foram analisados os logs da função no Amazon CloudWatch e as respostas obtidas por meio da rota `GET /nivelPlayer/latest/{sub}`, buscando identificar possíveis sinais de persistência indevida de estado entre usuários distintos.

O critério de confirmação definido para o teste foi a presença de dados do usuário A em logs, variáveis ou respostas associadas ao usuário B. Esse comportamento indicaria que informações de uma execução anterior permaneceram acessíveis no ambiente reutilizado.

A análise dos cabeçalhos de resposta evidenciou que um único contêiner atendeu todas as dez invocações registradas, sendo nove delas classificadas como *warm start* e apenas uma como *cold start*, conforme ilustrado na Figura 5.1. O resultado demonstra que o mesmo ambiente de execução foi reutilizado continuamente entre chamadas de usuários distintos ao longo do

experimento.

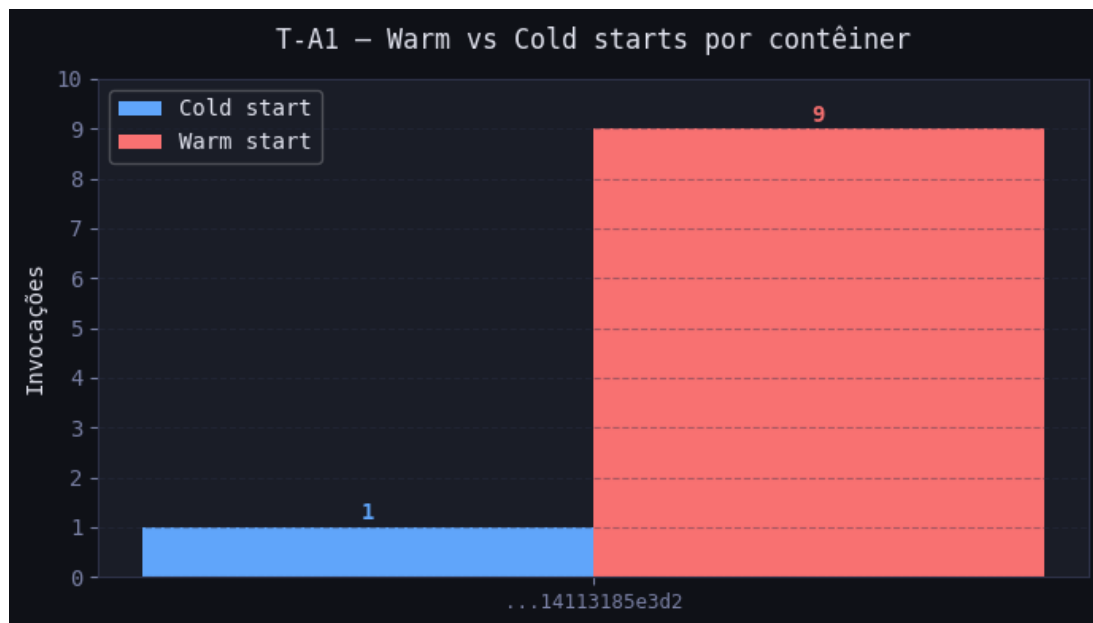


Figura 5.1: Distribuição de *warm* e *cold starts* por contêiner durante o Teste A1.

Quadro 5.4: Resultado do Teste A1: persistência de variáveis entre execuções.

Item avaliado	Resultado
Rota testada	POST /nivelPlayer.
Tipo de dado envolvido	Pontuação, recomendação e identificador do usuário.
Técnica utilizada	Requisições sequenciais com usuários diferentes.
Instrumentação aplicada	Cabeçalhos de rastreamento para identificação de contêiner, tipo de inicialização e contagem de invocações no mesmo ambiente.
Evidência analisada	Logs do CloudWatch e respostas da API.
Resultado observado	Reutilização persistente do mesmo contêiner em cenário de <i>warm start</i> .
Impacto principal	Confidencialidade e integridade.
Situação da vulnerabilidade	Confirmada como condição de risco de implementação.

Os resultados obtidos demonstram que o risco identificado não decorre de uma falha direta do serviço AWS Lambda, mas do uso inadequado do ambiente de execução pela aplicação. A reutilização persistente do mesmo contêiner para usuários distintos cria a condição necessária

para que variáveis globais, estados mantidos fora do escopo do manipulador ou dados temporariamente armazenados permaneçam acessíveis entre execuções sucessivas. Assim, o teste não caracteriza, por si só, a ocorrência de vazamento efetivo de dados, mas confirma uma condição operacional que pode favorecer esse tipo de falha caso a aplicação armazene informações sensíveis de forma inadequada.

Esse comportamento está alinhado às vulnerabilidades descritas pela OWASP Serverless Top 10 na categoria A3:2017 – *Sensitive Data Exposure*, especialmente em cenários nos quais aplicações assumem incorretamente que o ambiente de execução será destruído após cada invocação (OWASP Foundation, 2023). Assim, embora a reutilização de contêineres represente uma otimização legítima do modelo FaaS, o armazenamento inadequado de informações sensíveis fora do escopo da execução torna-se um risco relevante para arquiteturas *serverless* que processam dados de usuários.

5.5.2 Teste A2: Persistência de Arquivos no Diretório Temporário /tmp

O segundo teste avaliou o comportamento do diretório temporário /tmp, único diretório gravável no ambiente Lambda. Esse diretório pode ser utilizado para armazenamento temporário durante o processamento de arquivos ou dados intermediários. Entretanto, a documentação da AWS indica que o ambiente de execução pode ser reutilizado, o que implica que arquivos temporários podem permanecer disponíveis entre invocações caso não sejam removidos explicitamente (Amazon Web Services, 2025e).

Esse teste também se relaciona diretamente à categoria A3:2017 – *Sensitive Data Exposure* do OWASP Serverless Top 10. Essa categoria destaca que o armazenamento de dados no diretório /tmp sem remoção após o uso pode resultar em vazamento de informações sensíveis, especialmente quando o desenvolvimento assume incorretamente que o contêiner será destruído ao final de cada execução (OWASP Foundation, 2023). A OWASP também associa esse risco ao conceito de *Insecure Shared Space*, no qual resíduos de execuções anteriores podem permanecer acessíveis no ambiente reutilizado.

No contexto do *Pense Bem*, esse comportamento é relevante porque funções Lambda podem

manipular dados associados a sessões, jogos, desempenho dos alunos, arquivos de apoio e informações intermediárias utilizadas durante o processamento. Caso esses dados sejam gravados temporariamente em `/tmp` e não sejam removidos, uma execução posterior pode acessar resíduos deixados por invocações anteriores, comprometendo a confidencialidade das informações.

O procedimento consistiu em instrumentar a função para listar os arquivos existentes em `/tmp` antes do processamento da nova requisição. Em seguida, foram executadas dez chamadas consecutivas, verificando nos logs se arquivos criados em invocações anteriores continuavam presentes. A cada nova invocação, a função registrava a quantidade de arquivos encontrados antes da escrita do novo arquivo temporário.

O critério de confirmação foi a presença de arquivos pré-existent no diretório temporário antes da execução da nova chamada. Esse critério permite verificar se o ambiente foi reutilizado e se os dados temporários permaneceram disponíveis entre invocações.

Durante a execução do teste, observou-se que 9 das 10 invocações encontraram arquivos criados por execuções anteriores. Além disso, a contagem de arquivos cresceu de forma linear, iniciando em 0 e chegando a 9 arquivos ao final do experimento. Esse comportamento demonstrou que o contêiner não foi destruído entre as chamadas e que os arquivos temporários permaneceram disponíveis no ambiente de execução.

A Figura 5.2 apresenta a evolução da quantidade de arquivos pré-existent encontrados no diretório `/tmp` ao longo das invocações realizadas durante o experimento. Observa-se crescimento linear da quantidade de resíduos persistidos entre execuções consecutivas, indicando que o ambiente Lambda permaneceu reutilizado durante praticamente todo o teste. Esse comportamento demonstra que os arquivos temporários não foram removidos automaticamente entre invocações, permanecendo acessíveis no mesmo contêiner de execução.



Figura 5.2: Evolução da quantidade de arquivos pré-existent no diretório `/tmp` durante o Teste A2.

Quadro 5.5: Resultado do Teste A2: persistência de arquivos em `/tmp`.

Item avaliado	Resultado
Recurso testado	Diretório temporário <code>/tmp</code> da função Lambda.
Categoria OWASP relacionada	A3:2017 – <i>Sensitive Data Exposure</i> .
Conceito associado	<i>Insecure Shared Space</i> .
Técnica utilizada	Escrita e leitura de arquivos temporários entre invocações.
Quantidade de invocações	10 invocações consecutivas.
Evidência analisada	Logs do CloudWatch e contagem de arquivos pré-existent em <code>/tmp</code> .
Resultado observado	9 das 10 invocações encontraram arquivos de execuções anteriores, com crescimento linear da contagem de arquivos de 0 para 9.
Impacto principal	Confidencialidade.
Situação da vulnerabilidade	Confirmada como risco operacional e como evidência prática de espaço compartilhado inseguro.

Os resultados obtidos demonstram empiricamente o mecanismo descrito pela OWASP Serverless Top 10. O teste confirmou que arquivos deixados no diretório `/tmp` podem permanecer acessíveis em invocações posteriores, contrariando a suposição de que o ambiente Lambda, nesta implementação, é destruído ao final de cada execução. Assim, caso dados sensíveis fossem armazenados nesse diretório sem remoção explícita, haveria risco de exposição em uma execução futura.

Esse resultado reforça que o uso do diretório temporário deve ser acompanhado de limpeza explícita ao final da execução. No contexto do *Pense Bem*, essa preocupação é relevante para rotas que manipulam sessões, arquivos de apoio, dados intermediários ou informações associadas às atividades dos usuários. A ausência de limpeza do diretório `/tmp` pode transformar um recurso temporário em um espaço compartilhado inseguro, ampliando a superfície de exposição da aplicação.

5.5.3 Teste A3: Stress de *Warm Start* com K6

O terceiro teste teve como objetivo avaliar o comportamento das funções Lambda sob carga concorrente, verificando a ocorrência de *warm starts* e o reaproveitamento de contêineres em um cenário de maior volume de requisições. Diferentemente dos testes anteriores, que analisaram execuções sequenciais, este experimento buscou observar o comportamento operacional do ambiente durante um cenário de stress controlado.

O experimento foi executado com a ferramenta K6, utilizando um roteiro de carga dividido em três fases: aquecimento, stress e descida. Durante a fase principal do teste, foram utilizados 20 usuários virtuais simultâneos, com o objetivo de intensificar o reaproveitamento de ambientes Lambda e aumentar a probabilidade de reutilização de contêineres.

Quadro 5.6: Configuração do teste de stress com K6 no Teste A3.

Fase	Duração	Usuários virtuais	Objetivo
Aquecimento	15 segundos	5	Inicialização e estabilização do ambiente Lambda.
Stress	60 segundos	20	Forçar maior reaproveitamento de contêineres sob carga concorrente.
Descida	15 segundos	0	Encerramento gradual da carga aplicada.

O critério de confirmação do teste foi a predominância de *warm starts* e a ocorrência recorrente de eventos de reuso de contêiner durante a execução sob carga. Esses indicadores demonstram que múltiplas requisições puderam ser processadas por ambientes previamente

inicializados.

Os resultados do experimento indicaram a execução de 1846 requisições, das quais 922 foram classificadas como *warm starts* e apenas uma como *cold start*. Além disso, foram registrados 599 eventos de reuso de contêiner, com latência p95 de 628 ms, confirmando o reaproveitamento recorrente do ambiente de execução.

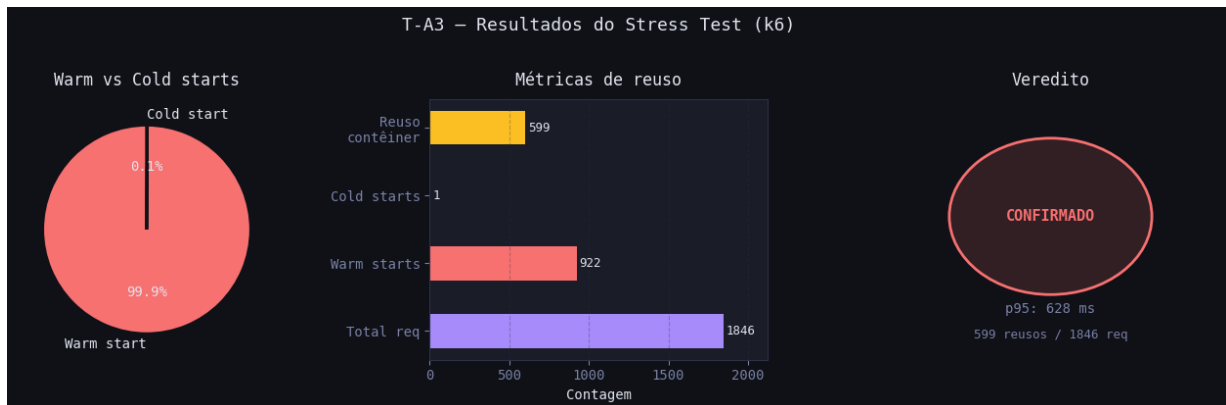


Figura 5.3: Resultados do stress test com K6 para avaliação de *warm starts* e reuso de contêiner no Teste A3.

A Figura 5.3 apresenta a predominância de *warm starts* durante o teste, bem como as métricas agregadas de reutilização de contêineres e o veredito do experimento. Os resultados evidenciam que, sob carga concorrente, o reuso de ambientes Lambda torna-se frequente e operacionalmente relevante.

Quadro 5.7: Resultado do Teste A3: stress de *warm start* com K6.

Item avaliado	Resultado
Ferramenta utilizada	K6.
Técnica utilizada	Geração de carga concorrente com usuários virtuais.
Total de requisições	1846 requisições.
<i>Warm starts</i>	922 ocorrências registradas.
<i>Cold starts</i>	1 ocorrência registrada.
Eventos de reuso de contêiner	599 eventos registrados.
Latência p95	628 ms.
Resultado observado	Predominância de <i>warm starts</i> e reuso recorrente de contêiner sob carga.
Impacto principal	Aumento da probabilidade de persistência de estado entre execuções.
Situação da vulnerabilidade	Confirmada como condição operacional relevante para a Vulnerabilidade A.

Os resultados demonstram que, sob carga concorrente, o reaproveitamento de ambientes Lambda ocorre de forma recorrente. Embora esse comportamento represente uma característica legítima do modelo FaaS, ele amplia a importância de práticas seguras relacionadas ao gerenciamento de estado, especialmente em aplicações que processam dados sensíveis. Dessa forma, o Teste A3 complementa os resultados dos Testes A1 e A2 ao demonstrar que o reuso de contêiner permanece frequente mesmo em cenários de maior volume de requisições.

5.5.4 Teste A4: Avaliação de Controle de Acesso e Permissões Excessivas na Role `games-api-role`

O quarto teste teve como objetivo avaliar se a role IAM associada à função Lambda da API de jogos seguia o princípio do menor privilégio (*least privilege*). Esse teste complementa os experimentos anteriores da Vulnerabilidade A, pois a reutilização de contêineres e a persistência de estado tornam-se mais críticas quando a função executa com permissões amplas sobre recursos sensíveis da conta AWS.

A avaliação foi realizada por meio do comando `aws iam simulate-principal-policy`, que permite simular decisões de autorização sem executar ações reais sobre os recursos. Foram testadas ações relacionadas ao DynamoDB, S3, Cognito e Lambda, classificando-se como críticas aquelas capazes de excluir tabelas, remover buckets ou alterar políticas de acesso.

Esse teste se relaciona diretamente à categoria A5:2017 – *Broken Access Control* do OWASP Serverless Top 10, que descreve o risco em implementações *serverless* com permissões excessivas ou acesso a recursos desnecessários. De acordo com essa categoria, funções que não seguem o princípio do menor privilégio podem se tornar uma porta de entrada para acesso indevido a recursos da conta, ampliando o impacto de uma função comprometida (OWASP Foundation, 2023).

A Figura 5.4 apresenta o resultado da simulação realizada para a role `games-api-role`. Observa-se que ações críticas como `dynamodb:DeleteTable`, `s3:DeleteBucket` e `s3:PutBucketPolicy` foram avaliadas como permitidas. Também foram permitidas ações amplas como `dynamodb:Scan`, `dynamodb:PutItem` e `s3:GetObject`. Por outro lado, ações como `cognito-idp:AdminDeleteUser` e `lambda:InvokeFunction` foram negadas.

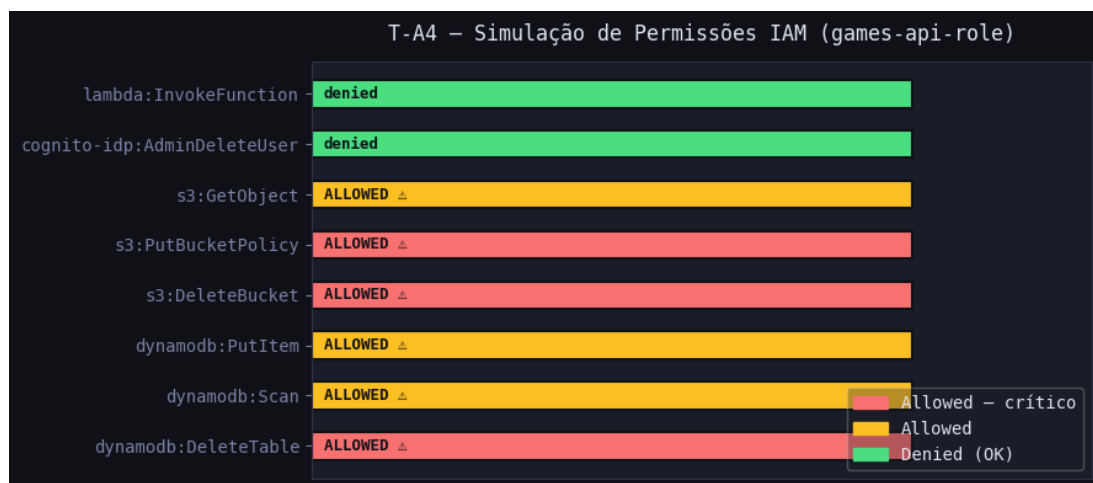


Figura 5.4: Resultado da simulação de permissões IAM para a role `games-api-role`.

Quadro 5.8: Resultado do Teste A4: permissões avaliadas na role `games-api-role`.

Ação avaliada	Necessária para a função?	Resultado
<code>dynamodb:PutItem</code>	Sim	ALLOWED
<code>dynamodb:Scan</code>	Não	ALLOWED
<code>dynamodb>DeleteTable</code>	Não	ALLOWED
<code>s3:GetObject</code>	Discutível	ALLOWED
<code>s3>DeleteBucket</code>	Não	ALLOWED
<code>s3:PutBucketPolicy</code>	Não	ALLOWED
<code>cognito-idp:AdminDeleteUser</code>	Não	DENIED
<code>lambda:InvokeFunction</code>	Não	DENIED

Os resultados demonstram que a role `games-api-role` possui permissões além do necessário para a operação esperada da função. Embora algumas permissões sejam compatíveis com o funcionamento da API, como `dynamodb:PutItem`, outras ações permitidas apresentam alto impacto e não são justificadas para o fluxo normal da aplicação, como `dynamodb>DeleteTable`, `s3>DeleteBucket` e `s3:PutBucketPolicy`.

A permissão `dynamodb:Scan` também representa risco relevante, pois permite leitura ampla dos dados armazenados, ampliando a exposição de informações sensíveis em caso de comprometimento da função. No mesmo sentido, permissões amplas sobre S3 podem permitir acesso indevido a objetos ou alteração da política de buckets, afetando confidencialidade, integridade e disponibilidade.

Dessa forma, o Teste A4 confirma uma falha de controle de acesso no contexto da role analisada. A combinação entre reutilização de ambientes Lambda, demonstrada nos Testes A1, A2 e A3, e permissões excessivas na role da função amplia o impacto potencial de uma exploração. Caso um código malicioso fosse executado no contexto da Lambda, as credenciais disponíveis poderiam ser utilizadas para executar ações destrutivas ou acessar recursos além do necessário para a aplicação.

Assim, o resultado reforça a necessidade de substituir políticas amplas por permissões específicas, limitadas às ações e recursos efetivamente exigidos pela função, em conformidade com

o princípio do menor privilégio.

5.6 Testes Relacionados à Vulnerabilidade B: Falhas de Monitoramento, Auditoria e Rastreabilidade Distribuída

A Vulnerabilidade B está relacionada à dificuldade de rastrear e correlacionar eventos distribuídos entre os diferentes serviços que compõem uma solução baseada em *serverless*. Em aplicações baseadas nesse modelo, uma única requisição pode atravessar múltiplos componentes, como API Gateway, Lambda, DynamoDB, S3 e EC2, gerando registros fragmentados e sem correlação direta (Amazon Web Services, 2025a; OWASP Foundation, 2023).

Embora provedores de nuvem disponibilizem mecanismos de auditoria e monitoramento, como Amazon CloudWatch, AWS CloudTrail e AWS X-Ray, esses recursos dependem de configuração explícita e, em muitos casos, não oferecem rastreabilidade completa por padrão. A ausência de identificadores de correlação, logs estruturados e cobertura adequada de auditoria reduz a visibilidade operacional da aplicação e dificulta a detecção de comportamentos anômalos ou abusivos.

No contexto do *Pense Bem*, esse risco é relevante porque o *back-end* depende de múltiplos serviços distribuídos para processar jogos, recomendações de questões e armazenamento de dados dos usuários. Dessa forma, falhas de monitoramento podem dificultar a identificação de chamadas indevidas, uso abusivo da instância EC2, falhas de autorização ou execuções anômalas envolvendo dados dos alunos.

5.6.1 Teste B1: Ausência de Access Logs e Tracing no API Gateway

O primeiro teste avaliou a configuração de observabilidade do Amazon API Gateway, verificando a existência de *access logs* e rastreamento distribuído por AWS X-Ray nas APIs utilizadas pelo *Pense Bem*. O objetivo foi identificar se as requisições recebidas pelo gateway poderiam ser auditadas e correlacionadas posteriormente.

No API Gateway, os *access logs* registram informações sobre quem acessou a API e como

esse acesso ocorreu, podendo incluir dados como identificador da requisição, IP de origem, método HTTP, rota acessada, status de resposta e latência (Amazon Web Services, 2025f). Já o tracing com AWS X-Ray permite acompanhar e analisar requisições conforme elas percorrem o API Gateway e os serviços integrados, auxiliando na identificação de gargalos, falhas e fluxos distribuídos (Amazon Web Services, 2025d).

Foram analisadas as APIs `dev-games-api`, `dev-questions-api` e `dev-modelo-inferencia-api`. O procedimento consistiu na inspeção das configurações dos *stages* do API Gateway, verificando se os *access logs* estavam habilitados e se o tracing via X-Ray encontrava-se ativo.

Os resultados indicaram que nenhuma das APIs possuía *access logs* habilitados e que o tracing distribuído também estava desativado em todos os casos analisados, conforme ilustrado na Figura 5.5.

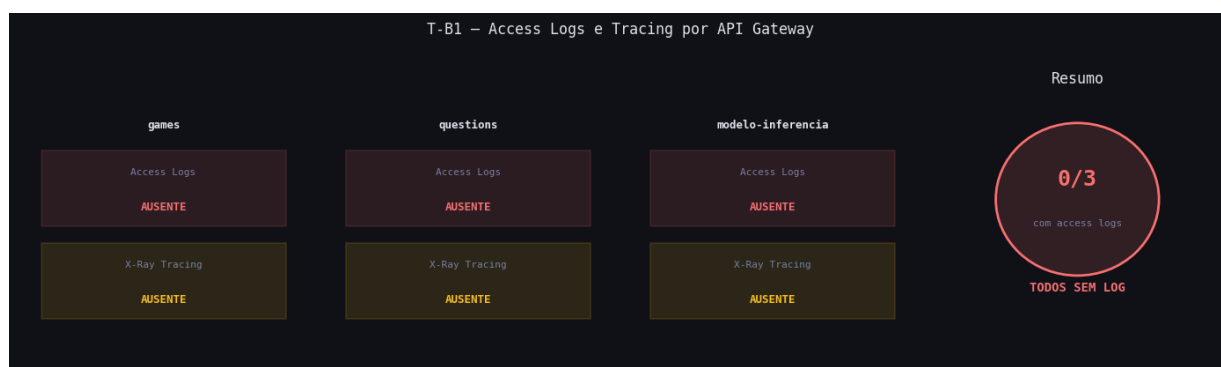


Figura 5.5: Avaliação de *access logs* e tracing distribuído no API Gateway durante o Teste B1.

Quadro 5.9: Resultado do Teste B1: auditoria no API Gateway.

Item avaliado	Resultado
Serviços analisados	<code>dev-games-api</code> , <code>dev-questions-api</code> e <code>dev-modelo-inferencia-api</code> .
Técnica utilizada	Inspeção da configuração dos <i>stages</i> do API Gateway.
<i>Access logs</i>	Ausentes nas três APIs analisadas.
Tracing distribuído (X-Ray)	Desabilitado nas três APIs analisadas.
Impacto principal	Ausência de rastreabilidade das requisições recebidas pelo gateway.
Situação da vulnerabilidade	Confirmada como falha de auditoria e monitoramento.

A ausência de *access logs* impede identificar informações essenciais para investigação de incidentes, como IP de origem, endpoint acessado, frequência de chamadas, códigos de resposta e padrões de abuso. Esse cenário está alinhado à categoria A10:2017 – *Insufficient Logging and Monitoring* do OWASP Serverless Top 10, que destaca que atacantes podem se beneficiar da falta de monitoramento e de resposta tempestiva para atingir seus objetivos sem serem detectados (OWASP Foundation, 2023). No contexto de *serverless computing*, essa limitação é agravada pela distribuição dos registros entre serviços gerenciados e pela dependência de configurações explícitas de auditoria. Assim, quando o API Gateway não registra as requisições recebidas e o tracing distribuído permanece desabilitado, torna-se mais difícil identificar acessos não autorizados, execuções excessivas, uso indevido de endpoints ou comportamentos anômalos em tempo hábil.

5.6.2 Teste B2: Ausência de Correlation IDs entre Serviços

Após verificar, no Teste B1, a ausência de registros no ponto de entrada da aplicação, o Teste B2 analisou os logs gerados pelas funções Lambda. O objetivo foi verificar se esses registros continham identificadores suficientes para relacionar uma execução da função à requisição que a originou e, assim, permitir a reconstrução do fluxo entre os serviços.

O procedimento consistiu na geração de tráfego nas APIs `games` e `questions`, seguida da

análise dos logs registrados no Amazon CloudWatch. Foram avaliados três elementos principais: presença de identificadores do API Gateway (`requestId`), presença de *correlation IDs* customizados e propagação de identificadores do AWS X-Ray.

Os resultados demonstraram ausência completa de *correlation IDs* customizados e ausência dos identificadores do API Gateway nos logs das funções Lambda. Apenas parte das linhas da API `games` continha identificadores de trace do X-Ray, enquanto a API `questions` não apresentou qualquer propagação de rastreamento distribuído.

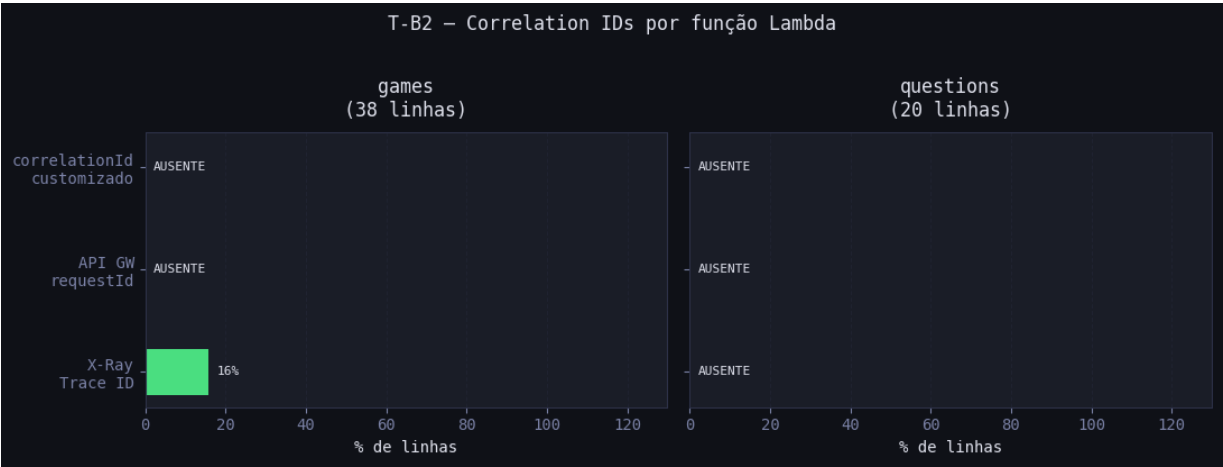


Figura 5.6: Avaliação de *Correlation IDs* e rastreamento distribuído no Teste B2.

Quadro 5.10: Resultado do Teste B2: rastreamento distribuído entre serviços.

Item avaliado	Resultado
Serviços analisados	dev-games-us-east-1 e dev-questions-us-east-1.
Técnica utilizada	Análise dos logs no CloudWatch após geração de tráfego controlado.
<i>Correlation IDs</i> customizados	Ausentes em ambas as funções Lambda.
Identificador do API Gateway	Ausente nos logs analisados.
Propagação de X-Ray	Parcial na API <code>games</code> e ausente na API <code>questions</code> .
Impacto principal	Impossibilidade de correlação entre gateway, Lambda e demais serviços.
Situação da vulnerabilidade	Confirmada como falha de rastreabilidade distribuída.

Os resultados indicam que os logs das funções Lambda não possuem identificadores compartilhados suficientes para vincular, de forma confiável, uma execução ao evento que a originou. Assim, mesmo quando há registros no CloudWatch, a ausência de campos padronizados de correlação dificulta a investigação de incidentes e a reconstrução de fluxos distribuídos.

5.6.3 Teste B3: Cobertura de Auditoria no DynamoDB

O terceiro teste avaliou a cobertura de auditoria das operações realizadas no Amazon DynamoDB, serviço responsável por armazenar dados relevantes da aplicação, como sessões, pontuações, recomendações e registros associados ao desempenho dos usuários. O objetivo foi verificar se ações de leitura, escrita, varredura e exclusão de itens poderiam ser rastreadas posteriormente por meio do AWS CloudTrail.

Essa verificação é relevante porque, em ambientes distribuídos, a existência de logs de execução da função Lambda não garante, por si só, uma trilha forense completa sobre o acesso aos dados. Para investigar acessos indevidos, alterações não autorizadas ou consultas anômalas a dados sensíveis, é necessário que as operações realizadas sobre o banco também sejam registradas por mecanismos de auditoria. No caso do DynamoDB, operações como `GetItem`, `Query`, `Scan`, `PutItem` e `DeleteItem` dependem da configuração adequada de eventos de dados no CloudTrail para que possam ser auditadas de forma detalhada.

O procedimento consistiu na análise da configuração do CloudTrail na conta AWS utilizada pelo ambiente do *Pense Bem*. Inicialmente, foi verificada a existência de *trails* configurados. Em seguida, avaliou-se se havia cobertura de *data events* para recursos do DynamoDB. Foram consideradas como operações relevantes aquelas diretamente associadas ao acesso e à manipulação de dados da aplicação, incluindo `GetItem`, `Query`, `Scan`, `PutItem` e `DeleteItem`.

Os resultados demonstraram ausência de *trails* configurados na conta analisada. Embora o CloudTrail possa disponibilizar um histórico básico de eventos de gerenciamento no console da AWS, esse histórico não substitui a configuração de um *trail* para fins de auditoria persistente, retenção e investigação forense. Dessa forma, considerou-se que a conta não possuía cobertura formal de auditoria contínua para eventos de gerenciamento nem para eventos de dados do

DynamoDB.

A Figura 5.7 sintetiza esse resultado. Observa-se que não havia *trails* configurados e, por consequência, tanto os eventos de gerenciamento quanto as operações de dados do DynamoDB foram classificados como não auditados para fins de auditoria persistente.



Figura 5.7: Cobertura de auditoria do DynamoDB avaliada durante o Teste B3.

Quadro 5.11: Resultado do Teste B3: auditoria de operações no DynamoDB.

Item avaliado	Resultado
Serviço analisado	Amazon DynamoDB.
Técnica utilizada	Inspeção da configuração do AWS CloudTrail e dos <i>data events</i> .
Recurso de auditoria esperado	<i>Trail</i> do CloudTrail com eventos de dados habilitados para tabelas DynamoDB.
CloudTrail	Nenhum <i>trail</i> configurado na conta analisada.
Eventos de gerenciamento	Sem auditoria persistente por ausência de <i>trail</i> configurado.
Operações de dados avaliadas	GetItem, Query, Scan, PutItem e DeleteItem.
Cobertura observada	Ausente para operações de leitura, escrita, varredura e exclusão de itens.
Impacto principal	Ausência de trilha forense sobre acesso e manipulação de dados da aplicação.
Situação da vulnerabilidade	Confirmada como falha crítica de auditoria.

Esse resultado reforça a Vulnerabilidade B, pois demonstra que o ambiente não possui visibilidade suficiente sobre uma das principais camadas de persistência da aplicação. Além

disso, a ausência de trilhas de auditoria dificulta a comprovação de medidas de segurança e controle de acesso sobre dados pessoais, aspecto relevante no contexto da LGPD (BRASIL, 2018). Assim, o Teste B3 evidencia que a rastreabilidade do *Pense Bem* não depende apenas dos logs das funções Lambda, mas também da configuração adequada dos mecanismos de auditoria dos serviços consumidos pela aplicação.

5.6.4 Teste B4: Qualidade e Estruturação dos Logs

O quarto teste avaliou a qualidade dos logs produzidos pelas funções Lambda, verificando se os registros possuíam estrutura adequada para auditoria automatizada, rastreamento e análise de eventos de segurança. Após os testes anteriores evidenciarem ausência de logs no ponto de entrada, falta de identificadores de correlação e ausência de auditoria no DynamoDB, este teste analisou se os logs existentes nas funções Lambda apresentavam qualidade suficiente para apoiar investigações e detecção de incidentes.

O procedimento consistiu na análise de logs recentes das funções `dev-games-us-east-1` e `dev-questions-us-east-1`, avaliando três critérios: percentual de logs estruturados em JSON, presença de identificadores de requisição e presença de identificadores de usuário. Esses critérios foram selecionados porque logs estruturados e campos padronizados facilitam consultas no CloudWatch, criação de métricas, geração de alertas e correlação entre eventos.

Os resultados demonstraram baixa taxa de logs estruturados, ausência de padronização entre funções e presença limitada de identificadores de usuário, comprometendo a rastreabilidade automatizada. Na função `dev-games-us-east-1`, apenas 31,6% das linhas analisadas estavam estruturadas em JSON, enquanto na função `dev-questions-us-east-1` não foram identificados logs estruturados. Embora parte dos registros possuísse identificadores de requisição, a ausência de padronização e de identificadores de usuário reduz a capacidade de associar eventos técnicos a ações realizadas por usuários específicos.

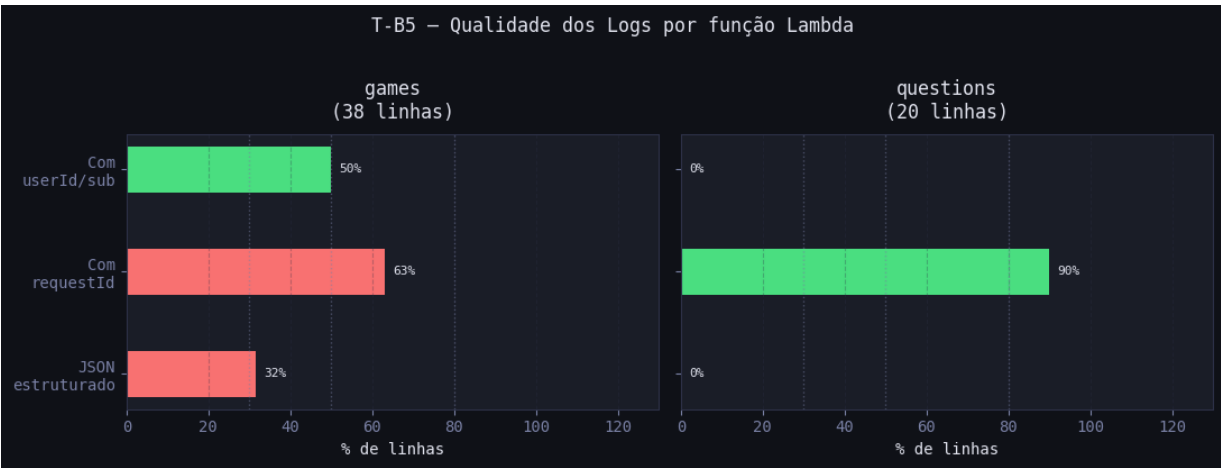


Figura 5.8: Avaliação da qualidade e estruturação dos logs das funções Lambda durante o Teste B4.

Quadro 5.12: Resultado do Teste B4: qualidade dos logs das funções Lambda.

Item avaliado	Resultado
Funções analisadas	dev-games-us-east-1 e dev-questions-us-east-1.
Técnica utilizada	Análise estrutural dos logs no CloudWatch.
Logs estruturados em JSON	32% na API games e 0% na API questions.
Logs com identificador de requisição	63% na API games e 90% na API questions.
Logs com identificador de usuário	50% dos logs JSON da API games e 0% na API questions.
Impacto principal	Baixa capacidade de auditoria automatizada, rastreamento de usuários e correlação de eventos.
Situação da vulnerabilidade	Confirmada como deficiência de qualidade e padronização dos logs.

Os resultados demonstram que os logs produzidos pelas funções Lambda não seguem um padrão consistente de estruturação e rastreabilidade. Essa limitação dificulta a utilização de ferramentas automatizadas de análise, detecção de incidentes e correlação de eventos. No contexto da Vulnerabilidade B, o achado reforça que a existência de logs isolados no CloudWatch não é suficiente: é necessário que esses registros sejam estruturados, padronizados e contenham campos mínimos de rastreamento para apoiar auditoria e resposta a incidentes.

5.7 Síntese dos Testes Executados

O Quadro 5.13 consolida os testes realizados, associando cada experimento à vulnerabilidade avaliada, às ferramentas utilizadas e às evidências analisadas.

Quadro 5.13: Matriz de testes por vulnerabilidade.

Código	Teste	Vulnerabilidade	Ferramenta	Evidência analisada
T-A1	Persistência de variáveis entre execuções.	A – Isolamento.	Lambda CloudWatch.	e Cabeçalhos de rastreamento, logs e respostas da API.
T-A2	Persistência de arquivos no diretório <code>/tmp</code> .	A – Isolamento.	Lambda CloudWatch.	e Contagem de arquivos residuais entre invocações.
T-A3	Stress de <i>warm start</i> com K6.	A – Isolamento.	K6 CloudWatch.	e <i>Warm starts</i> , <i>cold starts</i> , reuso de contêiner e latência p95.
T-A4	Permissões excessivas na role <code>games-api-role</code> .	A – Isolamento e controle de acesso.	AWS CLI e IAM Policy Simulator.	Ações IAM permitidas e negadas para a role da Lambda.
T-B1	Ausência de <i>access logs</i> e tracing no API Gateway.	B – Monitoramento.	AWS CLI.	Configuração de <i>stages</i> , <i>access logs</i> e X-Ray.
T-B2	Ausência de <i>correlation IDs</i> entre serviços.	B – Rastreabilidade.	CloudWatch Logs e AWS CLI.	Presença de <i>requestId</i> , <i>correlationId</i> e X-Ray Trace ID.
T-B3	Cobertura de auditoria no DynamoDB.	B – Auditoria.	AWS CLI e CloudTrail.	Existência de <i>trails</i> e <i>data events</i> para DynamoDB.
T-B4	Qualidade e estruturação dos logs.	B – Auditoria e rastreabilidade.	CloudWatch Logs e AWS CLI.	Percentual de logs JSON, <i>requestId</i> e identificador de usuário.

5.8 Discussão dos Resultados Experimentais

Os resultados obtidos demonstram que as vulnerabilidades identificadas na análise arquitetural do *Pense Bem* não se restringem a riscos conceituais, mas se manifestam como condições observáveis no ambiente testado. No caso da Vulnerabilidade A, os experimentos evidenciaram que a reutilização de ambientes Lambda ocorre tanto em chamadas sequenciais quanto em cenários de carga. Conforme descrito na documentação *Understanding the Lambda execution environment lifecycle*, a reutilização do ambiente de execução é um comportamento previsto do AWS Lambda e pode contribuir para a redução de latência em invocações subsequentes (Amazon Web Services, 2025e). Entretanto, a documentação *Best practices for working with AWS Lambda functions* recomenda que dados sensíveis ou específicos de usuários não sejam armazenados no ambiente reutilizável da função (Amazon Web Services, 2025b).

A persistência de variáveis, estados de módulo ou arquivos temporários não representa, por si só, uma falha do serviço AWS Lambda. O risco surge quando a aplicação assume incorretamente que cada invocação ocorre em um ambiente completamente novo ou quando armazena dados sensíveis fora do escopo da requisição. No contexto do *Pense Bem*, essa preocupação é relevante porque o sistema manipula dados associados ao desempenho de alunos, pontuações, recomendações e sessões de jogo.

Além disso, a análise de permissões IAM demonstrou que o impacto potencial de uma falha de isolamento pode ser ampliado quando a função executa com permissões superiores às necessárias. A presença de ações sensíveis permitidas para a role `games-api-role`, como exclusão de tabelas DynamoDB e alteração de políticas S3, evidencia que uma eventual função comprometida poderia afetar recursos críticos da aplicação. Dessa forma, a combinação entre reaproveitamento de ambiente, persistência de estado e permissões amplas transforma uma falha localizada em um risco mais abrangente para a confidencialidade, integridade e disponibilidade dos dados.

No caso da Vulnerabilidade B, os testes demonstraram que a rastreabilidade do ambiente é limitada. A ausência de *access logs* no API Gateway reduz a visibilidade sobre a origem, frequência e comportamento das requisições. A falta de *correlation IDs* dificulta a reconstrução

de fluxos distribuídos entre os serviços. A inexistência de auditoria persistente sobre operações no DynamoDB compromete a investigação de acessos e alterações em dados sensíveis. Por fim, a baixa qualidade estrutural dos logs limita a aplicação de consultas, métricas, alertas automatizados e correlação entre eventos.

Além dos impactos técnicos, esses achados também se relacionam aos requisitos de proteção de dados pessoais previstos na LGPD. No contexto do *Pense Bem*, dados como identificadores de usuários, pontuações, sessões, desempenho e recomendações podem estar associados a titulares de dados. Assim, a ausência de trilhas persistentes de auditoria, a baixa padronização dos logs, a falta de identificadores de correlação e a existência de permissões excessivas reduzem a capacidade de demonstrar controle sobre o tratamento dessas informações. Dessa forma, observabilidade, auditoria e rastreabilidade não representam apenas boas práticas técnicas, mas também elementos importantes para governança e conformidade em sistemas que tratam dados de usuários (BRASIL, 2018).

Esses achados estão alinhados às recomendações apresentadas no *AWS Well-Architected Framework: Security Pillar*, que destaca a importância de mecanismos de detecção, auditoria, rastreabilidade e resposta a eventos de segurança em ambientes de nuvem (Amazon Web Services, 2025a). Em arquiteturas *serverless*, essa necessidade torna-se ainda mais relevante, pois a execução da aplicação é distribuída entre serviços gerenciados e nem sempre os registros são centralizados ou correlacionados por padrão.

Observa-se, portanto, uma relação direta entre as duas vulnerabilidades avaliadas. A Vulnerabilidade A cria condições para persistência de estado, exposição de dados residuais e ampliação de impacto por permissões excessivas. A Vulnerabilidade B, por sua vez, reduz a capacidade de identificar, correlacionar e investigar esses comportamentos. Assim, as vulnerabilidades se reforçam mutuamente: quanto menor a visibilidade do ambiente, maior o risco de que falhas de isolamento, persistência indevida de dados ou permissões excessivas permaneçam sem detecção.

De forma geral, a validação experimental evidencia que a segurança em *serverless computing* depende não apenas dos mecanismos oferecidos pelo provedor de nuvem, mas também das decisões de implementação, configuração e observabilidade adotadas pela aplicação. No *Pense*

Bem, os resultados demonstram que algumas características comuns na computação sem servidor, como a reutilização de ambientes Lambda, podem ampliar riscos quando combinadas com persistência de estado, permissões amplas e baixa rastreabilidade. Assim, os testes realizados fornecem evidências práticas de como vulnerabilidades estruturais do modelo *serverless* podem se manifestar em um sistema real.

5.9 Contramedidas Indicativas

Embora o objetivo deste trabalho não tenha sido implementar correções na arquitetura, os resultados experimentais permitem estabelecer uma configuração segura de referência para comparação. Essa configuração representa o estado esperado de um ambiente alinhado a boas práticas de segurança para arquiteturas *serverless*, considerando isolamento, permissões, auditoria, logs e rastreabilidade.

Quadro 5.14: Comparação entre achados experimentais e configuração segura de referência.

Achado observado	Risco associado	Configuração segura esperada
Reuso de ambientes Lambda	Possível persistência de estado entre execuções.	Funções sem estado, sem armazenamento de dados sensíveis em variáveis globais ou fora do escopo da requisição.
Persistência de arquivos em <code>/tmp</code>	Exposição de resíduos temporários.	Remoção explícita de arquivos temporários ao final da execução e uso de nomes segregados por requisição.
Permissões IAM excessivas	Ampliação de impacto em caso de comprometimento.	Políticas IAM restritas às ações e recursos estritamente necessários.
Ausência de <i>access logs</i>	Baixa visibilidade no ponto de entrada da aplicação.	Logs de acesso habilitados no API Gateway com campos mínimos de auditoria.
Ausência de <i>correlation IDs</i>	Dificuldade de reconstrução do fluxo distribuído.	Identificador único propagado entre API Gateway, Lambda e demais serviços.
Ausência de <i>trails</i> do CloudTrail	Falta de auditoria persistente sobre operações de dados.	CloudTrail configurado com eventos de gerenciamento e eventos de dados para recursos críticos.
Logs pouco estruturados	Baixa capacidade de análise automatizada.	Logs em JSON com campos padronizados, como <i>timestamp</i> , <i>requestId</i> , <i>correlationId</i> , rota, status e usuário.

Como exemplo de contramedida para a Vulnerabilidade A, funções Lambda que utilizam o diretório `/tmp` devem remover explicitamente arquivos temporários ao final do processamento. Essa remoção pode ser realizada em uma etapa final da execução da função, garantindo que arquivos temporários associados a uma requisição não permaneçam disponíveis em invocações posteriores. Além disso, recomenda-se evitar o armazenamento de dados sensíveis nesse diretório e utilizar nomes de arquivos segregados por requisição, reduzindo o risco de exposição de resíduos temporários em cenários de *warm start*.

Também como medida relacionada à Vulnerabilidade A, dados específicos de usuário, como identificadores, pontuações, sessões ou recomendações, não devem ser armazenados em variáveis globais ou em estados mantidos fora do escopo do manipulador principal da função. Cada invo-

cação deve tratar os dados apenas no contexto da requisição corrente, evitando que informações de uma execução anterior permaneçam acessíveis em chamadas subsequentes.

No caso das permissões IAM, a política associada às funções Lambda deve ser limitada às ações estritamente necessárias para a execução da lógica da aplicação. Em vez de permitir ações amplas ou destrutivas, como `dynamodb:DeleteTable`, `s3:DeleteBucket` ou `s3:PutBucketPolicy`, recomenda-se restringir o acesso a operações específicas, como leitura, escrita ou atualização nos recursos efetivamente utilizados. Essa prática reduz o impacto potencial em caso de comprometimento de uma função ou uso indevido de suas credenciais temporárias.

Para a Vulnerabilidade B, uma contramedida relevante consiste na padronização dos logs das funções Lambda em formato estruturado, preferencialmente JSON. Esses registros devem incluir campos mínimos como *timestamp*, *requestId*, *correlationId*, identificador do usuário, rota acessada, status da operação e mensagem do evento. Com isso, torna-se possível realizar consultas automatizadas, criar métricas, configurar alertas e reconstruir o fluxo de uma requisição entre diferentes componentes da arquitetura.

Outra medida importante consiste na propagação de identificadores de correlação entre os serviços envolvidos no fluxo da aplicação. Um identificador único pode ser gerado no ponto de entrada da requisição e repassado entre API Gateway, funções Lambda e demais serviços utilizados. Essa prática facilita a rastreabilidade ponta a ponta e melhora a capacidade de investigação em caso de falhas, comportamentos anômalos ou incidentes de segurança.

Por fim, recomenda-se habilitar mecanismos de auditoria e monitoramento compatíveis com a criticidade dos recursos analisados. Isso inclui a configuração de *access logs* no Amazon API Gateway, a ativação de *trails* no AWS CloudTrail para eventos de gerenciamento e eventos de dados em recursos críticos, como tabelas DynamoDB, e a utilização de recursos de rastreamento distribuído quando aplicável. Essas medidas ampliam a visibilidade operacional e fortalecem a capacidade de auditoria do ambiente.

Essas contramedidas não eliminam todos os riscos associados ao modelo *serverless*, mas reduzem a probabilidade de exposição de dados residuais, diminuem o impacto de uma função comprometida e aumentam a capacidade de auditoria, correlação e investigação. Dessa forma,

a configuração segura de referência proposta permite comparar o estado observado nos testes com um estado esperado de maior maturidade em segurança.

Capítulo 6

Resultados e Considerações Finais

*Este capítulo apresenta a síntese dos resultados alcançados ao longo do trabalho, considerando a análise arquitetural e a validação experimental das vulnerabilidades identificadas no sistema *Pense Bem*. São discutidas as principais evidências obtidas, as limitações do estudo, possíveis direcionamentos para trabalhos futuros e as considerações finais da pesquisa.*

6.1 Síntese dos Resultados Obtidos

O presente trabalho teve como objetivo analisar a segurança de uma aplicação baseada em computação *serverless*, utilizando o sistema educacional *Pense Bem* como estudo de caso. A partir da revisão teórica, do mapeamento da arquitetura e da execução de testes experimentais, foi possível compreender como vulnerabilidades estruturais desse modelo podem se manifestar em um ambiente real, especialmente em fluxos que envolvem AWS Lambda, Amazon API Gateway, Amazon DynamoDB, Amazon S3, Amazon CloudWatch, AWS CloudTrail, AWS X-Ray e permissões IAM.

A análise arquitetural permitiu identificar os principais ativos críticos do sistema, incluindo funções Lambda, endpoints expostos pelo API Gateway, dados armazenados no DynamoDB, permissões IAM, mecanismos de monitoramento e fluxos de comunicação entre serviços. A partir desse mapeamento, foram priorizadas duas vulnerabilidades centrais: a Vulnerabilidade A, relacionada a falhas de isolamento e persistência de estado em ambientes Lambda reutilizados; e a Vulnerabilidade B, associada a falhas de monitoramento, auditoria e rastreabilidade

distribuída.

A validação experimental confirmou a relevância dessas vulnerabilidades no contexto do *Pense Bem*. Em relação à Vulnerabilidade A, os testes demonstraram que ambientes Lambda podem ser reutilizados entre invocações sucessivas, que arquivos temporários podem permanecer no diretório `/tmp`, que o reuso de contêineres ocorre de forma recorrente sob carga e que a role `games-api-role` possui permissões além do necessário para a operação esperada da função. Esses resultados demonstram que características operacionais legítimas do AWS Lambda, quando combinadas com práticas inadequadas de gerenciamento de estado e permissões excessivas, podem ampliar a superfície de exposição da aplicação.

No caso da Vulnerabilidade B, os testes evidenciaram limitações relevantes de observabilidade. Foram identificadas ausência de *access logs* e *tracing* no API Gateway, ausência de *correlation IDs* entre serviços, inexistência de *trails* do CloudTrail para auditoria persistente de operações no DynamoDB e baixa qualidade estrutural dos logs produzidos pelas funções Lambda. Esses achados demonstram que a existência de logs isolados não é suficiente para garantir rastreabilidade. Em arquiteturas distribuídas, é necessário que os registros sejam estruturados, padronizados e correlacionáveis.

De forma geral, os resultados obtidos confirmam que a segurança em arquiteturas *serverless* depende não apenas dos mecanismos oferecidos pelo provedor de nuvem, mas também de decisões adequadas de implementação, configuração e observabilidade. No contexto do *Pense Bem*, as vulnerabilidades analisadas afetam diretamente a confidencialidade, a integridade, a disponibilidade e a capacidade de investigação de incidentes envolvendo dados de desempenho, pontuações, recomendações e sessões de usuários.

Assim, o trabalho permitiu conectar a fundamentação teórica sobre segurança em *serverless computing* a evidências práticas observadas em um sistema real. A análise demonstrou que riscos amplamente documentados na literatura, como persistência de estado em ambientes reutilizados, permissões excessivas e monitoramento insuficiente, podem ser verificados empiricamente em aplicações que utilizam serviços gerenciados e execução orientada a eventos.

6.2 Limitações do Estudo

Este trabalho foi delimitado à análise de segurança do *back-end* do sistema *Pense Bem*, por ser a camada em que se concentram as funções Lambda, integrações com serviços gerenciados, permissões IAM, mecanismos de auditoria e registros de execução. Dessa forma, aspectos relacionados ao *front-end*, à experiência do usuário e à lógica pedagógica interna do modelo de recomendação não foram avaliados em profundidade.

A validação experimental também foi delimitada a um conjunto específico de rotas e recursos da aplicação. Essa seleção foi orientada pela relevância desses elementos para as vulnerabilidades analisadas, especialmente nos fluxos relacionados a isolamento de execução, permissões IAM, monitoramento, auditoria e rastreabilidade. Dessa forma, os resultados não devem ser interpretados como uma auditoria completa de segurança de todos os componentes do sistema, mas como uma validação direcionada dos pontos de exposição priorizados neste trabalho.

Além disso, alguns testes exigiram instrumentação temporária das funções, como a inclusão de cabeçalhos para identificação de contêineres e reutilização de ambiente. Essa abordagem foi adequada para fins experimentais, mas não representa necessariamente o comportamento padrão da aplicação sem instrumentação. Ainda assim, os resultados obtidos são relevantes, pois evidenciam condições operacionais que podem ampliar riscos quando não são adotadas práticas seguras de desenvolvimento.

Por fim, a análise de monitoramento e auditoria considerou as configurações observadas no ambiente durante o período de execução dos testes. Mudanças posteriores na infraestrutura, nas permissões IAM ou nos mecanismos de *logging* podem alterar parte dos resultados. Dessa forma, os achados devem ser compreendidos como um retrato do ambiente analisado no momento da validação experimental.

Outra limitação refere-se ao inventário completo de endpoints e controles de autorização. Embora as rotas selecionadas tenham sido documentadas e analisadas em função das vulnerabilidades priorizadas, este trabalho não realizou uma verificação exaustiva de todas as APIs públicas, privadas ou protegidas por token existentes no sistema. Assim, a identificação completa de rotas sem autorização, testes de bypass de autenticação e análise detalhada de políticas

de autorização por endpoint permanecem como possibilidades de aprofundamento futuro.

6.3 Trabalhos Futuros

Como continuidade deste trabalho, estudos futuros podem ampliar a análise para outros componentes do sistema *Pense Bem*, incluindo o *front-end*, os fluxos de autenticação, o modelo de recomendação e demais serviços de apoio utilizados pela aplicação. Essa ampliação permitiria uma visão mais abrangente da superfície de ataque e dos riscos associados ao sistema como um todo.

Outra possibilidade de continuidade consiste na aplicação prática de contramedidas sobre a arquitetura analisada, seguida de uma nova etapa de validação experimental. Nesse cenário, seria possível comparar o comportamento do ambiente antes e depois da adoção de controles voltados ao gerenciamento de estado, revisão de permissões IAM, estruturação de logs, auditoria de eventos e rastreamento distribuído.

Recomenda-se também que trabalhos futuros explorem mecanismos automatizados de verificação contínua, como análise periódica de permissões IAM, validação da cobertura de CloudTrail, detecção de ausência de *access logs*, verificação de padrões mínimos de logs estruturados e monitoramento de anomalias em rotas críticas. Essa continuidade permitiria avaliar a evolução da postura de segurança do sistema ao longo do tempo, reduzindo a dependência de análises manuais e pontuais.

Por fim, pesquisas futuras podem ampliar a investigação sobre segurança em *serverless computing* a partir de estudos aplicados em produtos reais. Embora a literatura apresente modelos, taxonomias e categorias de vulnerabilidades relevantes, a análise prática de sistemas em produção ou em ambientes próximos ao real permite observar como essas fragilidades se manifestam em decisões de arquitetura, configurações de serviços gerenciados, permissões IAM, fluxos de dados e mecanismos de monitoramento. Dessa forma, novos estudos podem aproximar a discussão acadêmica dos desafios enfrentados no desenvolvimento e operação de aplicações *serverless*.

6.4 Considerações Finais

A investigação realizada demonstrou que a adoção do modelo *serverless*, embora ofereça benefícios relevantes de escalabilidade, abstração operacional e eficiência de custos, também introduz desafios específicos de segurança. No caso do *Pense Bem*, esses desafios se manifestaram principalmente na reutilização de ambientes Lambda, na persistência de dados temporários, na existência de permissões IAM excessivas e na ausência de rastreabilidade adequada entre serviços distribuídos. Os experimentos realizados confirmaram que as vulnerabilidades analisadas não são apenas possibilidades teóricas, mas condições observáveis no ambiente estudado. A Vulnerabilidade A evidenciou que práticas inadequadas de gerenciamento de estado podem comprometer a confidencialidade e a integridade dos dados. A Vulnerabilidade B demonstrou que a ausência de logs estruturados, auditoria persistente e identificadores de correlação dificulta a detecção e investigação de comportamentos anômalos.

Também foi possível observar que as duas vulnerabilidades se reforçam mutuamente. Falhas de isolamento ou permissões excessivas tornam-se mais graves quando não existem mecanismos suficientes para detectar, rastrear e investigar sua ocorrência. Assim, a segurança do sistema depende de uma abordagem integrada, que combine boas práticas de desenvolvimento, controle adequado de permissões e mecanismos consistentes de observabilidade.

Dessa forma, a análise desenvolvida ao longo do trabalho evidencia a relevância de avaliar a segurança de aplicações *serverless* a partir de cenários reais, nos quais decisões de implementação, configuração de serviços gerenciados e mecanismos de observabilidade influenciam diretamente a exposição do sistema. No contexto do *Pense Bem*, os resultados obtidos oferecem uma base concreta para compreender os desafios de segurança desse modelo e orientar futuras evoluções da arquitetura analisada.

Referências Bibliográficas

Amazon Web Services. *AWS Well-Architected Framework: Serverless Applications Lens*. 2023. <<https://docs.aws.amazon.com/wellarchitected/latest/serverless-applications-lens/welcome.html>>. Acesso em: 15 nov. 2025.

Amazon Web Services. *AWS Well-Architected Framework: Security Pillar*. 2025. <<https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html>>. Acesso em: 01 dez. 2025.

Amazon Web Services. *Best practices for working with AWS Lambda functions*. 2025. <<https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html>>. Acesso em: 01 dez. 2025.

Amazon Web Services. *Lambda Execution Environment and Cold Starts*. 2025. Acesso em: 20 nov. 2025. Disponível em: <<https://docs.aws.amazon.com/lambda/latest/operatorguide/execution-environments.html>>.

Amazon Web Services. *Trace user requests to REST APIs using X-Ray in API Gateway*. 2025. <<https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-xray.html>>. Acesso em: 01 dez. 2025.

Amazon Web Services. *Understanding the Lambda execution environment lifecycle*. 2025. <<https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtime-environment.html>>. Acesso em: 01 dez. 2025.

Amazon Web Services. *Variables for access logging for API Gateway*. 2025. <<https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-variables-for-access-logging.html>>. Acesso em: 01 dez. 2025.

BRASIL. *Lei Geral de Proteção de Dados Pessoais (LGPD) — Lei nº 13.709, de 14 de agosto de 2018*. 2018. <https://www.planalto.gov.br/ccivil_03/_ato2015-2018/2018/lei/113709.htm>. Acesso em: 01 dez. 2025.

Center for Internet Security. *CIS Amazon Web Services Benchmarks*. 2026. Disponível em: <https://www.cisecurity.org/benchmark/amazon_web_services>. Acesso em: 20 jun. 2026.

CHANDRAMOULI, R. *Security Strategies for Microservices-based Application Systems*. [S.l.], 2019. Disponível em: <<https://csrc.nist.gov/pubs/sp/800/204/final>>. Acesso em: 20 jun. 2026.

Datadog. *State of Serverless 2023*. 2023. <<https://www.datadoghq.com/state-of-serverless/>>. Acesso em: 15 nov. 2025.

DORSETT, M. et al. A comprehensive review of denial of wallet attacks in serverless architectures. *arXiv preprint arXiv:2508.19284*, 2025. Acesso em: 20 jun. 2026. Disponível em: <<https://arxiv.org/abs/2508.19284>>.

GOOGLE CLOUD. *Tipos de serviços de nuvem: IaaS x PaaS x SaaS x modelos sem servidor*. 2025. Acesso em: 10 nov. 2025. Disponível em: <<https://cloud.google.com/discover/types-of-cloud-computing?hl=pt-BR>>.

IBM. *O que é FaaS?* 2025. Acesso em: 10 nov. 2025. Disponível em: <<https://www.ibm.com/br-pt/think/topics/faas>>.

IBM Brasil. *Relatório da IBM: custo médio de uma violação de dados no Brasil atinge R\$ 7,19 milhões*. 2025. <<https://brasil.newsroom.ibm.com/2025-07-30-Relatorio-da-IBM-Custo-medio-de-uma-violacao-de-dados-no-Brasil-atinge-R-7,19-milhoes>>. Publicação institucional da IBM Brasil. Acesso em: 01 dez. 2025.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. *Information technology — Security techniques — Code of practice for information security controls based on ISO/IEC 27002 for cloud services*. 2015.

LI, X.; LENG, X.; CHEN, Y. Securing serverless computing: Challenges, solutions, and opportunities. *arXiv preprint arXiv:2105.12581*, 2021. Disponível em: <<https://arxiv.org/abs/2105.12581>>.

MARIN, E.; PERINO, D.; PIETRO, R. D. Serverless computing: A security perspective. *arXiv preprint arXiv:2107.03832*, 2021. Disponível em: <<https://arxiv.org/abs/2107.03832>>.

MITRE ATT&CK. *Matrix: Enterprise – Cloud*. 2026. Disponível em: <<https://attack.mitre.org/matrices/enterprise/cloud/>>. Acesso em: 20 jun. 2026.

OWASP Foundation. *OWASP Serverless Top 10 Project*. 2023. <<https://owasp.org/www-project-serverless-top-10/>>. Acesso em: 15 nov. 2025.

Sysdig. *Identity and Access Management in the Cloud is Hard*. 2023. <<https://sysdig.com/blog/identity-access-management-difficult-cloud/>>. Acesso em: 20 nov. 2025.

TAURION, C. *Cloud Computing: Computação em Nuvem – Transformando o Mundo da Tecnologia da Informação*. [S.l.]: Brasport, 2009.