Optimizing Laravel for AWS Deployment (Step-by-Step)

Optimizing Laravel for AWS Deployment (Step-by-Step)

AWS gives you flexible building blocks—EC2 for compute, RDS for databases, ElastiCache for Redis, S3 for storage, and CloudFront for global edge caching. In this step-by-step guide, you'll prepare your Laravel app for production, provision AWS services, configure Nginx + PHP-FPM on EC2, wire up queues with Horizon, store assets on S3, and add a health check endpoint for load balancers. We'll keep performance and security front-and-center and link to deep dives where relevant.

1 – Production Prep: Env & Optimizations

Before deploying, make sure your app boots fast and reads secrets from environment variables. These commands pre-compile configuration/routes/views for minimal runtime overhead (see also Article #41 for the rationale).

```
# locally or in your build stage
php artisan key:generate --force
php artisan config:cache
php artisan route:cache
php artisan view:cache
php artisan event:cacheCode language: Bash (bash)
```

These commands write optimized PHP arrays to `bootstrap/cache`, reducing file I/O and framework bootstrap time in production.

.env.production (example) APP_ENV=production APP_DEBUG=false APP_URL=https://your-domain.com LOG_CHANNEL=stack LOG_LEVEL=warning # database via RDS DB_CONNECTION=mysql DB_HOST=your-rds-endpoint.amazonaws.com DB_PORT=3306 DB_DATABASE=app DB_USERNAME=app_user DB_PASSWORD=strong-

password # cache/session via ElastiCache Redis CACHE_DRIVER=redis SESSION_DRIVER=redis REDIS_HOST=your-redis.xxxxxx.use1.cache.amazonaws.com REDIS_PORT=6379 # files & assets FILESYSTEM_DISK=s3 AWS_ACCESS_KEY_ID=… AWS_SECRET_ACCESS_KEY=… AWS_DEFAULT_REGION=us-east-1 AWS_BUCKET=your-bucket AWS_USE_PATH_STYLE_ENDPOINT=false # queues (Redis) + Horizon QUEUE_CONNECTION=redis

Use a separate `.env.production` with real endpoints. In AWS, prefer Systems Manager Parameter Store or Secrets Manager instead of raw env files for credentials.

2 – Provision AWS: RDS, ElastiCache, S3, CloudFront

Create an RDS MySQL (or Postgres) instance, an ElastiCache Redis cluster, and an S3 bucket. Optionally add a CloudFront distribution on top of S3 for global asset delivery. Make sure your EC2 security groups allow egress to these services and RDS/Redis are restricted to your VPC.

```
# example: upload public build assets to S3 during deploy
php artisan storage:link   # for local; in prod we serve from S3 directly
# build your front-end (Vite/Mix) then
aws s3 sync public/build s3://your-bucket/build --deleteCode language: Bash (bash)
```

Syncing compiled assets to S3 lets CloudFront cache them globally. Set long cache-control headers for immutable filenames (Vite’s hashed files).

3 – EC2: PHP-FPM + Nginx Configuration

We'll run Laravel on EC2 behind Nginx with PHP-FPM. This mirrors our Docker/Nginx setup from Article #46 but installed on the instance directly.

```
# on Ubuntu EC2
sudo apt update
sudo apt install -y nginx php8.3-fpm php8.3-xml php8.3-mbstring
php8.3-zip php8.3-mysql php8.3-bcmath php8.3-curl unzip git
sudo systemctl enable php8.3-fpm nginxCode language: Bash (bash)
```

This installs Nginx and PHP 8.3 with common Laravel extensions. Enable services to start on boot so reboots don't cause downtime.

```
# /etc/nginx/sites-available/laravel.conf
server {
    listen 80;
    server_name your-domain.com;
    root /var/www/current/public;

    index index.php index.html;

    location / {
        try_files $uri $uri/ /index.php?$query_string;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/run/php/php8.3-fpm.sock;
        fastcgi_read_timeout 60s;
    }

    location ~* \.(jpg|jpeg|png|gif|css|js|ico|woff2?)$ {
        expires 7d;
        access_log off;
    }
}Code language: Nginx (nginx)
```

This server block routes requests to Laravel's `index.php` and serves static files with caching. Point `root` to your release directory (`/var/www/current`) to support zero-downtime symlink deployments.

```
sudo ln -s /etc/nginx/sites-available/laravel.conf /etc/nginx/sites-enabled/laravel.conf
sudo nginx -t && sudo systemctl reload nginxCode language: Bash (bash)
```

Symlink the site, test the config, and reload Nginx without dropping connections. For more Nginx hardening, see Article #56.

4 – Deploy Script: Zero-Downtime Releases

Use a simple release pattern: upload a timestamped build directory, run composer/artisan, then atomically switch the `current` symlink.

```
# /usr/local/bin/deploy.sh (run as a deploy user)
set -euo pipefail

APP_DIR=/var/www
RELEASES=$APP_DIR/releases
NEW=$RELEASES/$(date +%Y%m%d%H%M%S)

mkdir -p "$NEW"
# rsync or unzip your build into $NEW
rsync -az --delete build/ "$NEW/"

cd "$NEW"
composer install --no-dev --optimize-autoloader
php artisan config:cache
php artisan route:cache
php artisan view:cache
php artisan migrate --force
```

```
ln -sfn "$NEW" "$APP_DIR/current"
sudo systemctl reload php8.3-fpm
sudo systemctl reload nginxCode language: Bash (bash)
```

This script minimizes downtime by swapping symlinks only after the new release is warmed and migrations are applied. For CI/CD automation, see Article #54.

5 - Queues & Horizon on EC2

Use Horizon to manage Redis queues (jobs, notifications, broadcasts). Run it under systemd so it restarts on failure or reboot (see Article #45 for UI and scaling).

```
# /etc/systemd/system/horizon.service
[Unit]
Description=Laravel Horizon
After=network.target

[Service]
User=www-data
Type=simple
WorkingDirectory=/var/www/current
ExecStart=/usr/bin/php artisan horizon
Restart=always
RestartSec=3

[Install]
WantedBy=multi-user.targetCode language: TOML, also INI (ini)
```

Enable and start the service so jobs are continuously processed. Horizon also provides a dashboard at `/horizon` for throughput/failed job metrics.

```
sudo systemctl daemon-reload
sudo systemctl enable --now horizonCode language: Bash (bash)
```

Reload systemd and start Horizon immediately. Pair this with Redis in ElastiCache for a robust, low-latency queue layer (background on queues in Article #42).

6 – Optional: Octane for Concurrency

If your app is CPU-bound and you need higher RPS, consider running **Laravel Octane** (Swoole/RoadRunner) behind Nginx on EC2 (see Article #44 for benchmarks and configuration).

```
# /etc/systemd/system/octane.service
[Unit]
Description=Laravel Octane
After=network.target

[Service]
User=www-data
WorkingDirectory=/var/www/current
ExecStart=/usr/bin/php artisan octane:start --server=swoole --
port=9000 --workers=8 --task-workers=4
Restart=always
RestartSec=3

[Install]
WantedBy=multi-user.targetCode language: TOML, also INI (ini)
```

Expose Octane on an internal port and proxy to it from Nginx. Keep `max_requests` set in `config/octane.php` so workers recycle and memory stays healthy.

7 – S3 Filesystem & CloudFront

Move user uploads to S3 for durability and scale, and serve them via CloudFront for low latency worldwide.

```
// config/filesystems.php (snippet)
'disks' => [
    's3' => [
        'driver' => 's3',
        'key'    => env('AWS_ACCESS_KEY_ID'),
        'secret' => env('AWS_SECRET_ACCESS_KEY'),
        'region' => env('AWS_DEFAULT_REGION', 'us-east-1'),
        'bucket' => env('AWS_BUCKET'),
        'url'    => env('AWS_URL'), // set to CloudFront domain for
public URLs
        'visibility' => 'public',
    ],
],Code language: PHP (php)
```

Setting `AWS_URL` to your CloudFront domain ensures generated URLs (e.g., `Storage::url()`) use the CDN, not the S3 endpoint. Cache objects with long TTLs since filenames are versioned.

8 – Health Check Endpoint (UI)

Add a simple route for ALB/ELB health checks that verifies DB/Redis connectivity without heavy logic.

```
// routes/web.php
Route::get('/health', function () {
    try {
        DB::connection()->getPdo();
```

```
        Cache::put('healthcheck', now(), 5);
        return response()->json(['status' => 'ok', 'time' => now()],
200);
    } catch (\Throwable $e) {
        return response()->json(['status' => 'fail'], 500);
    }
});Code language: PHP (php)
```

Your load balancer can poll `/health` to remove unhealthy instances automatically. Keep it lightweight and fast.

9 – Security Hardening

- Terminate HTTPS at ALB or Nginx using an ACM certificate.
- Keep `APP_DEBUG=false` in production; never expose stack traces.
- Restrict SSH with key pairs and use SSM Session Manager where possible.
- Use Parameter Store/Secrets Manager for DB/API keys; rotate regularly.
- Enable OPcache and set sane limits (covered in Article #41).

Security posture improves resilience and compliance. Pair with WAF and security groups that only expose ports 80/443 to the internet, keeping DB/Redis private.

Wrapping Up

You configured a production-grade Laravel stack on AWS: EC2 with Nginx + PHP-FPM, RDS for MySQL, ElastiCache Redis for cache/sessions/queues, S3 + CloudFront for assets, and a health check endpoint for load balancers. With zero-downtime deploys, Horizon for queues,

and optional Octane for high concurrency, your app is ready to scale reliably and securely.

What’s Next

- CI/CD for Laravel Projects with GitHub Actions — automate your zero-downtime deployment pipeline.
- Laravel & Nginx: Best Practices for Production — tune Nginx timeouts, compression, and buffering for EC2.
- Laravel Deployment Checklist for 2025 — a concise pre-launch audit you can run for every release.