Ingeniería de Rendimiento en CI/CD Local: Zero-Bloat al Extremo

Estantería: Ingeniería y DevSecOps

Subtema: Rendimiento (Performance)

mercedev.es — 2026-09-04 | Epic 10 - Fase 3

El Desafío (Síntoma)

En el desarrollo de pipelines de **Integración Continua (CI)** locales, la velocidad de ejecución es crítica. El orquestador estático `merci total` empleaba **16.16 segundos** en ejecutarse debido a graves bloqueos de Entrada/Salida (I/O Blocking), importación global de librerías mastodónticas que no se llegaban a usar, iteraciones secuenciales de red y bucles de **Expresiones Regulares (Regex)** poco optimizados. Esta latencia rompía por completo el estado de flujo (flow) de la desarrolladora en cada guardado.

La Maniobra (Lógica)

Se aplicaron cuatro técnicas quirúrgicas en los scripts de validación, consiguiendo reducir el tiempo de compilación estático en más del 50% sin añadir ni una sola dependencia externa (Zero-Bloat).

1. Tree Pruning (Poda de Árboles) en auditoría: En lugar de iterar a ciegas con `rglob`, forzamos a `os.walk` a saltarse directorios colosales mediante modificación "in-place". El *auditor* pasó de escanear 14.000 archivos a solo 400 en milisegundos.

```
 for root, dirs, files in os.walk(root_dir):
    dirs[:] = [d for d in dirs if d not in {'.venv', 'node_modules',
'.git'}]
```

2. Carga Diferida (Lazy Loading) en publicador: Librerías pesadas como `weasyprint` (Generación de PDF Estático - **SSG**) añaden 0.45s solo por importarse. Ahora solo se importan dentro del bloque condicional que requiere forzosamente construir el PDF, saltándose el peaje si hay "Cache Hit".

3. Patrones Compuestos O(1) (Mega-Regex): En lugar de lanzar 120 acrónimos contra 400 archivos en un bucle O(N×M), se compiló un único gran patrón `re.compile(r'\b(API|CI|...)\b')`. Se resolvió en una sola pasada.

4. Native Concurrency (Concurrencia de Red) en Linkcheck: El script de enlaces perdía el 90% del tiempo esperando que el servidor **Nginx** contestase peticiones secuenciales. Al envolver la petición en un `ThreadPoolExecutor` de 50 hilos nativos, el tiempo bajó de 4.15s a 1.2s solapando toda la latencia.

El Aprendizaje / Deuda Técnica

Aplicar **Ingeniería de Confiabilidad del Sitio (SRE)** a nivel local, sin recurrir a paquetes NPM, requiere exprimir la biblioteca estándar de Python y domar la asincronía. El resultado ha sido la purga completa de cuellos de botella, cerrando el pipeline en **6.57s**. La deuda técnica pendiente pasa por la observabilidad extrema de futuras regresiones de tiempo.

💡 En resumen (Merci Explica):

Imagina que para revisar tu casa, en lugar de mirar en cada caja del trastero una por una (16 segundos), le dices al sistema: "Ignora el trastero, abre solo las habitaciones usadas hoy y manda a 50 personas a mirar a la vez". Ahora la revisión tarda 6 segundos y no gasta batería.