LARAVEL CHEAT SHEET

# Docker in production
# GitHub Actions

Build 3 images → push to GHCR → deploy over SSH

# TL;DR

- Build **separate images**: node builder → php-fpm → nginx.
- Push to a registry, deploy over SSH by running a script on the server.
- CI gets an **identity** ( `GITHUB_TOKEN` , a deploy key), not your credentials.
- Production secrets as `vars` / `secrets` , never in the image.

# Image topology

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

Used as a build-context for the next images.

```
php    →  composer --no-dev, code with --chown, non-root, supervisord
```

`EXPOSE 9000`. Assets copied from the `node` image.

```
nginx  →  serves public/ + assets from node, optional basic-auth (ARG)
```

# docker/php/Dockerfile — the essence

```
FROM your/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"]
```

# Repository secrets

```
SSH_HOST / SSH_USER / SSH_PORT   →  address, the deployer user, port
SSH_KEY                          →  the deploy private key (separate, not personal)
ENV_FILE (as vars)               →  the contents of the production .env
```

# Workflow — build: login to GHCR

```yaml
env:
  REGISTRY: ghcr.io
  PHP_IMAGE_NAME: your/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 }}      # not a PAT
```

# Workflow — build: push the image

```
- 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 }}/your/…-node:latest
```

`cache-from/to: type=gha` shortens later builds. Build **once**, promote the same artifact.

# 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` into `known_hosts` rather than `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"
```

`$PORT/$USER/$HOST` = `secrets.SSH_*`. Migrations as a named step in `deploy.sh`, not in the image.

# deploy.sh on the server

```
set -eo pipefail
docker compose pull
docker compose up -d --force-recreate       # see the note
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` = a few seconds of downtime — for zero-downtime, roll out a new tag alongside the old one and switch traffic. `docker system prune --volumes -f` in a deploy script **removes named volumes** — don't run it unconditionally.

# Common problems

- "Permission denied" → `--chown` in `COPY`, `storage/` and `bootstrap/cache` permissions in the image.
- Assets 404 → the build from the `node` image didn't reach php/nginx (check `build-contexts`).
- Deploy "passes" but old code → container not restarted, or a config cache from the previous release.

RECAP

# The sheet in short

- → 3 images: node builder → php → nginx
- → Login: `GITHUB_TOKEN` + `permissions: packages: write`
- → `cache-from/to: type=gha`, build once, promote the artifact
- → Deploy: `ssh-agent` + `ssh-keyscan`, not `StrictHostKeyChecking=no`
- → Migrations = a named step in `deploy.sh`
- → `down/up` = downtime; `prune --volumes` = data-loss risk

# Found this useful?

Save this cheat sheet for later.

Full article: dominik-dev.pl

**Share • Comment**