Docker Deployment Cheat Sheet

Laravel Blog on Hetzner VPS with Cloudflare, Traefik, Apache, MySQL, and GitHub Actions

```
Browser
  -> Cloudflare DNS + Proxy
  -> Hetzner VPS
  -> Traefik container
  -> Laravel Apache container
  -> MySQL container
```

A read-to-learn guide based on your kurdibuilds.dev deployment. It explains what each part does, why it exists, how the files connect, and how to debug the common problems you already saw.

Contents

How to use this guide

This is a read-to-learn cheat sheet for your kurdibuilds.dev deployment. It explains the idea first, then gives the exact files and commands.

The goal is not just to copy commands. The goal is to understand what each layer does, where each file lives, and how to debug the common errors you already hit.

Your final architecture:

```
Browser
  -> Cloudflare DNS + Proxy
  -> Hetzner VPS
  -> Traefik container
  -> Laravel Apache container
  -> MySQL container
```

The most important rule:

```
Only Traefik is public.
Laravel, MySQL, Redis, queues, and future app services stay private inside Docker networks.
```

1. Mental model: what you built

Your deployment is a single-server Docker hosting setup. It is small enough to understand, but it uses the same ideas used in larger production systems: a reverse proxy, isolated app containers, private service networks, persistent volumes, environment variables, and automated deployment.

A request flows like this:

```
User visits https://kurdibuilds.dev
  -> Cloudflare receives the browser request
  -> Cloudflare forwards to your Hetzner VPS
  -> Traefik receives the request on port 443
  -> Traefik reads Docker labels and finds the Laravel container
  -> Traefik forwards internally to kurdibuilds_app:80
  -> Apache serves Laravel from /var/www/html/public
  -> Laravel talks to MySQL at db:3306
```

What each layer does

```
Component | Public? | Role
Cloudflare | Yes | DNS, proxy, edge HTTPS, basic protection.
Hetzner VPS | Yes | Owns the server IP. Firewall allows only SSH, HTTP, and HTTPS.
Traefik | Yes | The only container that publishes ports 80 and 443.
Laravel Apache container | No | Receives traffic only from Traefik through Docker network.
MySQL container | No | Receives traffic only from Laravel through the internal Docker network.
```

Why Docker makes sense for your VPS

You want to host multiple apps on one server. Docker helps because each app can keep its own PHP version, Node version, database, services, and environment without fighting other apps.

Docker is especially useful when:

- One app needs PHP 8.4 and another needs a different runtime.
- One app needs MySQL and another needs PostgreSQL.
- You want to deploy multiple domains and subdomains to the same VPS.
- You want to rebuild one app without touching the others.
- You want future deployments to be repeatable.

2. Server layout and Docker networks

Folder structure

Recommended structure:

```
/opt/stacks/
├── traefik/
│   ├── docker-compose.yml
│   └── acme.json
│
├── apps/
│   └── kurdibuilds-blog/
│       ├── Dockerfile
│       ├── apache.conf
│       ├── docker-compose.prod.yml
│       ├── .env
│       ├── .dockerignore
│       ├── deploy.sh
│       └── Laravel app files
│
└── backups/
    └── kurdibuilds-blog/
```

Why this is good:

```
/opt/stacks/traefik = global reverse proxy
/opt/stacks/apps    = all hosted apps
/opt/stacks/backups = database and file backups
```

The proxy network

The proxy network is the shared road between Traefik and web-facing app containers.

Create it once:

```
docker network create proxy
```

Traefik joins this network. Your Laravel app joins this network. Future web apps also join this network.

MySQL does not need this network because MySQL should not be reachable from the web routing layer.

The internal network

Each app stack can also have a private internal network. Your Laravel app and MySQL share this network.

```
networks:
  proxy:
    external: true
  internal:
    internal: true
```

Meaning:

- proxy is shared across stacks.
- internal belongs only to this app stack.
- MySQL is reachable by Laravel, not by Traefik.

Why traefik.docker.network=proxy mattered

Your Laravel app is connected to two networks:

```
proxy
internal
```

When a container has multiple networks, Traefik might choose the wrong internal IP. This can cause Gateway Timeout.

The fix is this label:

```
- "traefik.docker.network=proxy"
```

That tells Traefik: use the proxy network to reach this app.

3. Public ports rule

Only Traefik should publish public ports:

```
ports:
  - "80:80"
  - "443:443"
```

Do not publish MySQL:

```
# Do not do this
ports:
  - "3306:3306"
```

If docker ps shows this for MySQL, it is safe:

```
3306/tcp, 33060/tcp
```

It means MySQL is listening inside Docker only.

If it shows this, it is dangerous:

```
0.0.0.0:3306->3306/tcp
```

That means MySQL is exposed publicly on the VPS.

4. Traefik: the public front door

Traefik is your reverse proxy. It receives requests from the internet and forwards them to the correct container based on domain names.

Traefik watches Docker and reads labels from containers. That is why app-specific routing lives in the app Compose file.

Traefik Compose file

Location:

```
/opt/stacks/traefik/docker-compose.yml
```

Example:

```
services:
  traefik:
    image: traefik:v3
    container_name: traefik
    restart: unless-stopped

    command:
      - "--api.dashboard=true"
      - "--providers.docker=true"
      - "--providers.docker.exposedbydefault=false"
      - "--entrypoints.web.address=:80"
      - "--entrypoints.websecure.address=:443"
      - "--entrypoints.web.http.redirections.entrypoint.to=websecure"
      - "--entrypoints.web.http.redirections.entrypoint.scheme=https"
      - "--certificatesresolvers.letsencrypt.acme.email=YOUR_EMAIL@example.com"
      - "--certificatesresolvers.letsencrypt.acme.storage=/letsencrypt/acme.json"
      - "--certificatesresolvers.letsencrypt.acme.httpchallenge=true"
      - "--certificatesresolvers.letsencrypt.acme.httpchallenge.entrypoint=web"
      - "--log.level=INFO"
      - "--accesslog=true"

    ports:
      - "80:80"
      - "443:443"

    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock:ro"
      - "./acme.json:/letsencrypt/acme.json"

    networks:
      - proxy

networks:
  proxy:
    external: true
```

Important settings explained

```
Setting | Meaning
`providers.docker=true` | Traefik reads Docker containers and labels.
`exposedbydefault=false` | Containers are not public unless you add `traefik.enable=true`.
`entrypoints.web.address=:80` | HTTP traffic enters on port 80.
`entrypoints.websecure.address=:443` | HTTPS traffic enters on port 443.
`httpchallenge.entrypoint=web` | Let's Encrypt verifies domains through HTTP on port 80.
Docker socket mounted read-only | Traefik can inspect containers but not write through the socket.
```

Start and restart Traefik

```
cd /opt/stacks/traefik
docker compose up -d

docker compose restart traefik

# From anywhere:
docker restart traefik
```

acme.json

Traefik stores Let's Encrypt certificates in this file:

```
/opt/stacks/traefik/acme.json
```

Create it:

```
cd /opt/stacks/traefik
touch acme.json
chmod 600 acme.json
```

Check if a domain exists inside it:

```
cat /opt/stacks/traefik/acme.json | grep kurdibuilds.dev
```

Better certificate check:

```
echo | openssl s_client -servername kurdibuilds.dev -connect kurdibuilds.dev:443 2>/dev/null | openssl x509 -noout -i
  ssuer -subject -dates
```

Good result includes an issuer like Let's Encrypt.

5. Laravel on Apache: the app container

Your Laravel app runs in a PHP Apache container. Apache listens on port 80 inside the container. Traefik forwards traffic to that internal port.

Apache must serve Laravel from the public directory, not from the project root.

Dockerfile

Location:

```
/opt/stacks/apps/kurdibuilds-blog/Dockerfile
```

Example:

```
FROM php:8.4-apache

RUN apt-get update && apt-get install -y \
    git \
    unzip \
    zip \
    curl \
    libcurl4-openssl-dev \
    libzip-dev \
    libpng-dev \
    libjpeg-dev \
    libfreetype6-dev \
    libonig-dev \
    libxml2-dev \
    libicu-dev \
    default-mysql-client \
    nodejs \
    npm \
    && docker-php-ext-configure gd --with-freetype --with-jpeg \
    && docker-php-ext-install \
        curl \
        pdo_mysql \
        mbstring \
        zip \
        bcmath \
        gd \
        intl \
        opcache \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

RUN a2enmod rewrite headers

COPY apache.conf /etc/apache2/sites-available/000-default.conf

COPY --from=composer:2 /usr/bin/composer /usr/bin/composer

WORKDIR /var/www/html

COPY . .

RUN composer install --no-dev --optimize-autoloader --no-interaction

RUN npm ci
RUN npm run build

RUN chown -R www-data:www-data /var/www/html/storage /var/www/html/bootstrap/cache

EXPOSE 80
```

Why PHP 8.4 mattered

Your Composer lock file included Laravel/Symfony packages that required PHP 8.4 or newer. When the Dockerfile used php:8.3-apache, Composer refused to install dependencies.

Lesson:

```
The PHP version inside the Docker image must satisfy composer.lock.
```

Apache config

Location:

```
/opt/stacks/apps/kurdibuilds-blog/apache.conf
```

Example:

```
ServerName kurdibuilds.dev

<VirtualHost *:80>
    ServerName kurdibuilds.dev
    ServerAlias www.kurdibuilds.dev

    DocumentRoot /var/www/html/public

    <Directory /var/www/html/public>
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Most important line:

```
DocumentRoot /var/www/html/public
```

Laravel must be served from the public folder.

Apache FQDN warning

This warning is not fatal:

```
AH00558: apache2: Could not reliably determine the server's fully qualified domain name
```

Adding this at the top of apache.conf suppresses it:

```
ServerName kurdibuilds.dev
```

But the warning itself does not cause Gateway Timeout.

6. .dockerignore

Location:

```
/opt/stacks/apps/kurdibuilds-blog/.dockerignore
```

Example:

```
.git
.github
node_modules
vendor
.env
.env.*
storage/logs/*
storage/framework/cache/*
storage/framework/sessions/*
storage/framework/views/*
```

Why this matters:

- The Docker image should not contain .env secrets.
- node_modules and vendor are rebuilt during Docker build.
- Old logs and cache files should not be baked into the image.

Important consequence:

```
.env is not copied into the image.
```

That is why php artisan key:generate failed inside the container. The command tried to write to /var/www/html/.env, but that file was intentionally not copied.

7. Laravel production Compose file

Location:

```
/opt/stacks/apps/kurdibuilds-blog/docker-compose.prod.yml
```

Example:

```
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    container_name: kurdibuilds_app
    restart: unless-stopped
    env_file:
      - .env
    depends_on:
      - db
    networks:
      - proxy
      - internal
    volumes:
      - app_storage:/var/www/html/storage
    labels:
      - "traefik.enable=true"
      - "traefik.docker.network=proxy"
      - "traefik.http.routers.kurdibuilds.rule=Host(`kurdibuilds.dev`) || Host(`www.kurdibuilds.dev`)"
      - "traefik.http.routers.kurdibuilds.entrypoints=websecure"
      - "traefik.http.routers.kurdibuilds.tls.certresolver=letsencrypt"
      - "traefik.http.services.kurdibuilds.loadbalancer.server.port=80"

  db:
    image: mysql:8.4
    container_name: kurdibuilds_db
    restart: unless-stopped
    environment:
      MYSQL_DATABASE: ${DB_DATABASE}
      MYSQL_USER: ${DB_USERNAME}
      MYSQL_PASSWORD: ${DB_PASSWORD}
      MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
    volumes:
      - db_data:/var/lib/mysql
    networks:
      - internal

volumes:
  db_data:
  app_storage:

networks:
  proxy:
    external: true
  internal:
    internal: true
```

Important labels

```
- "traefik.enable=true"
```

Makes this container visible to Traefik.

```
- "traefik.docker.network=proxy"
```

Forces Traefik to use the correct Docker network.

```
- "traefik.http.routers.kurdibuilds.rule=Host(`kurdibuilds.dev`) || Host(`www.kurdibuilds.dev`)"
```

Routes both root domain and www to this app.

```
- "traefik.http.services.kurdibuilds.loadbalancer.server.port=80"
```

Tells Traefik that Apache listens on port 80 inside the container.

8. Production .env

Location:

```
/opt/stacks/apps/kurdibuilds-blog/.env
```

Example:

```
APP_NAME="Kurdi Builds"
APP_ENV=production
APP_KEY=base64:YOUR_GENERATED_KEY_HERE
APP_DEBUG=false
APP_URL=https://kurdibuilds.dev

DB_CONNECTION=mysql
DB_HOST=db
DB_PORT=3306
DB_DATABASE=kurdibuilds
DB_USERNAME=laravel
DB_PASSWORD=your_strong_db_password
DB_ROOT_PASSWORD=your_strong_root_password

SESSION_DRIVER=database
CACHE_STORE=database
QUEUE_CONNECTION=database
```

Why DB_HOST=db

Inside Docker, each Compose service can reach another service by service name.

Your database service is named:

```
db:
```

So Laravel connects to:

```
DB_HOST=db
```

Wrong:

```
DB_HOST=127.0.0.1
```

Inside the Laravel container, 127.0.0.1 means the Laravel container itself, not the MySQL container.

Generate APP_KEY

Because .env is not copied into the image, generate the app key on the VPS and paste it into the VPS .env:

```
echo "base64:$(openssl rand -base64 32)"
```

Then set:

```
APP_KEY=base64:PASTE_GENERATED_KEY_HERE
```

Check what the app container sees:

```
docker exec kurdibuilds_app printenv APP_ENV
docker exec kurdibuilds_app printenv APP_KEY
docker exec kurdibuilds_app printenv DB_HOST
docker exec kurdibuilds_app printenv DB_DATABASE
```

9. Build, run, rebuild, restart

A Dockerfile is not a running thing. It is a recipe for building an image.

```
Thing | Meaning
Dockerfile | Build recipe for the Laravel app image.
Image | Built artifact created from the Dockerfile.
Container | Running instance of an image.
Volume | Persistent storage outside the container lifecycle.
Network | Private communication path between containers.
```

Build and start

```
cd /opt/stacks/apps/kurdibuilds-blog

docker compose -f docker-compose.prod.yml up -d --build
```

Clean rebuild of only the app

```
cd /opt/stacks/apps/kurdibuilds-blog

docker compose -f docker-compose.prod.yml build --no-cache app
docker compose -f docker-compose.prod.yml up -d --force-recreate app
```

Stop containers but keep data

```
docker compose -f docker-compose.prod.yml down
```

Stop containers and delete volumes

```
docker compose -f docker-compose.prod.yml down -v
```

Warning:

```
down -v deletes Docker volumes, including database data.
```

Do not run it in production unless you intentionally want to erase the database volume.

Laravel commands after deployment

```
docker exec kurdibuilds_app php artisan optimize:clear
docker exec kurdibuilds_app php artisan migrate --force
docker exec kurdibuilds_app php artisan config:cache
docker exec kurdibuilds_app php artisan route:cache
docker exec kurdibuilds_app php artisan view:cache
```

Fix permissions

```
docker exec kurdibuilds_app chown -R www-data:www-data /var/www/html/storage /var/www/html/bootstrap/cache
docker exec kurdibuilds_app chmod -R 775 /var/www/html/storage /var/www/html/bootstrap/cache
```

10. Vite assets: why CSS disappeared

Laravel with Vite expects this file in production:

```
/var/www/html/public/build/manifest.json
```

That file is created by:

```
npm run build
```

If it is missing, Laravel throws:

```
ViteManifestNotFoundException
```

Check Vite build output

```
docker exec kurdibuilds_app ls -la public/build
docker exec kurdibuilds_app ls -la public/build/manifest.json
docker exec kurdibuilds_app find public/build/assets -type f
```

Correct Dockerfile lines

The Dockerfile should include:

```
RUN npm ci
RUN npm run build
```

Then rebuild:

```
cd /opt/stacks/apps/kurdibuilds-blog

docker compose -f docker-compose.prod.yml build --no-cache app
docker compose -f docker-compose.prod.yml up -d --force-recreate app
```

11. HTTPS, Cloudflare, and mixed content

There are two HTTPS paths:

```
Browser -> Cloudflare
Cloudflare -> Hetzner VPS
```

Cloudflare's edge certificate protects the first path. Traefik's Let's Encrypt certificate protects the second path when Cloudflare is set to Full (strict).

Correct Cloudflare state

- A @ points to the Hetzner server IP.
- A www points to the Hetzner server IP.
- Proxy/orange cloud can be on after Let's Encrypt works.
- SSL/TLS mode should be Full (strict).
- Avoid Flexible for this setup.

Mixed content problem

The page loaded over HTTPS but requested assets over HTTP:

```
https://kurdibuilds.dev
requested http://kurdibuilds.dev/build/assets/app-XXXXX.js
```

Cause:

```
Traefik receives HTTPS from Cloudflare.
Traefik forwards to Apache over HTTP inside Docker.
Laravel may think the request is HTTP.
```

Fix in app/Providers/AppServiceProvider.php:

```
use Illuminate\Support\Facades\URL;

public function boot(): void
{
    if ($this->app->environment('production')) {
        URL::forceScheme('https');
    }
}
```

Also confirm:

```
APP_ENV=production
APP_URL=https://kurdibuilds.dev
```

Check for remaining HTTP asset URLs:

```
curl -s https://kurdibuilds.dev | grep -o "http://kurdibuilds.dev[^\"']*"
```

Good result: no output.

12. Deployment script and GitHub Actions

Your current automation is a basic CD pipeline.

Flow:

```
Push to main
  -> GitHub Actions starts
  -> GitHub SSHs into VPS as deploy user
  -> VPS runs deploy.sh
  -> deploy.sh pulls code, rebuilds, restarts, migrates, caches
```

deploy.sh

Location:

```
/opt/stacks/apps/kurdibuilds-blog/deploy.sh
```

Example:

```
#!/bin/bash
set -e

cd /opt/stacks/apps/kurdibuilds-blog

echo "Pulling latest code..."
git pull origin main

echo "Building app image..."
docker compose -f docker-compose.prod.yml build app

echo "Restarting app..."
docker compose -f docker-compose.prod.yml up -d --force-recreate app

echo "Running Laravel commands..."
docker exec kurdibuilds_app php artisan optimize:clear
docker exec kurdibuilds_app php artisan migrate --force
docker exec kurdibuilds_app php artisan config:cache
docker exec kurdibuilds_app php artisan route:cache
docker exec kurdibuilds_app php artisan view:cache

echo "Cleaning old images..."
docker image prune -f

echo "Deployment finished."
```

Make it executable:

```
chmod +x /opt/stacks/apps/kurdibuilds-blog/deploy.sh
```

Run manually:

```
cd /opt/stacks/apps/kurdibuilds-blog
./deploy.sh
```

GitHub Actions workflow

Repo file:

```
.github/workflows/deploy.yml
```

Example:

```
name: Deploy to Production

on:
  push:
    branches:
      - main

concurrency:
  group: production-deploy
  cancel-in-progress: true

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Deploy on VPS
        uses: appleboy/ssh-action@v1
        with:
          host: ${{ secrets.VPS_HOST }}
          username: ${{ secrets.VPS_USER }}
          key: ${{ secrets.VPS_SSH_KEY }}
          script: |
            /opt/stacks/apps/kurdibuilds-blog/deploy.sh
```

Required GitHub secrets:

```
Secret | Value
`VPS_HOST` | Your Hetzner server IP.
`VPS_USER` | `deploy`
`VPS_SSH_KEY` | Private SSH key that lets GitHub Actions log in as `deploy`.
```

Is this full CI/CD?

Not yet. This is automated deployment, or a basic CD pipeline.

A fuller CI/CD pipeline would also:

- Install Composer dependencies in GitHub Actions.
- Install Node dependencies in GitHub Actions.
- Run tests.
- Run code style checks.
- Build Docker image in GitHub Actions.
- Push the image to a registry.
- SSH into the VPS.
- Pull the image.
- Restart the container.
- Run migrations.
- Run a health check.

Your current setup is a good learning stage because it is understandable and useful.

13. Backups and restore basics

Backups should exist before automation. A deployment script can make changes quickly, so you need a way to recover the database.

Create backup folder

```
mkdir -p /opt/stacks/backups/kurdibuilds-blog
```

Backup using the root password from the MySQL container environment

```
docker exec kurdibuilds_db sh -c 'MYSQL_PWD="$MYSQL_ROOT_PASSWORD" mysqldump -uroot kurdibuilds' \
  > /opt/stacks/backups/kurdibuilds-blog/backup-$(date +%F-%H%M).sql
```

Check backup files

```
ls -lh /opt/stacks/backups/kurdibuilds-blog
head -n 20 /opt/stacks/backups/kurdibuilds-blog/backup-YYYY-MM-DD-HHMM.sql
```

If the password fails

Check the VPS .env:

```
cd /opt/stacks/apps/kurdibuilds-blog
grep DB_ .env
```

Check the MySQL container environment:

```
docker exec kurdibuilds_db printenv MYSQL_USER
docker exec kurdibuilds_db printenv MYSQL_PASSWORD
docker exec kurdibuilds_db printenv MYSQL_ROOT_PASSWORD
```

Important: MySQL uses these variables only when the database volume is first initialized. If you changed .env after the volume already existed, the live MySQL password may still be the old one.

If this is a new project and you can delete all DB data, this resets the DB volume:

```
cd /opt/stacks/apps/kurdibuilds-blog
docker compose -f docker-compose.prod.yml down -v
docker compose -f docker-compose.prod.yml up -d
docker exec kurdibuilds_app php artisan migrate --force
```

Warning: down -v deletes the database volume.

14. Security rules for this VPS

Security is mostly about reducing what is reachable and protecting keys.

Firewall rules

```
Port | Purpose | Source
22 | SSH | Anywhere if your IP is dynamic, but SSH key-only and root login disabled.
80 | HTTP redirect and Let's Encrypt HTTP challenge | Anywhere.
443 | HTTPS | Anywhere.
```

Docker security rules

- Only Traefik publishes public ports.
- Do not publish MySQL, Redis, internal app ports, or dashboards.
- Keep production .env only on the VPS.
- Do not commit secrets to GitHub.
- Treat the deploy user as powerful because Docker access is effectively root-level access.
- Keep SSH private keys safe and rotate them if exposed.

15. Debugging recipes

Certificate warning or self-signed certificate

```
# Check DNS
dig +short kurdibuilds.dev
dig +short www.kurdibuilds.dev

# Check Traefik logs
docker logs traefik --tail=100

# Inspect certificate issuer
echo | openssl s_client -servername kurdibuilds.dev -connect kurdibuilds.dev:443 2>/dev/null | openssl x509 -noout -i
  ssuer -subject -dates
```

If Traefik logs show NXDOMAIN, DNS is missing or not propagated.

If the issuer is not Let's Encrypt, Traefik does not yet have a valid certificate for that domain.

Gateway Timeout

```
docker ps -a
docker logs kurdibuilds_app --tail=100
docker logs traefik --tail=100

# Test whether Traefik can reach the app through the proxy network
docker run --rm --network proxy curlimages/curl:latest -I http://kurdibuilds_app:80

# Check containers on the proxy network
docker network inspect proxy --format '{{range .Containers}}{{.Name}}{{"\n"}}{{end}}'
```

Expected proxy network containers:

```
traefik
kurdibuilds_app
```

If the network curl fails, Traefik cannot reach Laravel. Check networks and traefik.docker.network=proxy.

Database connection refused

```
cd /opt/stacks/apps/kurdibuilds-blog
grep DB_ .env

docker exec kurdibuilds_app printenv DB_HOST

docker exec kurdibuilds_app php artisan config:clear
docker compose -f docker-compose.prod.yml restart app
```

Correct:

```
DB_HOST=db
```

Wrong:

```
DB_HOST=127.0.0.1
```

Vite manifest missing

```
docker exec kurdibuilds_app ls -la public/build/manifest.json
```

If missing, add to Dockerfile:

```
RUN npm ci
RUN npm run build
```

Then rebuild:

```
cd /opt/stacks/apps/kurdibuilds-blog
docker compose -f docker-compose.prod.yml build --no-cache app
docker compose -f docker-compose.prod.yml up -d --force-recreate app
```

Mixed content

```
curl -s https://kurdibuilds.dev | grep -o "http://kurdibuilds.dev[^\"']*"
```

If output appears, Laravel is still generating HTTP asset URLs. Force HTTPS in production and confirm APP_URL uses https.

Apache logs

```
docker exec kurdibuilds_app tail -n 100 /var/log/apache2/error.log
```

The FQDN ServerName warning is not fatal. Laravel errors are usually in Laravel logs.

Laravel logs

```
docker exec kurdibuilds_app tail -n 100 storage/logs/laravel.log
```

16. Daily command reference

Containers

```
docker ps
docker ps -a
docker logs kurdibuilds_app --tail=100
docker logs kurdibuilds_db --tail=100
docker logs traefik --tail=100
```

App container

```
docker exec -it kurdibuilds_app bash
docker exec kurdibuilds_app php -v
docker exec kurdibuilds_app php artisan about
docker exec kurdibuilds_app printenv DB_HOST
```

Networks

```
docker network ls
docker network inspect proxy --format '{{range .Containers}}{{.Name}}{{"\n"}}{{end}}'
```

Rebuild and restart

```
cd /opt/stacks/apps/kurdibuilds-blog

docker compose -f docker-compose.prod.yml build --no-cache app
docker compose -f docker-compose.prod.yml up -d --force-recreate app
```

Deploy manually

```
cd /opt/stacks/apps/kurdibuilds-blog
./deploy.sh
```

17. Golden rules

- Only Traefik exposes ports 80 and 443.
- Never expose MySQL publicly.
- Use DB_HOST=db inside Docker.
- Keep production .env only on the VPS.
- Use Docker volumes for persistent database and storage data.
- Add traefik.docker.network=proxy when an app has multiple networks.
- Build Vite assets during Docker build.
- Run migrations after deploy.
- Keep Cloudflare SSL mode on Full (strict).
- Do not run docker compose down -v unless you want to delete volume data.
- Read logs from the right layer: Traefik for routing, Apache for web server, Laravel for app, MySQL for database.
- Test with a tiny container like whoami before debugging a full Laravel stack.

18. Final mental model

```
Cloudflare = public shield and DNS
Traefik = front door and HTTPS router
Apache = Laravel web server
Laravel = application logic
MySQL = private database
Docker Compose = recipe for running the stack
Docker networks = private roads between containers
Docker volumes = data that survives container rebuilds
GitHub Actions = automation trigger after git push
```

To add a future app:

- Create a new folder under /opt/stacks/apps.
- Give it its own docker-compose.prod.yml.
- Connect its web container to the external proxy network.
- Give it Traefik labels for its domain.
- Keep its database on its own internal network.
- Do not publish database ports.
- Add a deploy script and GitHub Actions workflow when ready.

You now have a practical Docker deployment pattern for one VPS. It is simple enough to understand, but close to real production self-hosting.