Engenharia de Software Seguro
Laboratório 5 — Containerização, CI/CD e DevSecOps

Grupo 6
Elementos: Hugo Nunes, Miguel Teixeira
Repositório GitHub: miguelteixeira77/ess-task-api-lab5
Tag(s): lab5
Data: 22-02-2026

Objetivos principais:

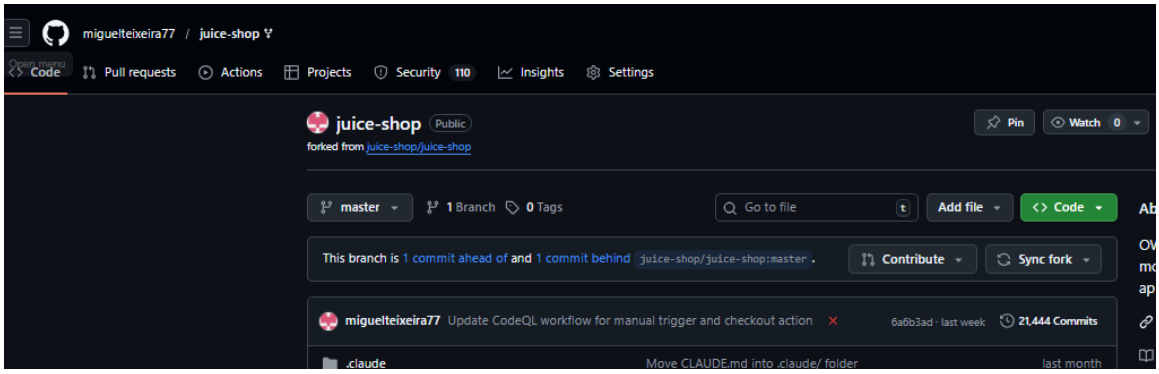
(i) Criar imagem Docker multi-stage.
(ii) Validar execução via container.
(iii) Publicar imagem em registry.
(iv) Automatizar build/test/publish em github Actions.
(v) Integrar controlos de segurança (secret scanning, image scanning, assinatura e SBOM).

Índice

1. GitHub Actions – Aplicação Juice Shop

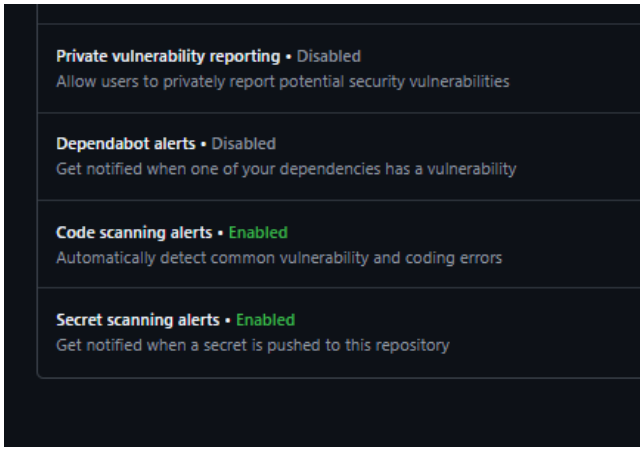
1.a Fork do Repositório

Foi realizado o fork do repositório oficial OWASP Juice Shop para a conta GitHub do grupo, mantendo o repositório público e adicionando todos os elementos do grupo como colaboradores.



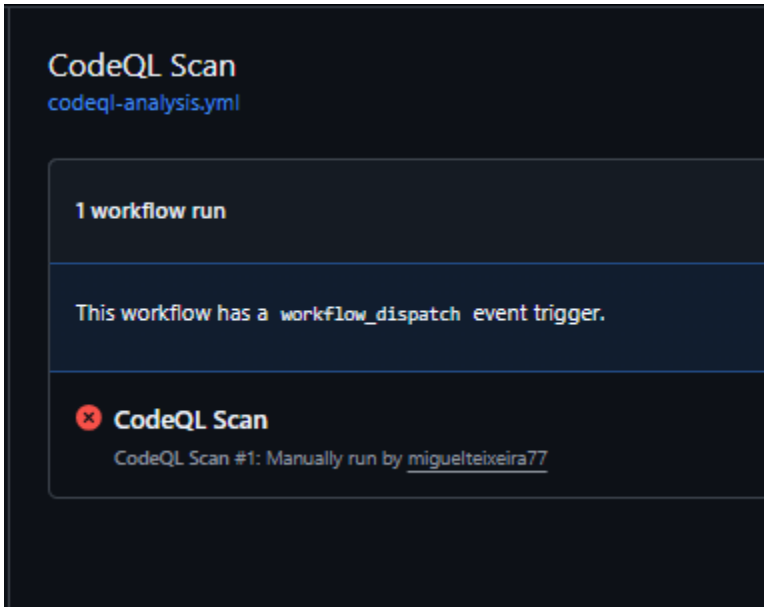
1.b Ativação do Code Scanning

Foi ativado o Code Scanning no menu Security do repositório, utilizando a configuração padrão do CodeQL disponibilizada pelo GitHub.



1.c Alteração do Trigger do Workflow

O workflow do CodeQL foi alterado para ser executado manualmente utilizando o trigger 'workflow_dispatch', permitindo a execução sob pedido no separador Actions.



1.d Inicialização da Base de Dados CodeQL

Nos logs da execução foi identificada a etapa 'Initialize CodeQL', responsável pela criação da base de dados de análise estática. Esta etapa executa o comando 'codeql database init', que prepara a representação intermédia do código fonte para posterior análise.

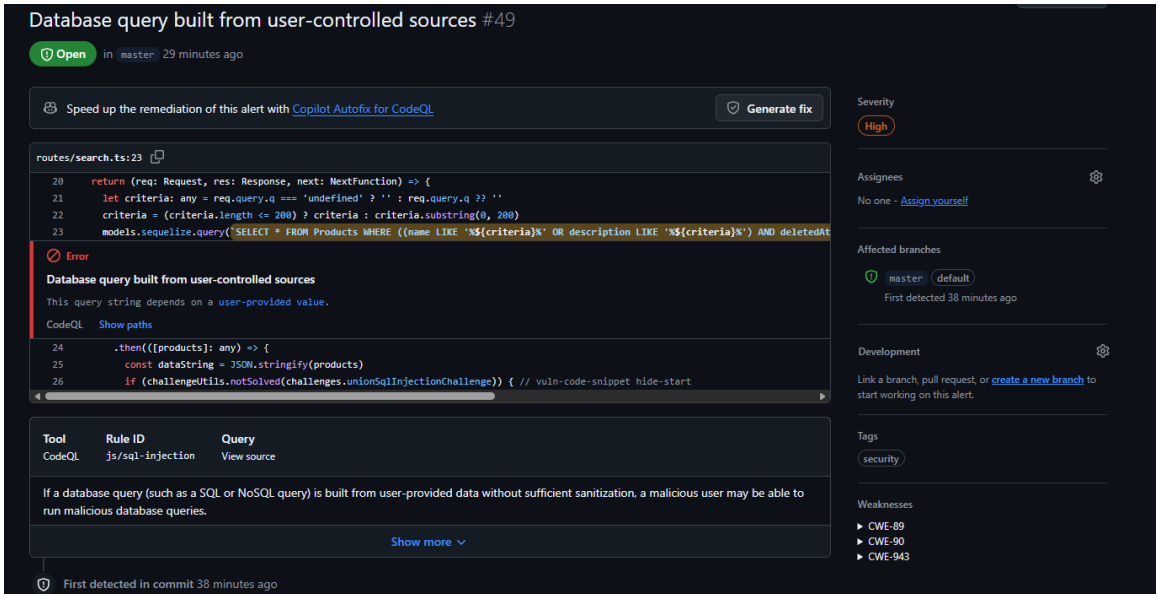
1.e Análise da Vulnerabilidade CWE-89 (SQL Injection)

O CodeQL identificou a vulnerabilidade 'Database query built from user-controlled sources' no ficheiro routes/search.ts. Esta vulnerabilidade ocorre porque dados provenientes de uma source controlada pelo utilizador (parâmetros HTTP) fluem diretamente para um sink sensível (execução de query SQL) sem validação ou utilização de prepared statements.

Source: Parâmetros HTTP (req.query, req.body, req.params).

Sink: Execução de queries SQL através de funções como db.query().

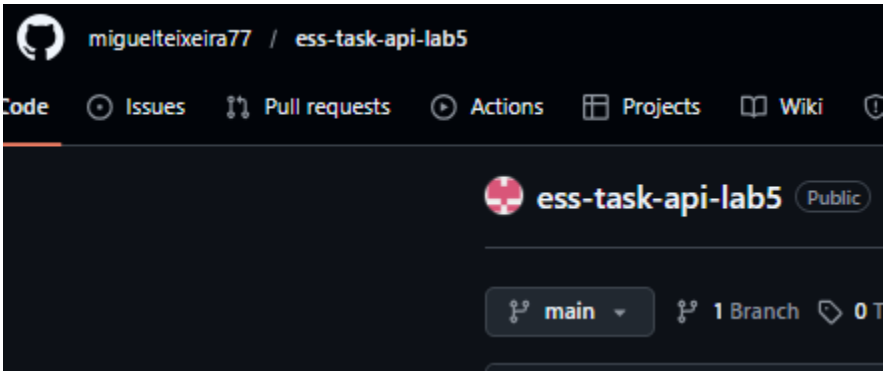
Esta situação corresponde à CWE-89 – Improper Neutralization of Special Elements used in an SQL Command.



2. GitHub Actions – API de Tarefas

2.a Criação do Repositório

Foi criado um novo repositório público na conta GitHub do grupo para alojar a API de tarefas.



2.b Adição do Código Fonte

Foi adicionado ao repositório o código fonte da API de tarefas desenvolvida anteriormente.

2.c Criação do Workflow Hello World

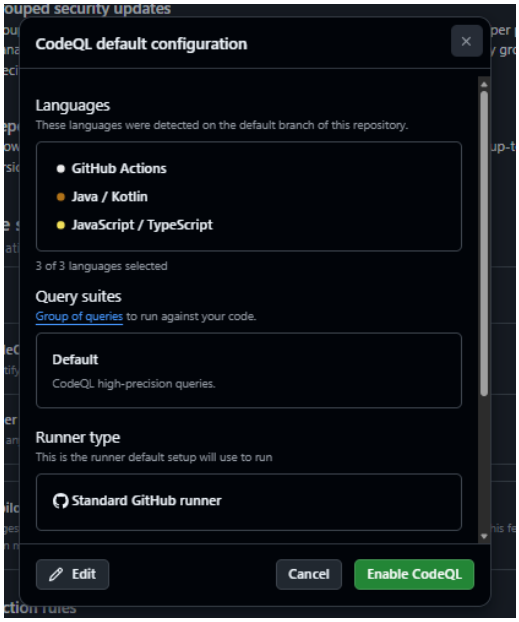
Foi criado o ficheiro '.github/workflows/hello-world.yml' com trigger 'workflow_dispatch', executando o comando 'echo Hello, world!'.

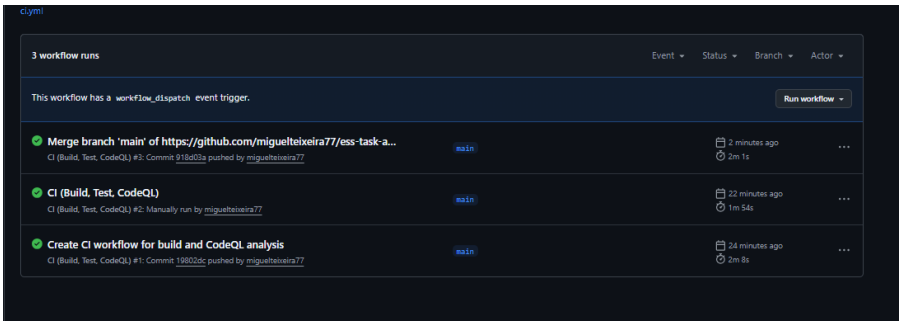
2.d Localização do Workflow

Após realizar pull do repositório para o ambiente local, o ficheiro encontra-se na diretoria: .github/workflows/hello-world.yml.

2.e Pipeline de Build com Maven e CodeQL

Foi implementado um pipeline de CI composto pelas etapas de compilação (mvn clean install), execução de testes unitários (mvn test) e análise estática com CodeQL.





2.f Vulnerabilidade Zip Slip (CWE-22)

Foi introduzida uma vulnerabilidade do tipo Zip Slip através de um método que extrai ficheiros ZIP sem validação do caminho das entradas. Esta vulnerabilidade permite Path Traversal, possibilitando a escrita de ficheiros fora do diretório pretendido.

A vulnerabilidade Zip Slip está relacionada com a CWE-22 (Path Traversal), uma vez que permite manipulação de caminhos no sistema de ficheiros.

3. Containerização da Aplicação

3.1 Criação do Dockerfile (Multi-stage)

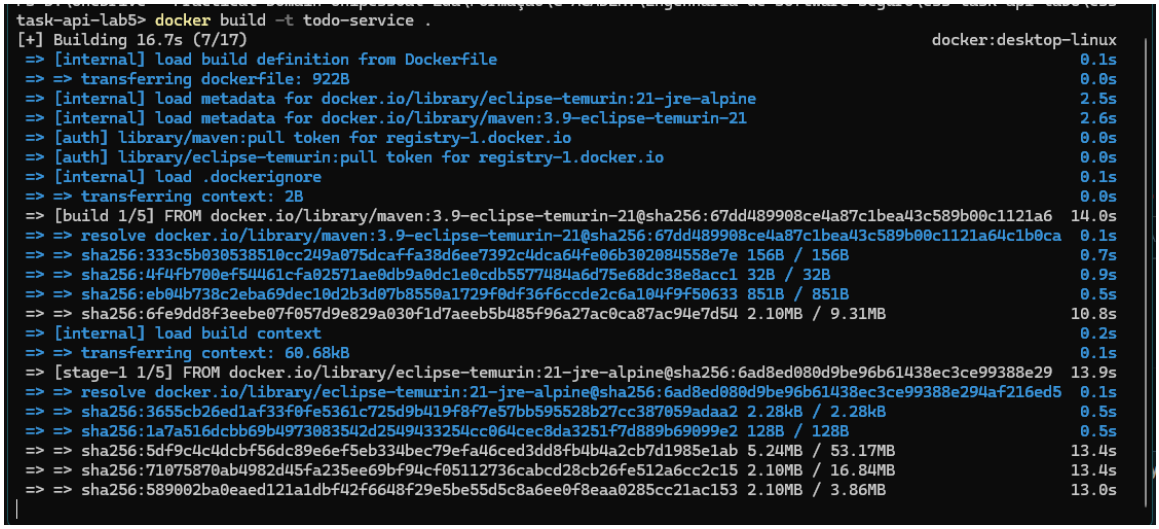
Foi criado um Dockerfile multi-stage, permitindo separar a fase de build (Maven) da fase de runtime (JRE). Esta abordagem reduz o tamanho final da imagem e melhora a segurança.

• Stage 1: Build com Maven

• Stage 2: Runtime com eclipse-temurin JRE

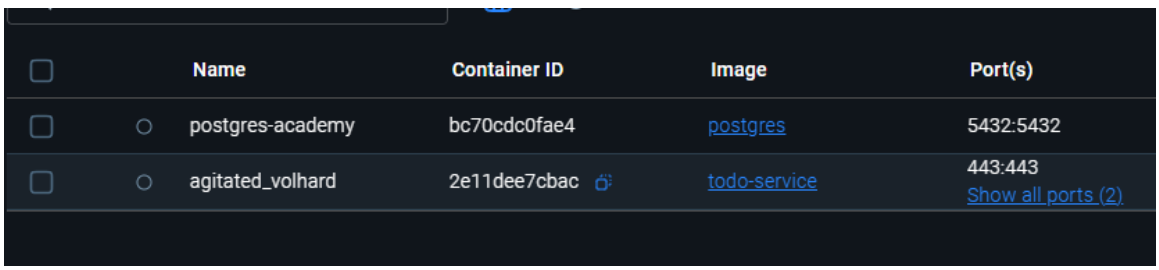
3.2 Build da Imagem Docker

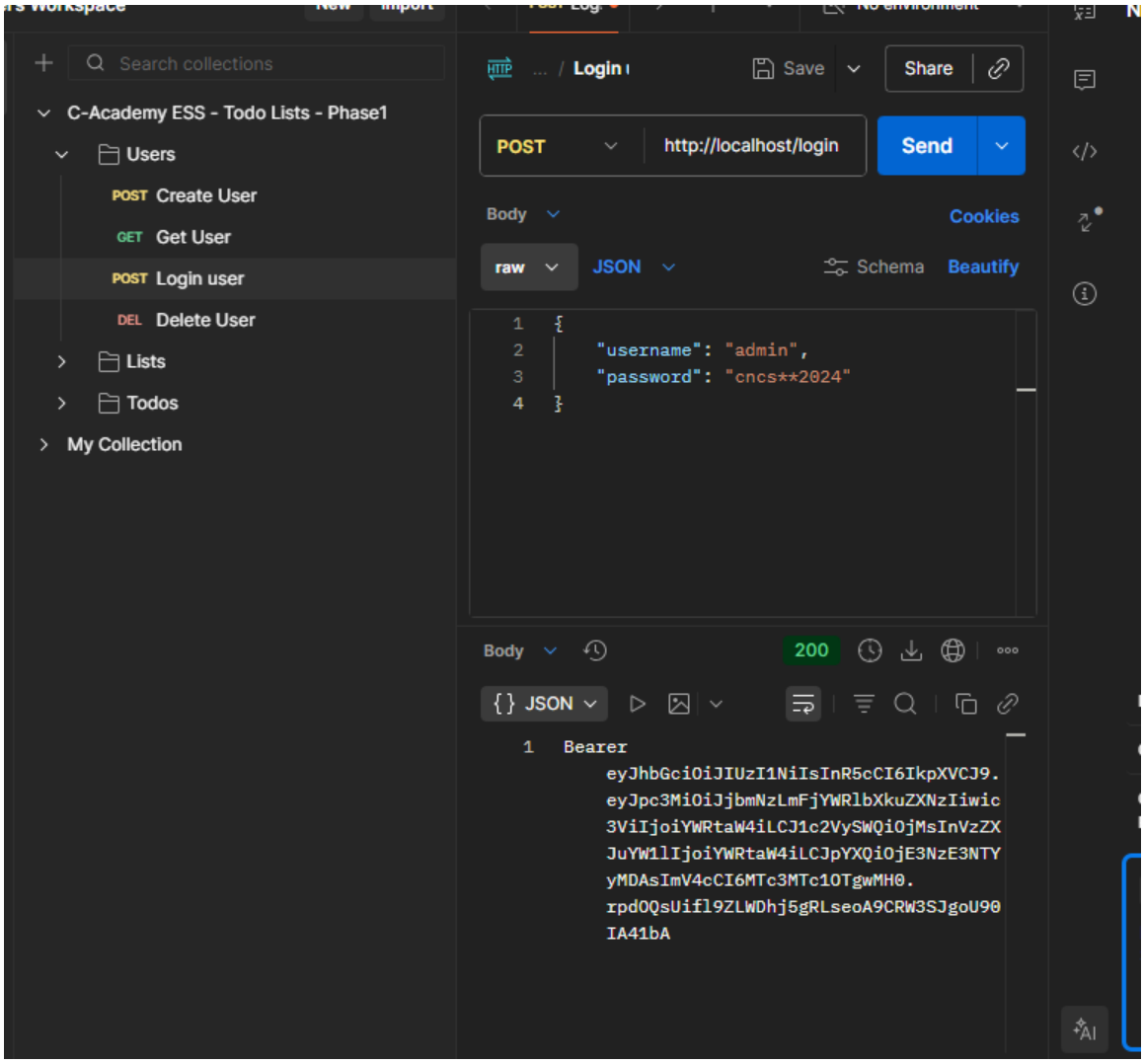
A imagem foi construída localmente utilizando o comando docker build.

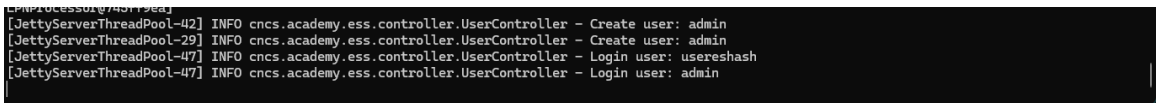


3.3 Execução da Imagem

A imagem foi executada com mapeamento de portas e validação do funcionamento da API.







3.4 Publicação no DockerHub

Após autenticação com docker login, a imagem foi publicada no DockerHub.

Comandos utilizados:

• docker tag

• docker push

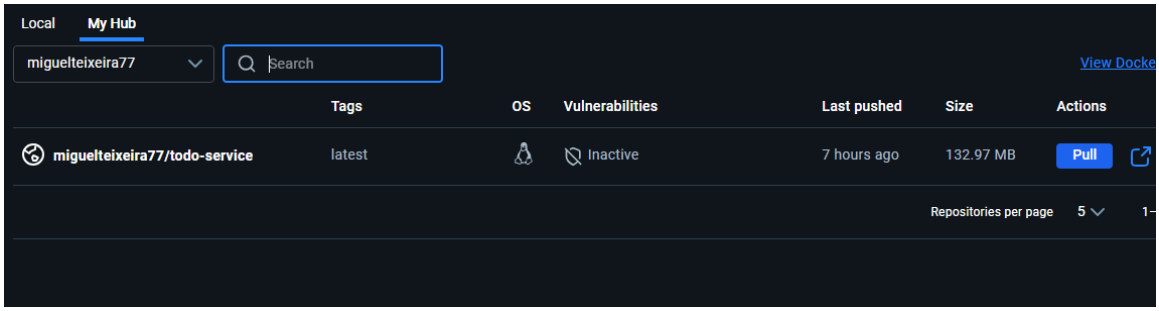
```
k-api-lab5\ess-task-api-lab5> docker tag todo-service miguelteixeira77/todo-service
```

```
k-api-lab5\ess-task-api-lab5> docker login
Authenticating with existing credentials... [Username: miguelteixeira77]

i Info → To login with a different account, run 'docker logout' followed by 'docker login'


Login Succeeded
```

```
k-api-lab5\ess-task-api-lab5> docker push miguelteixeira77/todo-service
Using default tag: latest
The push refers to repository [docker.io/miguelteixeira77/todo-service]
51024d09d55a: Pushed
c4247047ab40: Pushed
f1389b9b70dd: Pushed
1a005a500401: Pushed
d23efddf26cd: Pushed
4222d53aeb55: Pushed
01d7766a2e4a: Pushed
f3b2661a0534: Pushed
5231ace86ec4: Pushed
05ec7c6edf1e: Pushed
latest: digest: sha256:671ef24c66056b8d1d0dfadbc360765b4db9fe8fac98b69834d9fb8f4ffd5e3b size: 856
```

3.5 Commit do Dockerfile

O Dockerfile foi versionado e enviado para o repositório GitHub.



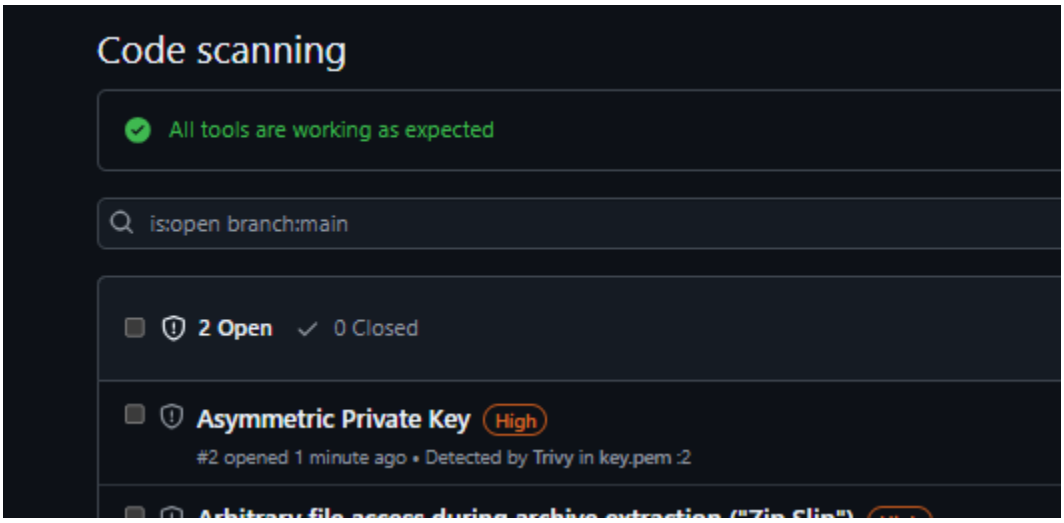
4. Pipeline CI/CD Seguro

4.1 Criação do Workflow Base (cicd-base.yml)

Foi criado um workflow no GitHub Actions para execução automática em push e pull request. Este inclui build Maven, testes e publicação da imagem no GitHub Container Registry (GHCR).

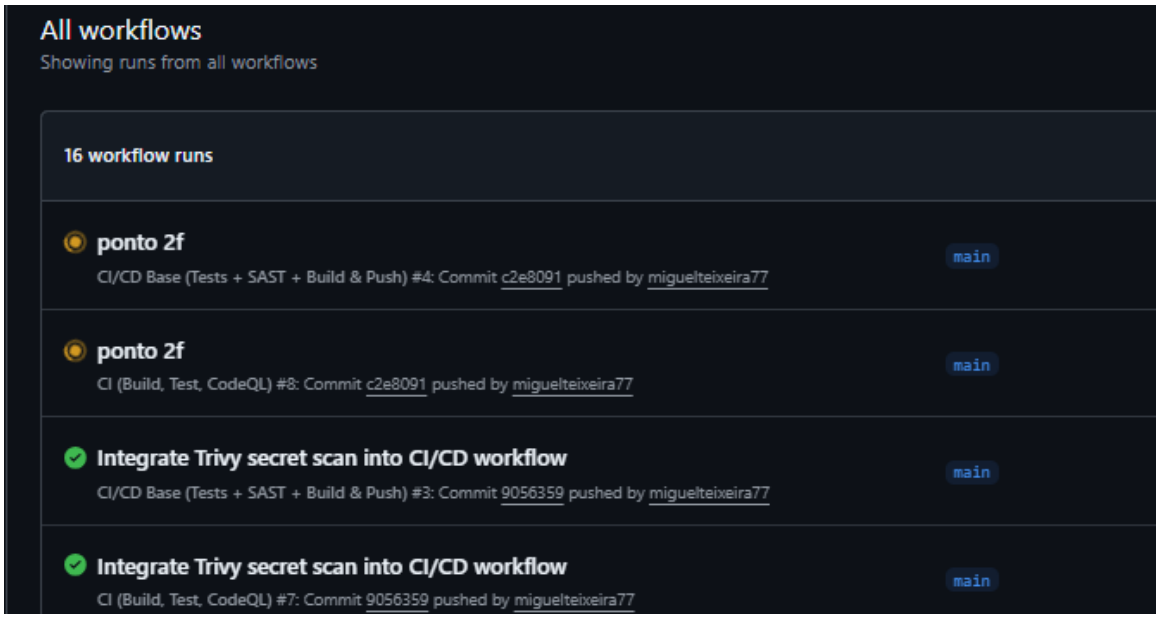
4.2 Deteção de Segredos com Trivy

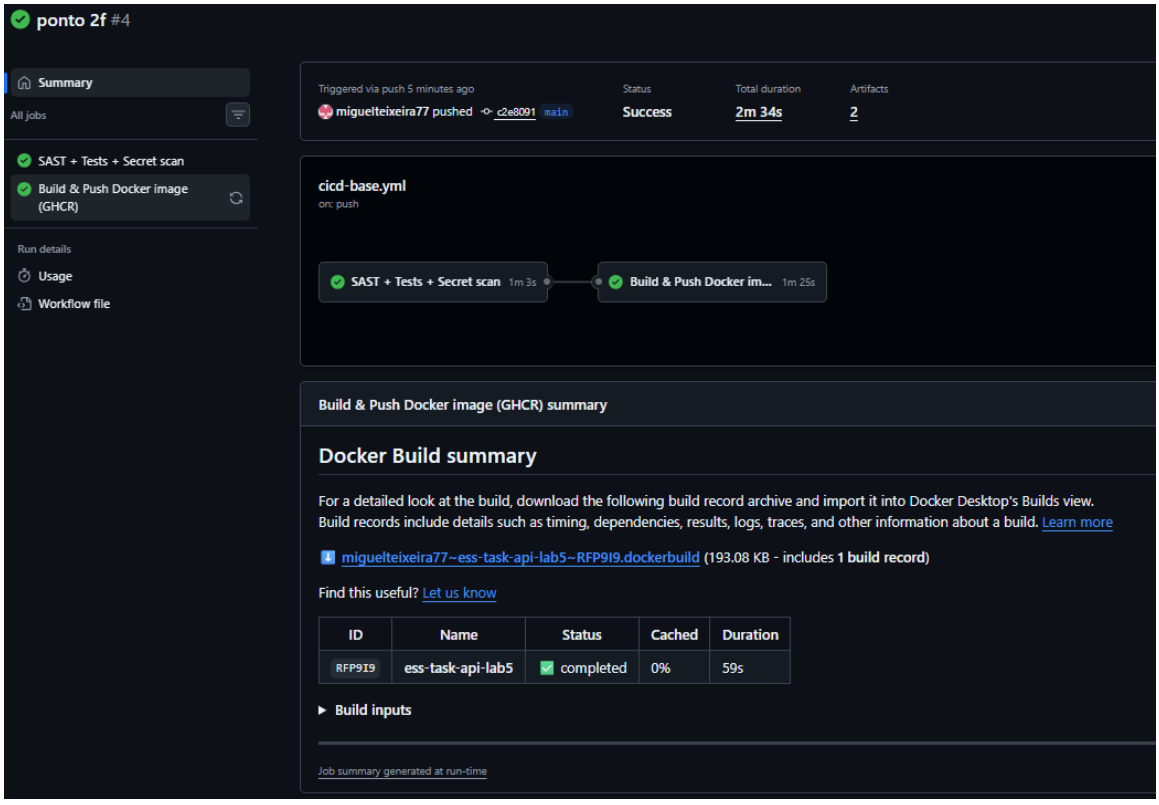
Foi integrado o Trivy para detetar segredos no código-fonte (filesystem scan). Os resultados foram exportados em formato SARIF e publicados na secção Security → Code Scanning.



4.3 Separação do Pipeline em Dois Jobs

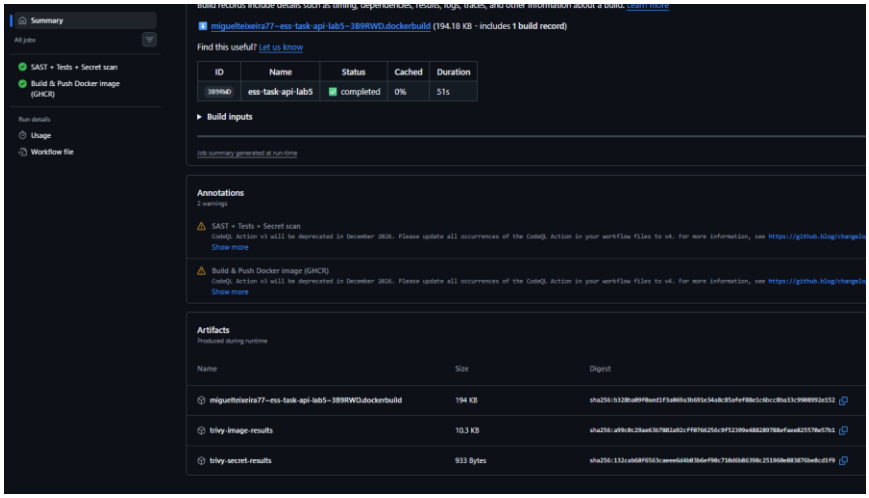
O pipeline foi dividido em dois jobs distintos: 1) SAST + Testes + Secret Scan 2) Build & Push da imagem Docker (dependente do primeiro através de 'needs').





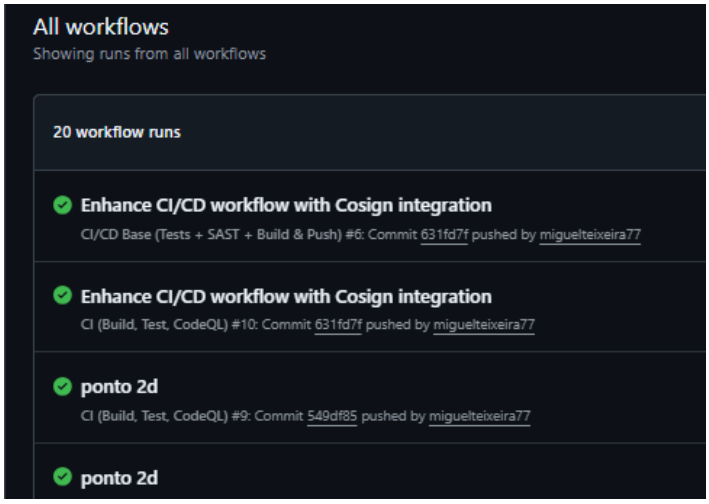
4.4 Scan de Vulnerabilidades da Imagem Docker

Foi realizado um scan de vulnerabilidades à imagem Docker utilizando Trivy. Os resultados foram exportados em SARIF e disponibilizados como artifact.



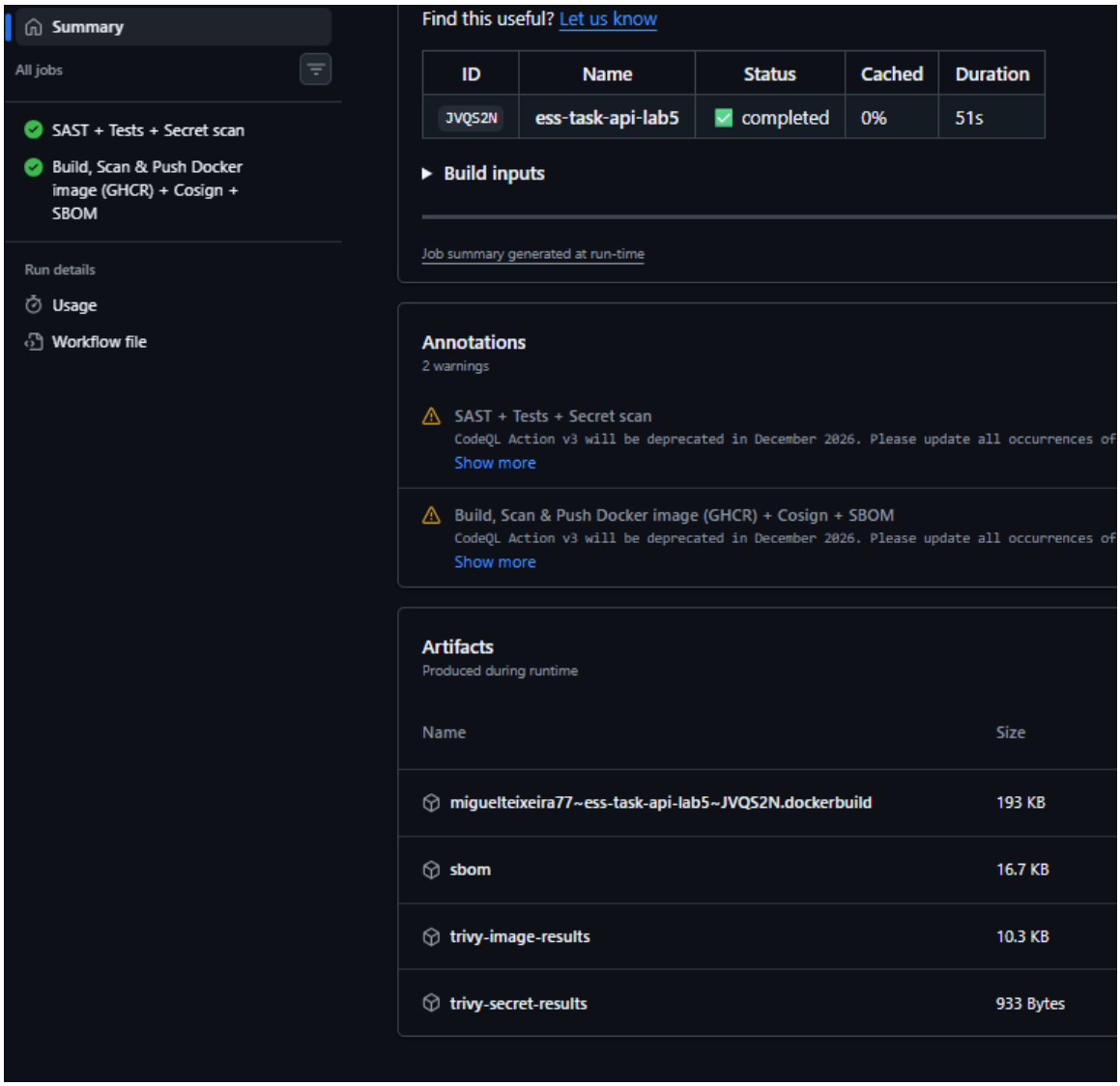
4.5 Assinatura da Imagem com Cosign

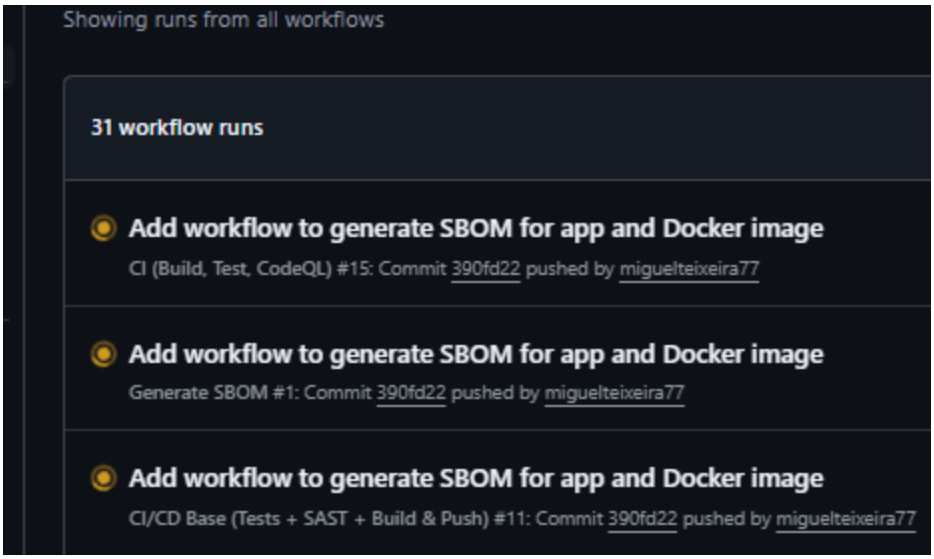
A imagem Docker foi assinada utilizando Cosign (modo keyless) com suporte OIDC do GitHub Actions. As assinaturas foram armazenadas no GHCR associadas à imagem publicada.



4.6 Geração de SBOM

Foi gerado um SBOM no formato CycloneDX (sbom.cdx.json), permitindo listar todas as dependências e componentes da imagem. O ficheiro foi disponibilizado como artifact do workflow.





Os ficheiros SBOM gerados no ponto 2.g ficam disponíveis como artefactos do workflow no GitHub Actions, podendo ser descarregados através do separador *Actions* após a execução do pipeline.

As assinaturas da imagem Docker, geradas com *cosign*, ficam armazenadas no GitHub Container Registry (GHCR) como artefactos associados à imagem publicada.

Tanto os SBOM como as assinaturas podem ser descarregados ou verificados localmente, garantindo a integridade e rastreabilidade da imagem construída.

5. Conclusão

A Ficha 5 permitiu implementar um pipeline CI/CD completo com integração de práticas DevSecOps. Foram aplicadas técnicas de deteção de segredos, análise de vulnerabilidades, assinatura de imagens Docker e geração de SBOM.

O resultado é um fluxo automatizado seguro que reforça a segurança da cadeia de fornecimento de software.

Este trabalho demonstra a aplicação prática de conceitos modernos de segurança em pipelines de integração e entrega contínua.