1v0 Ship v1.0 Faster

Deploying Laravel on AWS: Complete Guide (2025)

Deploying Laravel on AWS: Complete Guide (2025)

AWS offers multiple reliable paths to production: EC2 (you manage the box), Elastic Beanstalk (PaaS-like), and ECS Fargate (serverless containers). This guide gives you a modern end-to-end recipe: robust networking and IAM, RDS + ElastiCache + S3/CloudFront, ALB health checks, secure env management, CI/CD with GitHub Actions OIDC, and two deployment tracks (EC2 and ECS Fargate). Where relevant, we'll link deeper dives like #49 AWS Step-by-Step, #54 CI/CD, and #56 Nginx Best Practices.

1 — Choose an Architecture

- **EC2 + Nginx + PHP-FPM**: maximum control, easy to reason about; scale with an ALB + Auto Scaling Group. See the hands-on #49.
- **ECS Fargate**: serverless containers (no servers to manage). Great for blue/green and predictable costs.
- **Elastic Beanstalk**: PaaS convenience, but fewer knobs than ECS/EC2.

Common shared services regardless of path: **RDS** (MySQL/Postgres), **ElastiCache (Redis)** for cache/sessions/queues, **S3 + CloudFront** for assets/uploads, **ALB** for routing and health checks, **Parameter Store/Secrets Manager** for secrets, and **CloudWatch** for logs/metrics.

2 — IAM for GitHub Actions (OIDC)

Use GitHub's OIDC to let Actions assume an AWS role without long-lived keys. Create an IAM role with this trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": { "Federated": "arn:aws:iam::123456789012:oidc-
provider/token.actions.githubusercontent.com" },
    "Action": "sts:AssumeRoleWithWebIdentity",
    "Condition": {
      "StringLike": { "token.actions.githubusercontent.com:sub":
"repo:your-org/your-repo:*" },
      "StringEquals": { "token.actions.githubusercontent.com:aud":
"sts.amazonaws.com" }
    }
  }]
}Code language: JSON / JSON with Comments (json)
```

This limits role assumption to your repository. Attach policies permitting S3 (artifacts), CodeDeploy/ECS (deployments), or SSM/EC2 (remote commands). You'll use this role in the CI/CD workflow below. See also #54 CI/CD.

3 — CI/CD (GitHub Actions AWS CodeDeploy on EC2)

This workflow builds, uploads an artifact to S3, and triggers CodeDeploy to roll it onto your EC2 Auto Scaling Group with zero downtime.

```
# .github/workflows/deploy.yml
name: Deploy to AWS (EC2 via CodeDeploy)
```

```
on:
  push:
    branches: [ main ]
jobs:
  deploy:
    runs-on: ubuntu-latest
    permissions:
      id-token: write
      contents: read
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with: { node-version: 20 }
      - run: npm ci && npm run build
      - uses: php-actions/composer@v6
        with: { php_version: "8.3", args: "--no-dev --optimize-
autoloader" }
      - run: |
          php artisan config:cache
          php artisan route:cache
          php artisan view:cache
      - name: Archive artifact
        run: zip -r deploy.zip . -x ".git/*"
      - name: Configure AWS creds (OIDC)
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume:
arn:aws:iam::123456789012:role/GitHubDeployRole
          aws-region: us-east-1
      - name: Upload to S3
        run: aws s3 cp deploy.zip s3://your-artifacts-
bucket/deploy.zip
      - name: Trigger CodeDeploy
        run: |
          aws deploy create-deployment \
            --application-name laravel-app \
            --deployment-group-name laravel-asg \
            --s3-location bucket=your-artifacts-
```

```
bucket,key=deploy.zip,bundleType=zipCode language: YAML (yaml)
```

Actions uses OIDC to assume your AWS role (no access keys). CodeDeploy handles the rolling update across instances. For a fully scripted EC2 approach without CodeDeploy, see the shell-driven flow in #49.

```
# appspec.yml (at repo root, used by CodeDeploy)
version: 0.0
os: linux
files:
  - source: /
    destination: /var/www/releases/{{deployment-id}}
hooks:
  AfterInstall:
    - location: scripts/after_install.sh
      timeout: 300
  ApplicationStart:
    - location: scripts/start.sh
      timeout: 300Code language: YAML (yaml)
```

`appspec.yml` tells CodeDeploy where to unpack and which hook scripts to run (composer, artisan cache, symlink swap, FPM/Nginx reload). This pattern mirrors the zero-downtime release layout from #49.

4 — EC2 User-Data Bootstrap (One-Time)

Use launch templates with user-data to preinstall Nginx/PHP and create paths expected by CodeDeploy.

```
#!/bin/bash
set -e
apt-get update
apt-get 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
systemctl enable nginx php8.3-fpm
mkdir -p /var/www/releases /var/www/shared /var/www/current
chown -R www-data:www-data /var/www
Code language: Bash (bash)
```

This ensures new instances in the Auto Scaling Group are ready for CodeDeploy to drop releases into `/var/www/releases` and flip the `current` symlink. For Nginx hardening, see #56.

```
# /etc/nginx/sites-available/laravel.conf
server {
  listen 80;
  server_name _;
  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 ~* \.(css|js|jpg|jpeg|png|gif|webp|ico|woff2?)$ {
    expires 7d; access_log off;
  }
}Code language: Nginx (nginx)
```

Point the site root at the `current` symlink so deploys are atomic. Test and reload Nginx in your hook scripts.

5 — RDS, ElastiCache, S3/CloudFront

Plug AWS services into Laravel via `.env` and `config/`. Keep secrets in Parameter Store/Secrets Manager, injected at deploy.

.env.production (snippets) APP_ENV=production APP_DEBUG=false APP_URL=https://your-domain.com DB_CONNECTION=mysql DB_HOST=your-rds.cluster-xxxx.us-east-1.rds.amazonaws.com DB_PORT=3306 DB_DATABASE=app DB_USERNAME=app_user DB_PASSWORD=**** # better: load via SSM param at deploy CACHE_DRIVER=redis SESSION_DRIVER=redis REDIS_HOST=your-redis.xxxxxx.use1.cache.amazonaws.com REDIS_PORT=6379 FILESYSTEM_DISK=s3 AWS_BUCKET=your-bucket AWS_DEFAULT_REGION=us-east-1 AWS_URL=https://dxxxxx.cloudfront.net

RDS handles the relational workload; ElastiCache powers cache/sessions/queues; S3 stores uploads; CloudFront serves them globally. See #43 Caching for why Redis matters under load.

```
// 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'), // CloudFront URL for public reads
    'visibility' => 'public',
  ],
],Code language: PHP (php)
```

Setting `AWS_URL` to your CloudFront domain makes `Storage::url()` emit CDN links automatically (immutable hashed file names = long cache TTLs).

6 — ALB Health Checks

Create a lightweight health route; point the ALB target group to it for safe rolling updates.

```
// routes/web.php
Route::get('/health', function () {
  try {
    DB::connection()->getPdo();
    Cache::put('hc', now(), 5);
    return response()->json(['ok' => true, 't' => now()], 200);
  } catch (\Throwable $e) {
    return response()->json(['ok' => false], 500);
  }
});Code language: PHP (php)
```

The route checks DB and Redis quickly. During deploys, instances failing health are drained by ALB before termination. We used the same pattern in #49.

7 — ECS Fargate Option (Containers, No Servers)

Build a Docker image for your Laravel app (PHP-FPM) and run it behind Nginx in a two-container Task. Fargate handles capacity and patching.

```
# Dockerfile (app)
FROM php:8.3-fpm
RUN apt-get update && apt-get install -y git unzip libpq-dev libzip-dev libpng-dev && \
    docker-php-ext-install pdo_mysql bcmath zip gd
COPY --from=composer:2.7 /usr/bin/composer /usr/bin/composer
WORKDIR /var/www
COPY . .
RUN composer install --no-dev --optimize-autoloader && php artisan
```

```
config:cache && php artisan route:cache && php artisan view:cache
CMD ["php-fpm"]Code language: Dockerfile (dockerfile)
```

This image bakes in optimized caches so containers start fast. Store `.env` values in ECS task secrets sourced from Parameter Store/Secrets Manager, not inside the image.

```
{
  "family": "laravel-app",
  "networkMode": "awsvpc",
  "requiresCompatibilities": ["FARGATE"],
  "cpu": "512",
  "memory": "1024",
  "containerDefinitions": [
    {
      "name": "app",
      "image": "123456789012.dkr.ecr.us-
east-1.amazonaws.com/laravel:latest",
      "portMappings": [{ "containerPort": 9000, "protocol": "tcp" }],
      "secrets": [
        { "name": "DB_HOST", "valueFrom": "arn:aws:ssm:us-
east-1:123456789012:parameter/app/DB_HOST" }
      ],
      "linuxParameters": { "initProcessEnabled": true },
      "essential": true
    },
    {
      "name": "nginx",
      "image": "nginx:alpine",
      "portMappings": [{ "containerPort": 80, "protocol": "tcp" }],
      "mountPoints": [],
      "links": ["app"]
    }
  ]
}Code language: JSON / JSON with Comments (json)
```

Two containers in one task: PHP-FPM on 9000 and Nginx on 80. The Service is fronted by an ALB with the health check path set to `/health`. Blue/green deployments become trivial with ECS.

8 — Queues: Horizon on EC2/ECS

Run Horizon as a separate systemd service on EC2, or as a separate ECS Service using the same image but a different command.

```
# EC2: /etc/systemd/system/horizon.service
[Unit]
Description=Laravel Horizon
After=network.target
[Service]
User=www-data
WorkingDirectory=/var/www/current
ExecStart=/usr/bin/php artisan horizon
Restart=always
RestartSec=3
[Install]
WantedBy=multi-user.targetCode language: TOML, also INI (ini)
```

For ECS, define a second Service running `php artisan horizon` (no port exposed) with desired count >= 1 and CloudWatch Logs enabled. See #45 Horizon and #42 Queues.

9 — Observability & Security

- **Logs**: ship Nginx/PHP/Laravel logs to CloudWatch. Use retention policies.
- **Telescope**: secure `/telescope` with a Gate; use for deep request/query introspection (#48).
- **Secrets**: SSM Parameter Store / Secrets Manager. Avoid committing credentials.

- **WAF**: attach AWS WAF to ALB or CloudFront for L7 protection.
- **OPcache/Octane**: enable OPcache; optionally run Octane for high concurrency (#44).

```
// app/Providers/TelescopeServiceProvider.php (gate snippet)
protected function gate()
{
  Gate::define('viewTelescope', fn ($user) => in_array($user->email,
['admin@example.com']));
}Code language: PHP (php)
```

Locking down Telescope is essential in production; combine app-level auth with security groups and ALB rules. For a final pre-launch audit, use #58 Deployment Checklist.

10 — Minimal Admin Status UI

A tiny Blade page for on-call engineers to sanity-check DB/Redis and horizon queue sizes without shell access.

```
// routes/web.php
Route::middleware(['auth', 'can:viewAdmin'])->get('/admin/status',
function () {
  return view('admin.status', [
    'db' => optional(DB::select('SELECT 1 as ok'))[0]->ok ?? 0,
    'redis' => Cache::put('status_ping', now(), 5) === null ? 1 : 1,
    'queueSize' => Illuminate\Support\Facades\Queue::size(),
  ]);
});Code language: PHP (php)
```

The route is gated and returns a Blade view with simple indicators. Use Horizon for full visibility, but this helps verify env quickly during incidents.

```
<!-- resources/views/admin/status.blade.php -->
@extends('layouts.app')
```

```
@section('content')
<div class="container">
  <h1 class="mb-4">System Status</h1>
  <ul class="list-group">
    <li class="list-group-item">DB: {{ $db ? 'OK' : 'FAIL' }}</li>
    <li class="list-group-item">Redis: {{ $redis ? 'OK' : 'FAIL'
}}</li>
    <li class="list-group-item">Queue Size: {{ $queueSize }}</li>
  </ul>
  <a href="/horizon" class="btn btn-theme mt-3">Open Horizon</a>
</div>
@endsectionCode language: HTML, XML (xml)
```

Keep it minimal, authenticated, and non-sensitive. For real-time job internals, Horizon remains the primary dashboard.

Wrapping Up

You now have two solid AWS deployment paths for Laravel in 2025: EC2 with CodeDeploy (fine-grained control) and ECS Fargate (serverless containers). You wired RDS, ElastiCache, S3/CloudFront, ALB health checks, and secure env management via OIDC + Parameter Store. Add Horizon for queues, Telescope for diagnostics, OPcache/Octane for speed, and CloudWatch/WAF for resilience. Pick the path that matches your ops maturity and scaling goals.

1v0 Ship v1.0 Faster

What’s Next

- Optimizing Laravel for AWS Deployment (Step-by-Step) — deeper, box-level configuration on EC2.
- CI/CD for Laravel Projects with GitHub Actions — pipelines, caching, and safe rollouts.
- Laravel & Nginx: Best Practices for Production — timeouts, compression, and buffering.
- Laravel Deployment Checklist for 2025 — run this before each release.
- Optimizing Laravel for High Concurrency with Octane — when you need serious RPS gains.