php-multiversion-wiki

Cómo compilar y mantener una imagen Docker PHP multiversión

Referencia operativa del repo `gitlab-ci-pipeline-php` (fork de `edbizarro/gitlab-ci-pipeline-php`), que genera la imagen `abkrim/laravel-gitlab-ci` usada como base de CI para proyectos Laravel. La imagen soporta varias versiones de PHP a la vez (hoy 8.3/8.4/8.5) y cinco variantes por versión (`default`, `alpine`, `fpm`, `fpm-min`, `chromium`).

Repo: `~/development/dockers/gitlab-ci-pipeline-php` · GitLab: `abkrim/gitlab-ci-pipeline-php`

Estructura del repo

El repo es un **generador de Dockerfiles**, no código de aplicación:

- `php/<version>/Dockerfile` — variante Debian por defecto (`FROM php:X.Y`)
- `php/<version>/{alpine,fpm,fpm-min,chromium}/Dockerfile` — variantes. `alpine` usa scripts propios, `fpm-min` usa un set reducido, `chromium` añade Chromium para Laravel Dusk
- `php/scripts/*.sh` — scripts **compartidos** entre todas las versiones Debian (paquetes, extensiones PHP, Node+pnpm). Un cambio aquí afecta a todas las versiones a la vez
- `php/scripts/alpine/*.sh` — el mismo patrón, para Alpine
- `tests/goss-<version>.yaml` — aserciones `goss` (binarios, extensiones PHP cargadas, versiones). **No corren en CI** — solo en verificación manual local, lo que significa que gaps entre el Dockerfile y el goss pueden pasar desapercibidos durante mucho tiempo (ver más abajo)

Cómo añadir una versión de PHP nueva

1. Copiar la carpeta de la versión anterior más reciente (`php/8.4/` → `php/8.5/`), sustituyendo el número de versión en cada `Dockerfile` (`FROM php:X.Y*`, `PHP="X.Y"`, `ENV PHP_VERSION=X.Y`)
2. Copiar `tests/goss-<anterior>.yaml` → `tests/goss-<nueva>.yaml`, actualizando la aserción `php -v`
3. Añadir las 5 entradas nuevas a la matriz de `.gitlab-ci.yml` (bloque `.variant_matrix`) y decidir si la nueva versión pasa a ser `:latest` (mover `IS_LATEST: 'true'` de la fila anterior a la nueva)
4. **Antes de dar nada por bueno, arrancar la imagen base pelada** y comparar contra lo que los scripts asumen:

```
docker run --rm php:X.Y sh -c 'php -v; php -m'
```

Esto habría detectado el problema de OPcache descrito abajo en segundos, en vez de a los 130s de un build fallido.

5. Build local (mucho más rápido que iterar por CI — sin cola de runner, sin QEMU si tu máquina ya es la arquitectura objetivo):

```
docker build -f php/X.Y/Dockerfile -t local/phpXY-debug:test .
docker run --rm -t -v "$PWD":/var/www/html local/phpXY-debug:test goss -g tests/goss-X.Y.yaml
v
```

6. Solo cuando el build local + goss pasan, subir y dejar que el pipeline de GitLab valide el build multiarch real (`build:mr` en una MR, sin push; `build` en `master`, con push a Docker Hub)

Trampas conocidas (con causa raíz, no solo el parche)

OPcache embebido en el core desde PHP 8.5

La imagen oficial `php:8.5*` ya trae OPcache **compilado dentro del core** — `php -v` lo muestra en la misma línea de versión (`with Zend OPcache v8.5.x`), y `/usr/src/php/ext/opcache/` ni existe tras `docker-php-source extract`. Intentar `docker-php-ext-install opcache` contra esa base falla con:

```
cp: cannot stat 'modules/*': No such file or directory
make: *** [Makefile:89: install-modules] Error 1
```

`configure` corre "bien" pero no hay nada que compilar — el fallo llega solo al final del `make install-modules`, después de haber compilado sin problema el resto de extensiones anteriores en la lista (por eso el build muere a los ~130s en vez de fallar inmediatamente).

Solución aplicada: en `php/scripts/extensions.sh` y `php/scripts/alpine/extensions.sh`, comprobar dinámicamente antes de instalar:

```
if ! php -m | grep -qi 'opcache'; then
  export extensions="$extensions opcache"
fi
```

Importante: `php -m` lista la entrada como **`Zend OPcache`**, no `opcache` a secas — un primer intento con `grep -qi '^opcache$'` (anclado) no matcheaba nunca. El nombre del binario/paquete y el nombre que aparece en `php -m` no siempre coinciden (Redis sí aparece como `redis`; OPcache no).

La comprobación es dinámica en vez de un `if PHP_VERSION == 8.5`, para que si una versión futura hace lo mismo con otra extensión, el patrón siga valiendo sin tocar código.

Alpine y corepack (PHP 8.4, Alpine 3.24+)

Node dejó de distribuir `corepack` a partir de Node 25, y Alpine no lo empaqueta en ningún paquete propio. La cadena `corepack enable && corepack prepare pnpm@latest --activate` sigue siendo correcta en Debian (NodeSource sí trae corepack), pero en Alpine hay que instalar pnpm directamente:

```
npm install -g pnpm@latest
```

`set -e` no aborta una cadena `a && b` a mitad

En un script con `set -euo pipefail`, si `pecl install xdebug && docker-php-ext-enable xdebug` falla en el primer comando, la lista cortocircuita con estado no-cero **y el script continúa** — `set -e` solo dispara sobre el comando tras el **último** `&&` de una lista. Si el paso intermedio no puede fallar en silencio, cada comando va en su propia línea, no encadenado con `&&`.

Un rebuild programado (`schedule`) que falla no avisa a nadie

GitHub Actions solo notifica fallos de `schedule` por email al último autor del workflow, y el schedule se autodesactiva tras 60 días de repo inactivo. Una imagen que deja de reconstruirse en rojo sigue funcionando en caché — nadie lo nota hasta una auditoría manual. En GitLab, la migración añadió un job `notify-*-failure` con `when: on_failure` que postea a un chat de Telegram dedicado cuando `security-rebuild`/`monthly-rebuild` fallan.

Verificación antes de dar un build por bueno

No basta con que el `RUN` termine sin error — comprobar con `goss` que lo que se esperaba instalar realmente está ahí (esto fue lo que destapó, de paso, que la variante `alpine` nunca crea `$HOME/.config` — un gap presente desde PHP 7.3 hasta 8.4 que nadie había notado porque el CI no corre goss, solo el build):

```
docker run --rm -t -v "$PWD":/var/www/html <imagen> goss -g tests/goss-X.Y.yaml v
```

Referencias

- Repo: `~/development/dockers/gitlab-ci-pipeline-php`
- `CLAUDE.md` del repo — sección "Lecciones aprendidas", con el detalle completo fecha por fecha de cada trampa encontrada
- Ticket de migración a GitLab + PHP 8.5: Linear ADE-441

Revision #1
Created 2026-08-04 04:25:06 UTC by Abkrim
Updated 2026-08-04 04:25:06 UTC by Abkrim