XOps Game Store: Pipeline Inteligente CI/CD com IA e Segurança

1° Martim Silva nº52023
Informática Web. Móvel e na Núvem
Departamento Informática
Covilhã, Portugal

2° Rodrigo Gonçalves nº51823
Informática Web. Móvel e na Núvem
Departamento Informática
Covilhã, Portugal

3° Rodrigo Soares nº51589
Informática Web. Móvel e na Núve
Departamento Informática
Covilhã, Portugal

***Abstract*—Este trabalho descreve o desenvolvimento de um pipeline de Integração e Entrega Contínuas (CI/CD) para um website estático de venda de jogos, a XOps Game Store. O objetivo principal foi construir um fluxo de entrega automatizado que, para além de compilar e publicar o site, integrasse capacidades de Inteligência Artificial(IA) e verificações de segurança a nível de código (SAST) e de aplicação em execução (DAST).**

O pipeline foi implementado com GitHub Actions e publica o site automaticamente em GitHub Pages. A IA é utilizada para gerar uma análise automática dos ficheiros do projeto, enquanto ferramentas como CodeQL e OWASP ZAP analisam vulnerabilidades de segurança. O projeto foi organizado em quatro sprints seguindo uma abordagem inspirada em Scrum, desde o setup inicial até à demonstração final. Os resultados mostram que é possível construir, com ferramentas gratuitas, um pipeline CI/CD simples, mas com funcionalidades de qualidade e segurança típicas de projetos profissionais.

***Index Terms*—CI/CD, DevOps, Inteligência Artificial, Segurança, CodeQL, OWASP ZAP, GitHub Actions.**

I. Introduction

A adoção de práticas de Integração Contínua e Entrega Contínua (CI/CD) é hoje fundamental para garantir entregas rápidas e consistentes de software. Em projetos web, um pipeline bem definido reduz erros manuais, automatiza testes e facilita o deploy para produção. Neste trabalho é apresentado o desenvolvimento de um pipeline CI/CD inteligente para a XOps Game Store, um site estático em HTML/CSS inspirado em plataformas de venda de jogos (como a Steam). Para além das etapas tradicionais de build e deploy, o pipeline integra:

- uma etapa de IA, para análise automática dos ficheiros do projeto;
- uma etapa de SAST, com GitHub CodeQL, para analisar o código;
- uma etapa de DAST, com OWASP ZAP, para testar a aplicação já publicada.

O trabalho foi dividido em quatro sprints:

- Sprint 1 — Setup ePlaneamento: criação do repositório GitHub, configuração do Jira, definição do Team Contract e arquitetura inicial.
- Sprint 2 — PipelineBásico: criação do pipeline CI/CD com build, teste simples e deploy para GitHub Pages.
- Sprint 3 — IA e Segurança: integração de IA, SAST e DAST no pipeline.
- Sprint 4 — Demonstração eRelatório: consolidação, demonstração final e elaboração deste relatório.

O objetivo global foi mostrar, de forma prática, como combinar automação, IA e segurança num pipeline moderno.

II. Metodologia

A gestão do projeto seguiu uma abordagem incremental, inspirada em Scrum:

- O trabalho foi organizado em sprints, cada um com objetivos claros e entregáveis definidos.
- Foi usado o Jira para gerir o backlog, criar tarefas, acompanhar o estado das issues e documentar evidências.
- Cada sprint tinha um conjunto de "Definition of Done" associado às tarefas, como por exemplo: prints do pipeline, links para o site e atualizações no README.

As principais ferramentas utilizadas foram:

- GitHub —- repositório do código, gestão de versões e issues de segurança (CodeQL).
- GitHub Actions —- implementação do pipeline CI/CD no ficheiro .github/ workflows/deploy.yml.
- GitHub Pages —- alojamento do site estático publicado automaticamente pelo pipeline.
- Jira —- gestão de tarefas, sprints e artefactos.
- ChatGPT / OpenAI — apoio na revisão lógica e textual do código e geração do passo a passo (representado no pipeline pela job ai-review).

- CodeQL —- análise estática de código (SAST).
- OWASP ZAP —- análise dinâmica de segurança (DAST) ao site publicado.

A metodologia consistiu em evoluir o pipeline a cada sprint, adicionando novas capacidades sem quebrar as funcionalidades anteriores.

III. Arquitetura do Sistema

A arquitetura global do sistema pode ser vista em dois níveis: aplicação e pipeline.

A. 1. Aplicação Web

A aplicação é um site estático composto por:

- ficheiros HTML (por exemplo, index.html, páginas de jogos, compras, promoções);
- ficheiro CSS (styles.css) e outros recursos visuais;
- diretórios como img/, files/ e vid/, com imagens, páginas auxiliares e conteúdos multimédia.

Não existe back-end ou base de dados: o foco do projeto está no pipeline e na segurança do front-end.

B. 2. Pipeline CI/CD

O pipeline está definido no ficheiro: .github/workflows/deploy.yml

A arquitetura do pipeline, em alto nível, é:

- 1. Build e Test: checkout do repositório e teste básico aos links HTML.
- 2. Deploy: publicação do artefacto _site em GitHub Pages.
- 3. IA Review: passo lógico que representa a análise da IA ao código.
- 4. SAST: análise estática com CodeQL.
- 5. DAST: análise dinâmica com OWASP ZAP ao site publicado.

Na prática, o fluxo segue a sequência:

Código GitHub GitHub Actions (build test IA SAST DAST deploy) GitHub Pages Esta arquitetura permite que, a cada push para a branch main, toda a pipeline seja executada automaticamente.

IV. Pipeline CI/CD Inteligente

O pipeline é desencadeado por qualquer push ou pull_request na branch main. O ficheiro deploy.yml contém cinco jobs principais: build-and-test, deploy, ai-review, codeql e zap-scan.

A. Job build-and-test

Esta job corre em ubuntu-latest e executa:

- Checkout do repositório com actions/checkout@v4.
- Verificação de links HTML, usando lycheeverse/lychee-action@v1. Este passo percorre os ficheiros HTML e valida se os links não estão partidos.
- Preparação do artefacto _site, copiando o conteúdo do repositório para uma pasta temporária, excluindo diretórios internos como .git, .github e ficheiros .zip.
- Upload do artefacto, com actions/upload-pages-artifact@v3, que será depois usado na fase de deploy.

Se esta etapa falhar, as restantes jobs dependentes não são executadas, garantindo que o deploy e as análises de segurança só acontecem sobre builds válidos.

B. Job deploy

Esta job depende de build-and-test e usa actions/deploy-pages@v4 para:

- recuperar o artefacto _site;
- publicar o conteúdo em GitHub Pages, na URL pública do projeto.

Após a conclusão desta job, o site fica acessível em: https://martim182004.github.io/xops-game-store-pipeline/

C. Job ai-review

A job ai-review representa a integração de IA na pipeline. Ela depende de build-and-test e executa:

- novo checkout do repositório;
- criação de um ficheiro ia-review.log com a lista de ficheiros HTML/CSS analisados.

No YAML é passada a variável de ambiente OPENAI_API_KEY, configurada como secret no GitHub. Embora o pipeline de demonstração não faça uma chamada real à API, esta job mostra como o pipeline foi preparado para, em contexto real, enviar ficheiros ou resumos para a IA produzir comentários ou sugestões de melhoria.

D. Job codeql

A job codeql faz a integração de SAST:

- 1. Faz checkout do código.
- 2. Inicializa o CodeQL com github/codeql-action/init@v3, configurado para a linguagem javascript (relevante para os scripts utilizados nas páginas).
- 3. Executa a análise com github/codeql-action/analyze@v3.

Os resultados são enviados para a secção Security Code scanning alerts do GitHub. No projeto foi detetado, por exemplo, um alerta de gravidade média relativo a inclusão de código JavaScript a partir de uma fonte externa sem verificações de integridade.

E. Job zap-scan

Por fim, a job zap-scan corre apenas após deploy e utiliza zaproxy/action- baseline@v0.12.0 para:

- fazer spidering ao site publicado;
- executar passive scanning;
- gerar um relatório com os alertas encontrados.

A configuração usada (cmd_options: ”-a”) ativa verificações mais completas, mas a job é marcada com continue-on-error: true para que eventuais problemas na geração de artefactos não façam falhar todo o pipeline.

O output mostra dezenas de regras marcadas como PASS e um conjunto de WARN- NEW, com avisos típicos de aplicações estáticas hospedadas em GitHub Pages
(por exemplo, cabeçalhos de segurança ausentes).

V. Integração de IA

A integração de IA no pipeline é representada pela job ai-review. A ideia foi demonstrar como se poderia incorporar uma etapa de revisão automática suportada por modelos como o ChatGPT:

- A variável OPENAI_API_KEY é configurada como secret no repositório, de forma a não expor a chave pública.
- Na job, esta variável é disponibilizada via env, mostrando que o pipeline está pronto para consumir a API da OpenAI.
- O script de demonstração recolhe a lista de ficheiros relevantes e escreve num ficheiro de log, que pode ser o ponto de partida para uma integração real.

Na prática deste trabalho académico, a IA foi usada sobretudo fora do pipeline, como apoio ao planeamento, à escrita do README e à revisão do código. A job ai-review formaliza essa integração dentro da pipeline de forma controlada e segura.

Entre as limitações observadas estão:

- Dependência de conectividade externa e da disponibilidade da API.
- Possível custo associado, se fosse usada de forma intensiva.
- Necessidade de validação humana das sugestões geradas.

VI. Análise SAST com CodeQL

O CodeQL é a ferramenta oficial da GitHub para análise estática de código. No contexto deste projeto, foi usado para analisar o código JavaScript incluído no site.

Após a execução da job codeql, o GitHub passou a apresentar, na secção Security, um alerta de gravidade média com o título: Inclusion of functionality from an untrusted source

O alerta refere-se à inclusão de bibliotecas JavaScript a partir de CDNs externos (por exemplo, jQuery), sem mecanismos de integridade (como Subresource Integrity). Embora esta prática seja comum em sites simples, pode abrir a porta a ataques caso o recurso externo seja comprometido. Este resultado é importante por duas razões:

- 1. Mostra que o SAST está funcional, detetando potenciais problemas.
- 2. Obriga a refletir sobre boas práticas, como minimizar dependências externas ou adicionar verificações adicionais.

VII. Análise DAST com OWASP ZAP

Para complementar a análise estática, foi utilizada a ferramenta OWASP ZAP em modo baseline, através da action oficial zaproxy/action-baseline. O ZAP é executado num contentor Docker e faz:

- Spidering do site, descobrindo URLs e páginas acessíveis;
- Passive scanning das respostas HTTP;
- Geração de um resumo com:
  o número de URLs analisadas (146 no caso do projeto);
  o número de testes PASS;
  o número de WARN-NEW, FAIL-NEW, etc.

No scan realizado, o relatório indicou:

- FAIL-NEW: 0 — sem falhas críticas novas;
- WARN-NEW: 19 — vários avisos, nomeadamente:
  o ausência de cabeçalhos de segurança (Content Security

Policy, HSTS, X-Frame-Options, etc.);
o ausência de tokens anti-CSRF em páginas de compra;
o conteúdos servidos sem algumas diretivas de cache.

Estas observações são típicas de aplicações estáticas simples e, muitas vezes, limitadas pelas próprias características do GitHub Pages. Ainda assim, o relatório é útil para discutir como se poderia endurecer a aplicação num contexto real (por exemplo, usando um servidor próprio com configuração de cabeçalhos).

VIII. Resultados

Os principais resultados do projeto podem ser resumidos em três pontos:

- 1. Pipeline automatizado funcional
  A cada push ou pull_request em main, o pipeline é acionado automaticamente.
  O build e o deploy são executados sem intervenção manual, mantendo o site sempre atualizado.

- 2. Integração com IA
  A pipeline inclui uma job dedicada à IA, preparada para consumir a API da OpenAI.
  A IA foi utilizada na prática como apoio ao desenvolvimento, planeamento e documentação.

- 3. Análise de segurança inicial
  O CodeQL identificou um alerta relevante sobre inclusão de código de fontes externas.
  O ZAP analisou a aplicação publicada e produziu um conjunto de avisos que ajudam a refletir sobre cabeçalhos de segurança e proteção contra ataques comuns.

No conjunto, o projeto cumpre os objetivos propostos: construir um pipeline CI/ CD simples, mas com preocupações reais de qualidade e segurança.

IX. Discussão e Ética

A introdução de IA e automação de segurança levanta questões técnicas e éticas. Onde a IA ajudou:

- Aceleração do planeamento e documentação.
- Sugestões de melhorias ao pipeline.
- Apoio na explicação e clareza técnica.

Onde a IA não substitui o humano:

- A IA não "ouve" o contexto completo da disciplina nem a intenção da avaliação.
- As recomendações técnicas precisam de ser validadas manualmente.
- Um mau prompt pode levar a respostas incorretas.

Riscos e limitações:

- Dependência excessiva da IA pode reduzir o espírito crítico e a aprendizagem real.
- Questões de privacidade: enviar código sensível para serviços externos pode ser problemático em contextos empresariais.
- Viés nos modelos: as recomendações podem refletir práticas mais comuns, mas não necessariamente as mais adequadas ao contexto.

Do ponto de vista ético, é importante que a IA seja vista como uma ferramenta de apoio, e não como substituto de responsabilidade técnica. O desenvolvedor continua responsável pelo código que entrega.

X. Conclusões

Este projeto demonstrou que é possível construir, com um conjunto de ferramentas gratuitas, um pipelineCI/CDinteligente para um site estático, integrando:

- build e testes básicos;
- deploy automático para GitHub Pages;
- análise assistida por IA;
- verificações de segurança SAST (CodeQL) e DAST (OWASP ZAP).

Ao longo dos quatro sprints, foram atingidos os objetivos definidos: desde o setup do projeto e planeamento no Jira, até à integração de IA e segurança e à preparação da demonstração final.

Como trabalho futuro, seria interessante:

- substituir a job de IA por uma integração real com a API, gerando comentários automáticos mais detalhados;
- melhorar o front-end com testes funcionais adicionais;
- migrar o site para uma infraestrutura que permita configurar cabeçalhos de segurança, mitigando muitos dos avisos detetados pelo ZAP;
- adicionar testes unitários e de integração, aumentando a cobertura de qualidade.

O projeto permitiu consolidar conhecimentos de desenvolvimento web, DevOps, segurança e uso responsável de IA no contexto de engenharia de software.

REFERENCES

[1] GitHub, “GitHub Actions Documentation.” Disponível em: https://docs.github.com/actions

[2] GitHub, “GitHub Pages Documentation.” Disponível em: https://docs.github.com/pages

[3] GitHub Security Lab, “CodeQL: The Industry’s Leading Code Analysis Engine.” Disponível em: https://codeql.github.com

[4] OWASP Foundation, “OWASP Zed Attack Proxy (ZAP).” Disponível em: https://owasp.org/www-project-zap/

[5] Lychee, “Lychee Broken Link Checker.” GitHub Repository. Disponível em: https://github.com/lycheeverse/lychee.

[6] Atlassian, “Jira Software Documentation.” Disponível em: https://support.atlassian.com/jira-software/

[7] Scrum.org, “Scrum Guide – The Definitive Guide to Scrum.” Disponível em: https://scrumguides.org

[8] W3C, “Subresource Integrity (SRI) Specification.” Disponível em: https://www.w3.org/TR/SRI/

[9] Mozilla Developer Network (MDN), “HTTP Security Headers.” Disponível em: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

[10] Docker, “Docker Documentation.” Disponível em: https://docs.docker.com/