Sistema de Manutenção Proativa e Resposta a Incidentes de Segurança

Diogo Nunes[1] and F. Jorge Duarte[1]

ISEP - IPP, Porto 1211895,fjd@isep.ipp.pt

Abstract. Este artigo descreve o desenho, a implementação e a avaliação de uma plataforma de monitorização e resposta automática a vulnerabilidades e incidentes operacionais. A solução colmata a ausência de mecanismos automatizados e auditáveis para (i) publicação de páginas de manutenção, (ii) encerramento controlado da aplicação e (iii) notificação estruturada perante quebras de segurança ou picos de utilização. Integra uma *pipeline* CI/CD (Continuous Integration and Continuous Deployment) que, a cada duas horas, analisa dependências NuGet e npm, normaliza resultados e publica-os no Azure Log Analytics; consoante a severidade, uma Azure Logic App notifica a equipa e pode acionar o encerramento do Azure App Service. No perímetro, o Azure Front Door, protegido por WAF (Web Application Firewall), gere o encaminhamento entre a aplicação e páginas de manutenção alojadas em Azure Storage. Em complemento, o Azure Application Insights e o Azure Monitor recolhem telemetria e métricas de CPU/RAM, originando alertas que garantem visibilidade operacional contínua. Na avaliação, verificou-se que os *health probes* a cada 2 s suportaram *failover* observado de ∼15 s, com limite alvo inferior a 30 s, os ataques simulados foram bloqueados (HTTP 403) e os relatórios de vulnerabilidades foram ingeridos e distribuídos com fiabilidade. A abordagem demonstra viabilidade de resposta proativa, escalável e alinhada com práticas DevSecOps (Development, Security and Operations) em ambientes *cloud-native*.

Keywords: Monitorização · Observabilidade · Resposta automática a incidentes · DevSecOps · Azure Front Door · Azure Monitor

1 Introdução

A digitalização acelerada e a centralização de serviços web elevaram os requisitos de disponibilidade, segurança e observabilidade. Qualquer falha ou ataque pode traduzir-se em impactos operacionais, legais e reputacionais. É, por isso, essencial monitorizar, antecipar e reagir a comportamentos inesperados em produção com mecanismos automatizados e auditáveis.

Apresenta-se o desenho e a implementação de um sistema de manutenção proativa e resposta automatizada a incidentes de segurança em aplicações web. A solução assenta em serviços Microsoft Azure e práticas DevSecOps (Development, Security and Operations). A validação foi feita com uma prova de conceito composta por *backend* em ASP.NET Core e *frontend* em Angular, o que permitiu recolher telemetria real e exercitar cenários de falha e de ataque.

1.1 Descrição do Problema

Não existiam, no contexto que motivou este trabalho, mecanismos integrados que assegurassem, de forma autónoma e auditável, a deteção atempada de anomalias e vulnerabilidades, o encerramento controlado da aplicação quando necessário, a publicação de páginas de manutenção (planeada e de emergência) e a notificação estruturada da equipa técnica. Pretendia-se reduzir MTTD (Mean Time to Detection) e MTTR (Mean Time to Resolution), diminuir intervenção manual em momentos críticos e preservar a experiência do utilizador.

1.2 Objetivos

Apresentam-se de seguida os principais objetivos do trabalho:

- Integrar Application Insights e Azure Monitor para recolha de telemetria e análise contínua de fiabilidade e desempenho.
- Configurar alertas acionáveis (recursos e segurança), com notificações automáticas.
- Operar páginas de manutenção (planeada/emergência) com ativação controlada e comunicação clara.
- Implementar respostas automáticas (Logic Apps/Functions), incluindo encerramento seguro da aplicação perante vulnerabilidades críticas.
- Validar a solução com testes funcionais, de carga e cenários de ataque.

2 Estado da Arte

A solução proposta insere-se na confluência de observabilidade em ambientes *cloud-native*, automatização da resposta a incidentes e integração de práticas DevSecOps nas *pipelines* de desenvolvimento e operação. Esta secção sintetiza os principais trabalhos e tecnologias relevantes que enquadram o desenho adotado.

2.1 Trabalhos Relacionados

A investigação recente em operações em *cloud* tem procurado aumentar a visibilidade ponta a ponta, detetar anomalias mais cedo e reduzir os tempos de deteção e resolução de incidentes. A linha comum passa por instrumentação consistente, modelos de deteção cada vez mais robustos e automação criteriosa da resposta [1,2,5].

Na observabilidade, a correlação de *logs*, *traces* e *metrics* tornou-se prática corrente, suportada por instrumentação aberta como OpenTelemetry e por abordagens de análise comportamental UEBA (User and Entity Behavior Analytics). Conjugam-se técnicas estatísticas com modelos de aprendizagem para identificar desvios [3]. Continuam, porém, desafios de calibração e de controlo do ruído de alertas [2].

Em AIOps (Artificial Intelligence for IT Operations), a aplicação de métodos de ML (Machine Learning) e NLP (Natural Language Processing) ajuda a

correlacionar eventos, reduzir redundâncias e priorizar alertas, com ganhos reportados na redução de MTTD/MTTR [1,2]. Ganham relevância modelos preditivos para antecipar incidentes, mas persistem necessidades de explicabilidade e adaptação contínua a contextos dinâmicos [2].

Na resposta automatizada a incidentes SOAR (Security Orchestration, Automation and Response), generalizou-se o uso de *playbooks* para tarefas repetitivas de isolamento, bloqueio e recolha de evidências. A prática dominante é a automação assistida: automatizar o que é rotineiro, mantendo aprovação humana quando as ações são disruptivas, assegurando auditoria e rastreabilidade [5].

Quanto à resiliência e manutenção, destacam-se o recurso a WAF, *health probes* frequentes e *failover* para páginas estáticas fora da aplicação, bem como padrões ao nível da aplicação, como *circuit breakers*, *fallbacks* e *feature flags* [7]. A eficácia depende de limiares e temporizações bem afinados para evitar oscilações e proteger a experiência do utilizador.

Por fim, a integração de práticas DevSecOps nas *pipelines* CI/CD tornou-se transversal, combinando SAST (Static Application Security Testing), DAST (Dynamic Application Security Testing) e SCA (Software Composition Analysis) com análises de IaC (Infrastructure as Code), SBOMs (Software Bill of Materials) e PaC (Policy as Code) para reforçar a governança. Modelos de maturidade apoiam uma adoção faseada, especialmente relevante em organizações com recursos limitados [6].

2.2 Tecnologias Existentes

As três principais plataformas *cloud-native*, Microsoft Azure, Amazon Web Services (AWS) e Google Cloud Platform (GCP), oferecem capacidades convergentes para observabilidade, segurança e automação, com diferenças de integração e governança [8].

Na observabilidade e monitorização, o Azure integra *logs*, *metrics*, alertas e *tracing* distribuído em Monitor e Application Insights. Inclui KQL (Kusto Query Language) e Smart Detection. A AWS combina CloudWatch, X-Ray e CloudTrail numa abordagem modular. O GCP agrega Cloud Monitoring, Cloud Logging e Cloud Trace, com suporte nativo a OpenTelemetry [4,8].

Na gestão de páginas de manutenção, no Azure é possível usar Deployment Slots do App Service e Traffic Manager. Também é comum servir páginas estáticas via Storage integrado com Front Door ou Application Gateway. Na AWS, a combinação típica é S3 com CloudFront, podendo existir resposta direta via ALB. No GCP, usa-se Cloud Storage com Cloud Load Balancer ou uma implementação dedicada no Cloud Run.

Na automação CI/CD e orquestração, o Azure disponibiliza Azure DevOps e GitHub Actions para *pipelines as code* e serviços *serverless* como Logic Apps e Functions. Na AWS, o conjunto inclui CodeCommit, CodeBuild, CodePipeline e CodeDeploy. No GCP, Cloud Build e Cloud Deploy integram-se de forma estreita com GKE e Cloud Run.

2.3 Síntese das Escolhas Tecnológicas

A opção por **Microsoft Azure** reflete o alinhamento com a infraestrutura existente da organização (políticas de segurança e pipelines já definidos) e a integração nativa necessária para observabilidade e resposta automatizada (Monitor, Application Insights, Log Analytics, Front Door/WAF, Logic Apps). Ao nível da *stack*, adotou-se **ASP.NET Core** (*backend*), **Angular** (*frontend*) e **SQL Server** (dados), privilegiando documentação, integração com o ecossistema e produtividade.

3 Análise e Desenho da Solução

Esta secção apresenta o enquadramento do problema, a definição dos requisitos e o desenho arquitetural que suporta monitorização, segurança e continuidade de serviço na aplicação.

3.1 Domínio do Problema

A organização opera uma aplicação web em Microsoft Azure com requisitos elevados de disponibilidade e segurança. É necessário detetar e responder a incidentes com rapidez, bloquear tráfego malicioso no perímetro e mitigar o risco contínuo de vulnerabilidades em dependências NuGet/npm. O problema a resolver consiste em garantir visibilidade ponta a ponta e resposta automática em produção: filtrar e encaminhar pedidos no perímetro, monitorizar aplicação e infraestrutura com alertas de baixo ruído, gerir continuidade através de páginas de manutenção com comutação rápida e automatizar a triagem de vulnerabilidades com ação preventiva quando crítico. O objetivo é reduzir MTTD e MTTR e preservar a experiência do utilizador mesmo perante falhas ou ameaças.

3.2 Engenharia de Requisitos

O levantamento de requisitos foi estruturado de acordo com o modelo FURPS+ [9]. Esta abordagem permite sintetizar de forma clara os objetivos operacionais e não funcionais que a solução deve cumprir, conforme se apresenta de seguida:

– **Funcionalidade (F)/Requisitos Funcionais (RF): RF1** bloquear automaticamente pedidos maliciosos com base em regras WAF; **RF2** alertar a equipa quando um pedido malicioso é bloqueado; **RF3** executar periodicamente a *pipeline* que analisa vulnerabilidades no *backend* e *frontend*; **RF4** registar todas as vulnerabilidades detetadas para auditoria; **RF5** notificar por alerta e email quando forem detetadas vulnerabilidades; **RF6** desligar automaticamente a aplicação quando existirem vulnerabilidades críticas no *backend*; **RF7** notificar com justificação técnica sempre que a aplicação for desativada; **RF8** redirecionar para página de manutenção de emergência quando a aplicação está indisponível; **RF9** Ativação manual de manutenção

planeada com definição de IPs autorizados; **RF10** assegurar acesso normal quando não houver manutenção ativa; **RF11** alertar e registar eventos quando limites de CPU/RAM forem ultrapassados; **RF12** detetar anomalias via Application Insights e alertar a equipa; **RF13** registar pedidos maliciosos bloqueados com detalhes para auditoria.
- **Usabilidade (U): U1** mensagens claras nas páginas de manutenção; **U2** relatórios legíveis por email.
- **Fiabilidade (R): R1** redirecionar automaticamente para página de emergência em falha.
- **Desempenho (P): P1** *failover* entre aplicação e emergência $< 30\,\text{s}$.
- **Suporte/Manutenção (S): S1** passos para ativar manutenção planeada claramente documentados.
- **Segurança (X): X1** apenas IPs autorizados acedem em manutenção; **X2** infraestrutura isolada (VNet + Private Endpoints); **X3** vulnerabilidades registadas centralmente; **X4** pedidos maliciosos bloqueados antes da aplicação.
- **Restrições (C): C1** infraestrutura integralmente em Microsoft Azure.

3.3 Arquitetura da Solução (modelo C4)

Adotou-se o modelo C4 [10] para a documentação arquitetural. Esta secção apresenta a vista lógica do Nível 2 (Contentores), ilustrada na Fig. 1, evidenciando o percurso do tráfego e a separação de responsabilidades. No perímetro, o Azure Front Door com WAF filtra e encaminha pedidos. A aplicação (Web App) encontra-se alojada em App Service e persiste dados em Azure SQL Server Database. As páginas de manutenção *Scheduled Maintenance Page* e *Emergency Maintenance Page*, residem em Storage Accounts e suportam *failover*/*fallback* rápidos. A observabilidade é assegurada por Application Insights, Azure Monitor e Log Analytics. A *pipeline* de CI/CD (Azure Pipelines) integra-se com o App Service e com o Log Analytics, automatizando a análise de vulnerabilidades e a disponibilização de *logs*. As respostas automáticas (notificação e, quando aplicável, encerramento controlado da aplicação) são orquestradas por uma Azure Logic App.

Alternativas consideradas

- **Páginas de manutenção no *frontend***: solução simples, mas acopla o *failover* à própria aplicação e exigiria novos *deploys* para ajustes.
- **Separar *frontend* e *backend* em App Services distintos**: aumentaria a escalabilidade e isolamento, mas também a complexidade operacional.
- **Application Gateway vs Front Door**: optou-se por Front Door pelo encaminhamento global, gestão centralizada de regras e facilidade na aplicação de listas de permissões em manutenção.

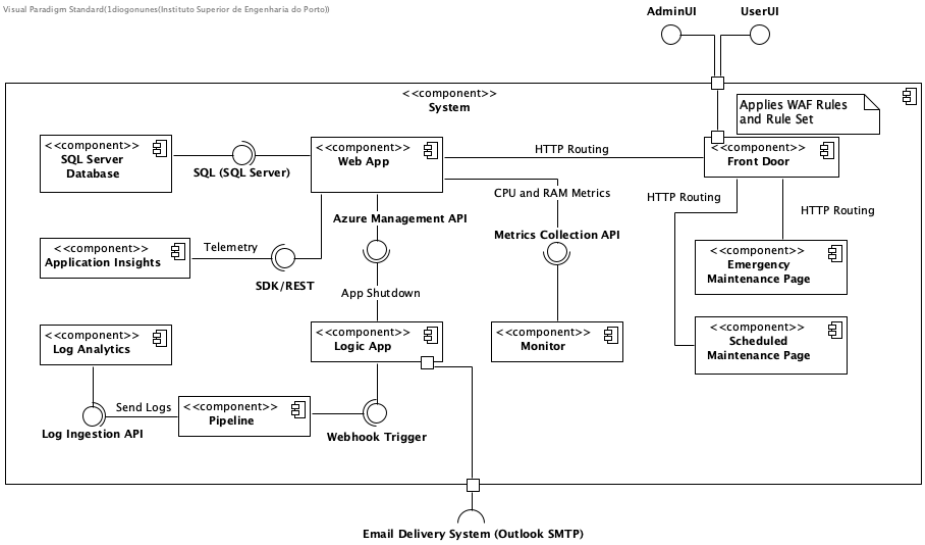

Fig. 1. Diagrama de vista lógica — Nível 2.

4 Implementação da Solução

Com base no desenho arquitetural, esta secção detalha a concretização da solução na plataforma Microsoft Azure, incluindo topologia arquitetural, *pipeline* de vulnerabilidades e mecanismos de monitorização e manutenção.

4.1 Descrição da Implementação

Topologia e Recursos da Infraestrutura A solução foi implementada na Microsoft Azure com isolamento de rede em Virtual Network e comunicação privada entre serviços por Private Endpoints. O ponto de entrada é o Azure Front Door Premium com WAF, responsável por filtragem e encaminhamento global. A aplicação (Angular no *frontend* e ASP.NET Core no *backend* é alojada em App Service e persiste dados em Azure SQL Database com acesso privado. As páginas de manutenção (planeada e de emergência) são servidas a partir de Azure Storage Accounts. A observabilidade integra Application Insights, Azure Monitor e Log Analytics, garantindo telemetria e alertas centralizados.

Pipeline de Análise e Resposta a Vulnerabilidades A *pipeline* CI/CD (Azure Pipelines) corre de 2 em 2 horas, auditando dependências do *backend* (NuGet) e do *frontend* (npm), consolidando resultados e enviando-os para Log Analytics. Consoante a gravidade das vulnerabilidades, desencadeia notificações e, quando aplicável, encerramento controlado do serviço.

Stage 1 — Análise de vulnerabilidades no *backend*. Nesta fase é feita uma auditoria periódica a pacotes .NET, produzindo um relatório JSON com a lista de vulnerabilidades e respetivas severidades. A Fig. 2 ilustra, de forma sintetizada, as etapas principais desta fase da *pipeline*.

Fig. 2. Etapas principais da Stage 1 da *pipeline* de vulnerabilidades (*backend*).

Stage 2 — Análise de vulnerabilidades no *frontend*. Nesta fase é realizada a auditoria a pacotes npm, gerando um relatório JSON que inclui apenas vulnerabilidades com severidade *High* ou superior. A Fig. 3 apresenta as etapas principais desta fase da *pipeline*.

Fig. 3. Etapas principais da Stage 2 da *pipeline* de vulnerabilidades (*frontend*).

Stage 3 — Execução do `log-vulnerabilities.js`. Nesta fase é executado o *script* de consolidação e encaminhamento: normaliza campos, agrega resultados e publica no Log Analytics, acionando também a Logic App com o relatório consolidado. A *pipeline* termina em `failed` sempre que existam vulnerabilidades detetadas (todas no *backend*; $\geq$ High no *frontend*), forçando triagem. A Fig. 4 ilustra as etapas principais desta fase da *pipeline*.

Fig. 4. Etapas principais da Stage 3 da *pipeline* de vulnerabilidades (consolidação e resposta).

Gestão de segredos na *pipeline*. As credenciais e parâmetros sensíveis são mantidos como *secrets* e injetados em tempo de execução como variáveis de ambiente, com permissões mínimas via RBAC (Role-Based Access Control). Os *secrets* são `AZURE_CLIENT_ID`, `AZURE_CLIENT_SECRET`, `AZURE_TENANT_ID`, `LOGS_INGESTION_ENDPOINT`, `DATA_COLLECTION_RULE_ID`, `STREAM_NAME` e `LOGICAPP_CALLBACK_URL`.

Funcionamento interno do `log-vulnerabilities.js`. O *script* lê os relatórios NuGet e npm, uniformiza campos (pacote, versão, severidade, origem, *timestamp*), aplica um mapeamento comum de severidades, remove duplicados e enriquece registos com metadados de execução (*pipeline*, *stage*, *commit*). Em seguida, produz *payloads* JSON para ingestão analítica e *payload* de notificação para a Logic App.

Ingestão de logs no Log Analytics. A ingestão é feita via Logs Ingestion API, através de um Data Collection Endpoint (DCE) e de uma Data Collection Rule (DCR) que mapeiam os registos para a tabela personalizada `Vulnerabilities_CL`. Falhas temporárias originam novas tentativas. Os dados ficam imediatamente disponíveis para KQL e para as regras de alerta.

Alertas baseados em *logs* de vulnerabilidades. A regra "PESTI Pipeline Vulnerability Scan Detected" avalia a tabela `Vulnerabilities_CL` numa janela

móvel de 15 min e dispara quando existe pelo menos um registo novo, independentemente da severidade ou da origem (NuGet/npm). Emite notificação no portal e por email através do grupo de ação.

Alerta de vulnerabilidades críticas no *backend*. A regra "PESTI Critical NuGet Vulnerabilities Detected" filtra a `Vulnerabilities_CL` por origem NuGet e severidade `CRITICAL` na mesma janela de 15 min. Para além da notificação no portal e por email, invoca a Logic App para resposta automatizada.

Resposta automatizada via Logic App. A Logic App recebe o relatório consolidado, transforma-o em tabela HTML e envia *email* detalhado. Se forem detetadas vulnerabilidades *Critical* no *backend*, executa o encerramento controlado da aplicação, enviando um segundo *email* com justificação e mantendo registos para auditoria.

Gestão de Pedidos e Mecanismos de Manutenção O Front Door aplica primeiro as regras do WAF (bloqueio a 403 de tráfego malicioso) e, consoante o estado das origens e a manutenção ativa, encaminha para a aplicação ou para as páginas estáticas no Storage, preservando a experiência do utilizador.

Alerta de blocos críticos no WAF. Existe uma regra que avalia minuto a minuto os registos do WAF com Inbound Anomaly Score crítico; agrega a referência, IP e regras acionadas, enviando notificação imediata para triagem.

Gestão de páginas de manutenção. Em manutenção planeada, uma Rule Set aplica *whitelisting* por IP e redireciona o restante tráfego para o *static website* em Storage. Antes de ativar, edita-se o HTML com janela temporal e contactos. Em falha inesperada, o *failover* para a página de emergência ocorre automaticamente.

Monitorização e Alertas com Azure Monitor e Application Insights Foram configurados alertas de CPU e RAM com limiar de 80%, janela de 5 min e reavaliação a cada minuto. No Application Insights está ativa a Smart Detection (anomalias de exceções, falhas, latência, dependências, memória e *traces*) para deteção precoce e diagnóstico.

4.2 Testes

A validação combinou testes manuais e automatizados de forma a confirmar o cumprimento dos requisitos da solução. Nos manuais, confirmou-se a manutenção planeada com IP *whitelisting*, o *failover* para a página de emergência com comutação observada próxima de 15 s e o *fallback* automático após recuperação. O intervalo observado é consistente com os *health probes*, sustentando *failover* $<$ 30 s. Verificou-se ainda o ciclo completo de vulnerabilidades (detetar $\rightarrow$ ingerir $\rightarrow$ notificar $\rightarrow$ encerrar quando crítico).

Nos automatizados, uma *collection* do Postman validou os bloqueios 403 para *SQL Injection*, *Cross-Site Scripting* e *Command Injection* e corroboração em Log Analytics. Um teste de carga (Azure Load Testing) provocou a ativação dos alertas de CPU/RAM, com limiares de 80% (média 5 min; reavaliação a cada 1 min).

4.3 Avaliação da Solução

Os resultados cumprem os critérios definidos. Face às métricas estabelecidas (CPU/RAM a 80% e *failover* $< 30\,\text{s}$), o *failover* ficou em torno de 15 s e o retorno automático foi estável. Na vertente de segurança no perímetro, foram simulados ataques de *SQL Injection*, *Cross-Site Scripting* (XSS) e *Command Injection*, com múltiplos *payloads* por tipo de ataque. Em todos os casos, os pedidos maliciosos foram bloqueados com `HTTP 403` pelo WAF e ficaram registados para análise em Log Analytics; numa execução adicional apenas com tráfego legítimo, não se observaram blocos indevidos, evidenciando ausência de falsos positivos no conjunto de testes efetuado. Sob carga, os limiares de CPU/RAM foram atingidos como previsto e os alertas emitidos apresentaram baixo ruído.

A *pipeline* com cadência de 2 h detetou e ingeriu vulnerabilidades na tabela `Vulnerabilities_CL`. As regras acionaram as notificações e, quando a severidade foi *Critical* no *backend*, ocorreu encerramento controlado com justificação, incluindo alertas baseados em *logs* (p. ex., WAF *Critical Blocks* e *Vulnerabilities Detected*). Estes resultados confirmam que os mecanismos de monitorização, alerta e resposta automática se comportam de forma consistente com os requisitos definidos.

5 Conclusões

Nesta secção apresentam-se os principais resultados obtidos, discutindo em que medida os objetivos foram alcançados e apontando limitações e possíveis linhas de evolução futura.

5.1 Objetivos Concretizados

A implementação cumpriu os objetivos definidos para o projeto. Segue uma lista dos resultados alcançados:

- **Observabilidade**: integração de Azure Monitor e Application Insights para telemetria em tempo real e deteção inteligente de anomalias.
- **Alertas e notificações**: configuração de alertas para recursos e vulnerabilidades (CPU/RAM a 80% com janela de 5 minutos e avaliação a cada 1 minuto e regras KQL sobre `Vulnerabilities_CL`), com notificações acionáveis.
- **Continuidade de serviço**: disponibilização de páginas de manutenção planeada e de emergência, com *failover* validado em cerca de 15 segundos e *fallback* automático.
- **Resposta automática a incidentes**: *pipeline* periódica (cadência de 2 horas), ingestão no Log Analytics e orquestração via Logic App para notificação e encerramento controlado quando crítico no *backend*.
- **Documentação e validação**: processo documentado e testes manuais e automatizados a corroborar os requisitos.

5.2 Limitações e Trabalho Futuro

Foram observadas limitações operacionais. A propagação de alterações no Azure Front Door é lenta (2–10 minutos) e afeta iterações rápidas. A página de manutenção planeada exige edição manual e fica sujeita a erro humano. Para enriquecer as notificações é preciso recorrer à Logic App, porque o formato nativo é rígido. A *pipeline* ainda não integra testes unitários. Estas questões não invalidam os resultados, mas aumentam a complexidade operacional e justificam mitigação.

Como linhas de evolução, propõe-se automatizar a manutenção planeada, incluindo a edição da página e a gestão da Rule Set. Recomenda-se considerar um *token* de acesso temporário além do IP *whitelisting*. Deve avaliar-se uma abordagem híbrida de perímetro (Front Door + Application Gateway) para reduzir ainda mais o tempo de transição. Importa integrar testes automatizados na *pipeline*. Por fim, considera-se relevante analisar a adoção de ferramentas complementares de SCA (Software Composition Analysis) e QA (Quality Assurance), como Dependabot, SonarQube e Snyk.

Em síntese, a solução cumpriu os objetivos de desempenho, segurança e disponibilidade, demonstrando a eficácia de uma abordagem DevSecOps proativa em ambiente Azure.

References

1. Notaro, P., Cardoso, J., Gerndt, M.: A Systematic Mapping Study in AIOps. In: ICSOC 2020 Workshops, LNCS 12632, pp. 110–123 (2021). https://api.semanticscholar.org/CorpusID:229210898
2. Remil, Y., Bendimerad, A., Mathonat, R., Kaytoue, M.: AIOps Solutions for Incident Management: Technical Guidelines and A Comprehensive Literature Review. arXiv:2404.01363 (2024). https://arxiv.org/abs/2404.01363
3. Wang, X. et al.: Graph-based Anomaly Detection and Root Cause Analysis for Microservices in Cloud-Native Platform. In: 2024 IEEE SustainCom (2024). https://api.semanticscholar.org/CorpusID:276935952
4. Kratzke, N.: Cloud-Native Observability: The Many-Faceted Benefits of Structured and Unified Logging—A Multi-Case Study. Future Internet 14(10), 274 (2022). https://doi.org/10.3390/fi14100274
5. Karlzén, H., Sommestad, T.: Automatic Incident Response Solutions: A Review of Proposed Solutions' Input and Output. In: ARES 2023, pp. 37:1–37:9 (2023). https://doi.org/10.1145/3600160.3605066
6. Cheenepalli, J., Hastings, J.D., Ahmed, K.M., Fenner, C.: Advancing DevSecOps in SMEs: Challenges and Best Practices for Secure CI/CD Pipelines. arXiv:2503.22612 (2025). https://arxiv.org/abs/2503.22612
7. Microsoft Azure: Circuit Breaker pattern. https://learn.microsoft.com/en-us/azure/architecture/patterns/circuit-breaker
8. Google Cloud: Google Cloud Platform Comparison with AWS and Azure. https://cloud.google.com/docs/get-started/aws-azure-gcp-service-comparison
9. Grady, R.B.: Practical Software Metrics for Project Management and Process Improvement. Prentice Hall (1992).
10. Brown, S.: C4 Model for Visualising Software Architecture. https://c4model.com