Taller Pipeline DevSecOps Básico - Versión Simplificada

Integrantes:
Matías Navia
Eduardo Miranda
Ignacio Rebolledo
Cristóbal Ríos

Docente:
Rafael Ramos Aqueda

Asignatura:
Seguridad Inf. en Des. Sw.

Fecha: 21/06/2025

Tabla de Contenidos

1 Implementación y Resultados

1.1 Dockerización

Para la implementación del manejo de contenedores se crearon dos archivos `Dockerfile`, uno para el backend y otro para el frontend, así como un archivo `docker-compose.yml` ubicado en el directorio raíz del proyecto denominado `myapp-segura`. Esta estrategia permitió contenerizar ambos servicios de manera independiente, pero orquestados conjuntamente a través de un único comando, facilitando la ejecución y despliegue del sistema completo en cualquier entorno compatible con Docker.

El contenedor del **backend** en NestJS se construye a partir de una imagen base de Node.js. Durante el proceso de construcción se copian los archivos del proyecto, se instalan las dependencias necesarias mediante `npm install` y se compila el código TypeScript a JavaScript mediante `npm run build`. El contenedor expone el puerto 3000 y ejecuta el servidor utilizando el comando `node dist/src/main.js`, permitiendo servir la API de forma eficiente y productiva.

El contenedor del **frontend** se construye a partir de una imagen base de Node.js, donde primero se copian los archivos del proyecto, se instalan las dependencias necesarias mediante `npm install` y se genera la versión optimizada de la aplicación React con Vite usando `npm run build`. Luego, en una segunda etapa, se utiliza una imagen de Nginx para servir los archivos estáticos generados, copiándolos al directorio correspondiente de Nginx. Finalmente, el contenedor expone el puerto 80 y ejecuta el servidor web Nginx para servir la aplicación de forma eficiente.

1.2 Configuración del Pipeline DevSecOps

Para garantizar la automatización de tareas clave como el análisis de seguridad, la construcción de contenedores y las pruebas de integración, se implementó un pipeline centralizado en el repositorio raíz `myapp-segura`. Este pipeline, definido mediante GitHub Actions dentro de la carpeta `.github/workflows`, se encarga de orquestar todos los procesos necesarios para validar la integridad y seguridad de la aplicación completa, tanto del frontend como del backend.

La ejecución del pipeline se activa ante eventos de `push` y `pull_request` sobre la rama `main`, asegurando así que todo cambio en producción sea previamente analizado y verificado.

Fases del Pipeline Principal

1. **Análisis de seguridad con CodeQL**. Se incluyen dos jobs paralelos: `codeql-analysis-backend` y `codeql-analysis-frontend`, encargados de analizar el código del backend y del frontend respectivamente. Cada uno realiza el `checkout` del repositorio con submódulos, instala las dependencias necesarias con `npm install` (usando `--legacy-peer-deps` para compatibilidad), y ejecuta el análisis estático utilizando GitHub CodeQL. Esta

herramienta identifica vulnerabilidades conocidas y malas prácticas de programación, y sus resultados se registran en la pestaña de seguridad del repositorio.

2. **Construcción y pruebas con Docker**. Una vez finalizados los análisis de seguridad, el job `build-and-test` procede a construir la imagen Docker del backend. Posteriormente, instala `docker-compose` y levanta el entorno completo de la aplicación mediante el archivo `docker-compose.yml`, el cual orquesta tanto el servicio de frontend como el backend en contenedores aislados. Para validar el correcto funcionamiento, se realiza una solicitud HTTP POST al endpoint `/auth/login` tras un retardo controlado, asegurando que el backend está operativo. Si la respuesta es exitosa, los servicios se detienen correctamente, y la imagen generada se empaqueta como artefacto (`.tar`) y se publica en GitHub Actions para su descarga o despliegue posterior.

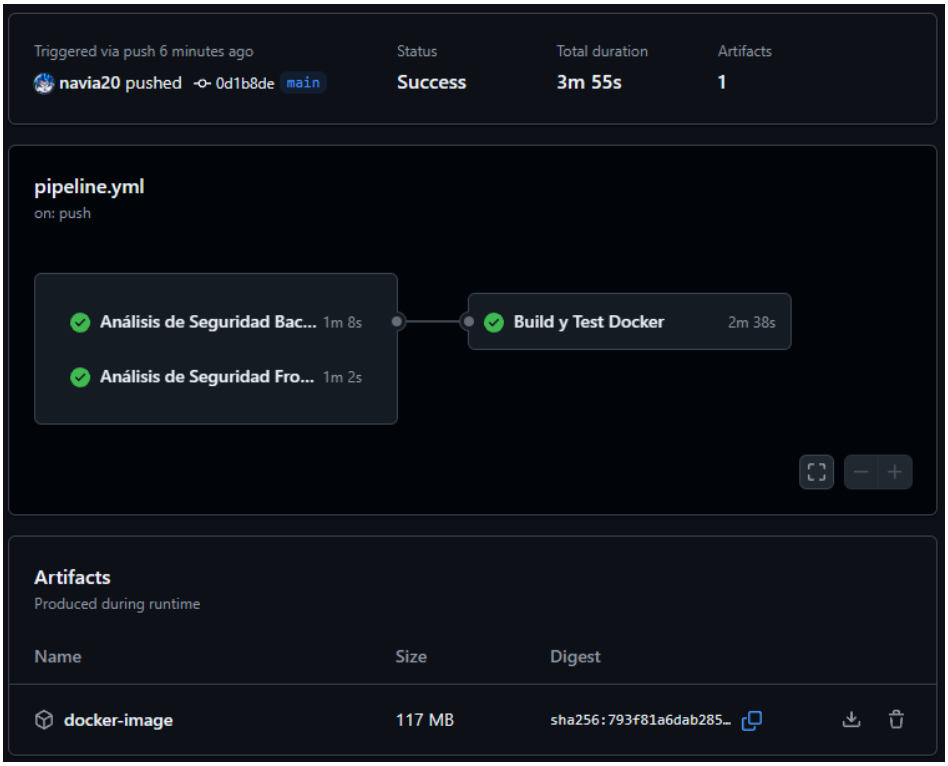

Figure 1: Evidencias de análisis de seguridad de Frontend y Backend con herramienta CodeQL y build con test Docker creando Imagen. *https://github.com/EduM1randa/myapp-segura/actions/runs/15797990072*

Consideraciones de Diseño

Este pipeline centralizado responde a la necesidad de validar el sistema completo en un solo flujo, garantizando coherencia entre los módulos y reduciendo la complejidad operativa. Se optó por centralizar la ejecución en el repositorio `myapp-segura`, ya que contiene como

submódulos tanto al frontend como al backend, permitiendo una orquestación integral del proyecto.

- **Shift-left de seguridad**: el análisis de CodeQL se realiza al comienzo para detener la ejecución ante vulnerabilidades críticas.
- **Construcción reproducible**: la instrucción `--no-cache` en el build de Docker asegura que cada ejecución parta de un entorno limpio, evitando resultados inconsistentes.
- **Validación funcional**: la prueba con `curl` simula una petición real al servidor, verificando que los contenedores no solo se construyen, sino que responden de forma efectiva.
- **Portabilidad de artefactos**: guardar la imagen Docker como archivo `.tar` permite su transporte y reutilización en ambientes sin acceso directo al repositorio.

1.3 Integración con SonarCloud

Con el objetivo de fortalecer la seguridad y la calidad del código fuente de forma automatizada, se implementó el análisis estático mediante la plataforma **SonarCloud** tanto en el repositorio del `backend` como en el del `frontend`. Esta integración se realizó por separado debido a que ambos repositorios son independientes y externos al propietario del repositorio raíz `myapp-segura`, lo cual impidió realizar la integración desde un único pipeline centralizado. En ambos casos, se utilizó el **workflow oficial de SonarCloud para GitHub Actions**, ubicado en la carpeta `.github/workflows`, el cual se ejecuta automáticamente ante eventos de `push` y `pull_request` sobre la rama principal. Cada workflow consta de tres fases principales: instalación de dependencias, ejecución de pruebas con cobertura, y escaneo del código con SonarCloud.

Backend

En el repositorio del backend, se configuró un pipeline específico que realiza el análisis de código utilizando SonarCloud. El flujo de trabajo incluye el `checkout` del código, la instalación de dependencias mediante `npm install`, y la ejecución de pruebas unitarias con `npm run test -- --coverage`, permitiendo así la generación del reporte de cobertura de código. Finalmente, se ejecuta el comando de escaneo utilizando el `token de autenticación` almacenado en los secrets del repositorio bajo la clave `SONAR_TOKEN`.

Frontend

De forma análoga, en el repositorio del frontend también se integró SonarCloud mediante un workflow personalizado. En este caso, se especificó de manera explícita el path del archivo de cobertura `lcov.info` generado por la herramienta de pruebas de React, utilizando el parámetro `-Dsonar.javascript.lcov.reportPaths`. Todo el análisis se ejecuta desde el subdirectorio `frontend-devsecops`, mediante la configuración del parámetro `working-directory` en cada paso del workflow.

A continuación se presenta el análisis comparativo entre los módulos **Backend** y **Frontend** del proyecto DevSecOps, evaluados mediante la herramienta SonarCloud.

Métrica	**Backend**	**Frontend**
Quality Gate	**Failed** (SonarWay)	**Passed** (SonarWay)
Vulnerabilidades	0 o muy bajas	0
Bugs	0	0
Code Smells	Algunos presentes	0
Security Hotspots	0	0
Cobertura de código	No configuradas	No configuradas
Duplicaciones	Mínimas / controladas	No configuradas
Mantenibilidad	Aceptable	Excelente (0 issues)
Fiabilidad	Buena	Muy buena

Table 1: Comparativa de métricas entre backend y frontend.

De acuerdo a las metricas se puede decir que en el Backend si la cobertura está por debajo del umbral mínimo (por ejemplo, menos del 80% en nuevo código), el Quality Gate falla, pero como no se esta midiendo el Coverage del codigo y la herramienta obligatoriamente mide el Coverage, este error no se va a tomar en cuenta ya que no se esta haciendo testing al codigo. En general las metricas arrojan valores positivos a la hora de analizar los repositorios, Ambos proyectos están libres de errores graves, sin bugs ni vulnerabilidades, El Quality Gate aprobado indica buena base DevSecOps, y por ultimo la Mantenibilidad y fiabilidad están sólidas en ambos lados, con notas "A"

2 Evidencias

2.1 SonarCloud

El análisis completo del frontend y Backend se encuentra disponible en SonarCloud:

https://sonarcloud.io/summary/overall?id=navia20_frontend-devsecops&branch=main

https://sonarcloud.io/summary/overall?id=navia20_backend-devsecops&branch=main

2.2 Docker

Intefáz de DockerDesktop en donde se puede visualizar el contenedor myapp-segura ejecutando tanto el contenedor del backend como el del frontend.

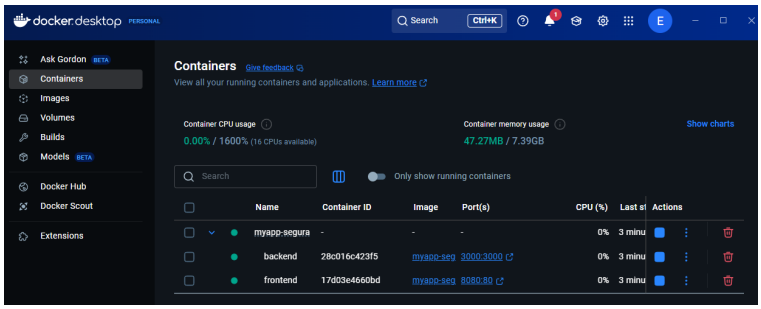

Figure 2: Contenedores corriendo exitosamente al buildear docker-compose

Logs del contenedor del backend funcionando.

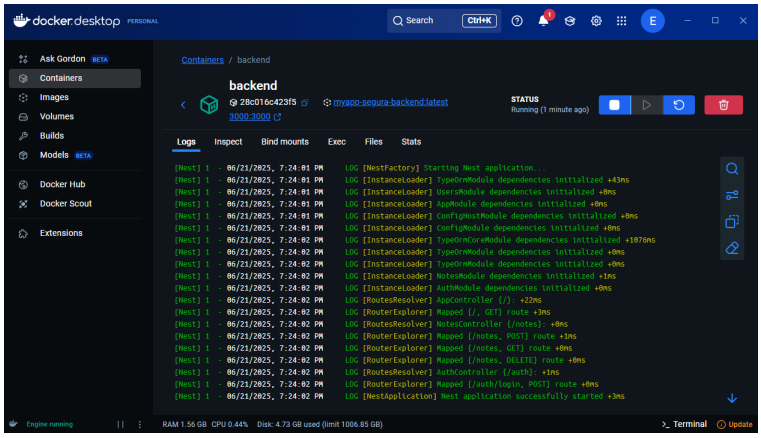

Figure 3: Logs contenedor backend

Logs del contenedor del forntend funcionando.

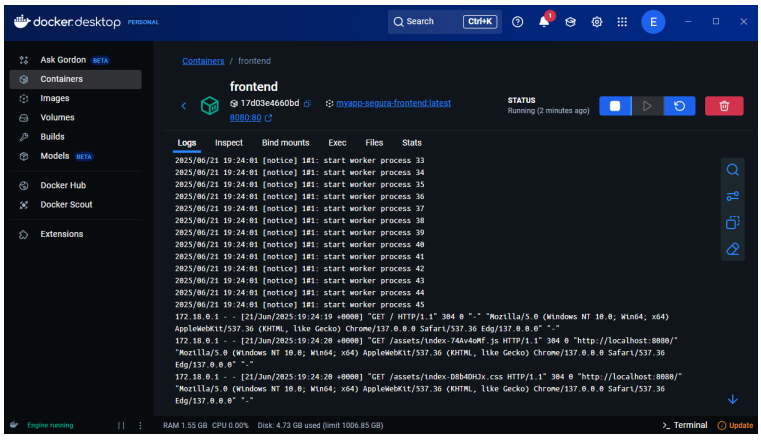

Figure 4: Logs contenedor frontend

2.3 Ejecución App

Login de la aplicación funcionando luego del build con Docker.

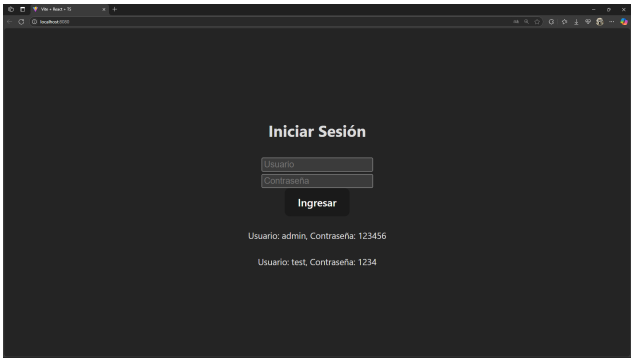

Figure 5: Aplicación Funcionando

3 Conclusión

De este taller, se vió y aprendió por medio de "ensayo y error" cómo integrar herramientas de automatización y análisis de calidad en un pipeline DevSecOps para aplicaciones web. Me quedó claro cómo la contenedorización con Docker facilita el despliegue y la portabilidad de los servicios, y cómo GitHub Actions permite automatizar procesos como pruebas, análisis de seguridad y construcción de imágenes. También la importancia del análisis estático con SonarCloud para mejorar la calidad del código y detectar vulnerabilidades de forma temprana.

Entre las dificultades encontradas estuvo entender la integración de múltiples herramientas y procesos en un pipeline cohesivo, especialmente al manejar configuraciones específicas para cada servicio y la gestión de secretos para la autenticación. También fue un desafío comprender cómo coordinar los análisis de seguridad y las fases de construcción y prueba en diferentes repositorios y pipelines independientes.

Para futuras mejoras, sería útil automatizar aún más la integración de los despliegues en ambientes de producción con pasos de validación adicionales y mayor control sobre las versiones. Además, implementar una documentación más detallada del proceso y de las configuraciones facilitaría su mantenimiento y escalamiento. También sería beneficioso incorporar análisis de vulnerabilidades en tiempo real durante la ejecución del entorno en producción, para detectar eventuales riesgos de manera proactiva.

En resumen, este ejercicio reforzó la importancia de la automatización, seguridad y calidad en el desarrollo de software, y evidenció áreas clave para optimizar aún más los procesos en un entorno DevSecOps.

4 Anexo

https://github.com/EduM1randa/myapp-segura