ŚCIĄGA LARAVEL

Docker na produkcji GitHub Actions

Build 3 obrazów → push do GHCR → deploy przez SSH

TL;DR

- Zbuduj **osobne obrazy**: node builder → php-fpm → nginx.
- Wypchnij do registry, wdróż przez SSH uruchamiając skrypt na serwerze.
- CI dostaje **tożsamość** (`GITHUB_TOKEN` , klucz deploy), nie Twoje dane.
- Sekrety produkcyjne jako `vars` / `secrets` , nigdy w obrazie.

Topologia obrazów

```
node   →  npm ci && npm run build  →  public/build
```

Używany jako build-context dla kolejnych obrazów.

```
php    →  composer --no-dev, kod z --chown, non-root, supervisord
```

`EXPOSE 9000`. Assety kopiowane z obrazu `node`.

```
nginx  →  serwuje public/ + assety z node, opcjonalny basic-auth (ARG)
```

docker/php/Dockerfile — istota

```
FROM twoj/php-8.4-fpm-alpine:latest
RUN addgroup -g 1000 appuser && adduser -D -u 1000 -G appuser appuser
COPY --from=composer:latest /usr/bin/composer /usr/bin/composer
COPY composer.json composer.lock ./
RUN composer install --no-dev --optimize-autoloader \
    --no-interaction --prefer-dist --no-scripts
COPY --chown=appuser:appuser . .
COPY --chown=appuser:appuser --from=node /var/www/public/build /var/www/public/build
USER appuser
CMD ["/usr/bin/supervisord", "-n", "-c", "/etc/supervisord.conf"]
```

Sekrety repozytorium

```
SSH_HOST / SSH_USER / SSH_PORT   →  adres, użytkownik deployer, port
SSH_KEY                          →  prywatny klucz deploy (osobny, nie osobisty)
ENV_FILE (jako vars)             →  zawartość produkcyjnego .env
```

Workflow — build: login do GHCR

```yaml
env:
  REGISTRY: ghcr.io
  PHP_IMAGE_NAME: twoj/laravel-production-php
  DOCKER_BUILDKIT: 1

jobs:
  build:
    runs-on: ubuntu-latest
    permissions: { contents: read, packages: write }   # GHCR push
    steps:
      - uses: actions/checkout@v4
      - uses: docker/setup-buildx-action@v3
      - uses: docker/login-action@v3
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}      # nie PAT
```

Workflow — build: push obrazu

```
      - run: |
          mkdir -p docker/node docker/php
          echo "${{ vars.ENV_FILE }}" > docker/php/.env

      - uses: docker/build-push-action@v5     # node → php → nginx
        with:
          context: .
          file: docker/php/Dockerfile
          push: true
          tags: ${{ env.REGISTRY }}/${{ env.PHP_IMAGE_NAME }}:latest
          cache-from: type=gha
          cache-to: type=gha,mode=max
          build-contexts: |
            node=docker-image://${{ env.REGISTRY }}/twoj/…-node:latest
```

`cache-from/to: type=gha` skraca kolejne buildy. Buduj **raz**, promuj ten sam artefakt.

Workflow — deploy: SSH agent

```
deploy:
  needs: build
  runs-on: ubuntu-latest
  timeout-minutes: 15
  steps:
    - uses: actions/checkout@v4
    - uses: webfactory/ssh-agent@v0.9.1
      with: { ssh-private-key: ${{ secrets.SSH_KEY }} }
    - run: ssh-keyscan -p ${{ secrets.SSH_PORT }} \
           ${{ secrets.SSH_HOST }} >> ~/.ssh/known_hosts
```

`ssh-keyscan` do `known_hosts` zamiast `StrictHostKeyChecking=no`.

Workflow — deploy: transfer + trigger

```
      - run: |
          echo "${{ vars.ENV_FILE }}" > .env
          { echo "REGISTRY=${{ env.REGISTRY }}"; echo "TAG=latest"; } >> .env
      - run: |
          scp -P $PORT docker-compose.production.yml \
            $USER@$HOST:~/laravel/docker-compose.yml
          scp -P $PORT .env deploy.sh $USER@$HOST:~/laravel/
      - run: |
          ssh -p $PORT $USER@$HOST \
            "cd ~/laravel && chmod +x deploy.sh && ./deploy.sh"
```

Zmienne `$PORT/$USER/$HOST` = `secrets.SSH_*`. Migracje jako nazwany krok w `deploy.sh`, nie w obrazie.

deploy.sh na serwerze

```
set -eo pipefail
docker compose pull
docker compose up -d --force-recreate       # patrz uwaga
docker compose exec -T app php artisan migrate --force
docker compose exec -T app php artisan optimize
docker compose exec -T app php artisan storage:link
```

`down` / `up` = kilka sekund przestoju — dla zero-downtime wprowadź nowy tag obok starego i przełącz ruch. `docker system prune --volumes -f` w deployu **usuwa nazwane wolumeny** — nie uruchamiaj bezwarunkowo.

Częste problemy

- „Permission denied" → `--chown` w `COPY`, uprawnienia `storage/` i `bootstrap/cache` w obrazie.
- Assety 404 → build z obrazu `node` nie trafił do php/nginx (sprawdź `build-contexts`).
- Deploy „przechodzi", ale stary kod → kontener nie zrestartowany albo cache configu z poprzedniego wydania.

PODSUMOWANIE

Ściąga w skrócie

- → 3 obrazy: node builder → php → nginx
- → Login: `GITHUB_TOKEN` + `permissions: packages: write`
- → `cache-from/to: type=gha`, buduj raz, promuj artefakt
- → Deploy: `ssh-agent` + `ssh-keyscan`, nie `StrictHostKeyChecking=no`
- → Migracje = nazwany krok w `deploy.sh`
- → `down/up` = przestój; `prune --volumes` = ryzyko utraty danych

Przydało się?

Zapisz tę ściągę na później.

Pełny artykuł: dominik-dev.pl

Udostępnij • Skomentuj