Guia de Deploy - DMCA Guard Platform

Este guia fornece instruções detalhadas para deploy da plataforma DMCA Guard em diferentes provedores de cloud.

Railway (Recomendado)

Railway oferece a melhor experiência para deploy de aplicações Next.js com banco PostgreSQL integrado.

1. Preparação

Pré-requisitos

- Conta no Railway (https://railway.app)
- Código no GitHub/GitLab
- Variáveis de ambiente configuradas

Estrutura de Arquivos

```
# Verificar se existe railway.json
cat > railway.json << EOF
{
  "build": {
    "builder": "NIXPACKS"
  },
  "deploy": {
    "startCommand": "yarn start",
    "healthcheckPath": "/api/health"
  }
}
EOF
```

2. Deploy Automático

Via Dashboard Railway

1. **Acesse** railway.app (https://railway.app) e faça login
2. **Clique** em “New Project”
3. **Selecione** “Deploy from GitHub repo”
4. **Escolha** seu repositório
5. **Configure** as variáveis de ambiente

Via CLI Railway

```
# Instalar Railway CLI
npm install -g @railway/cli

# Login
railway login

# Inicializar projeto
railway init

# Adicionar PostgreSQL
railway add postgresql

# Deploy
railway up
```

3. Configuração de Variáveis

```
# Configurar via CLI
railway variables set NEXTAUTH_SECRET="your-secret-here"
railway variables set OPENAI_API_KEY="sk-your-key"
railway variables set SENDGRID_API_KEY="SG.your-key"

# Ou via arquivo .env
railway variables set --file .env.production
```

Variáveis Específicas do Railway

```
# .env.production
# Railway fornece automaticamente
DATABASE_URL=${{Postgres.DATABASE_URL}}
RAILWAY_STATIC_URL=${{RAILWAY_STATIC_URL}}
RAILWAY_PUBLIC_DOMAIN=${{RAILWAY_PUBLIC_DOMAIN}}

# Suas variáveis
NEXTAUTH_URL=${{RAILWAY_PUBLIC_DOMAIN}}
APP_URL=${{RAILWAY_PUBLIC_DOMAIN}}
```

4. Configuração de Banco

```
# Conectar ao banco Railway
railway connect postgresql

# Executar migrações
railway run npx prisma migrate deploy

# Seed inicial (opcional)
railway run npx prisma db seed
```

5. Domínio Customizado

Via Dashboard

1. **Acesse** seu projeto no Railway

2. **Vá** para “Settings” > “Domains”
3. **Adicione** seu domínio customizado
4. **Configure** DNS conforme instruções

Configuração DNS

```
# Adicionar CNAME no seu provedor DNS
CNAME www your-app.railway.app
CNAME @ your-app.railway.app
```

☁ Vercel

Vercel é ideal para frontend com serverless functions, mas requer banco externo.

1. Preparação

Configuração do Projeto

```
// vercel.json
{
  "framework": "nextjs",
  "buildCommand": "yarn build",
  "devCommand": "yarn dev",
  "installCommand": "yarn install",
  "functions": {
    "app/api/**/*.js": {
      "maxDuration": 30
    }
  },
  "env": {
    "NEXTAUTH_URL": "https://your-domain.vercel.app"
  }
}
```

2. Deploy via CLI

```
# Instalar Vercel CLI
npm install -g vercel

# Login
vercel login

# Deploy
vercel

# Deploy para produção
vercel --prod
```

3. Configuração de Banco

Opção 1: Supabase (Recomendado)

```
# 1. Criar projeto no Supabase
# 2. Obter connection string
# 3. Configurar no Vercel

vercel env add DATABASE_URL
# Cole a URL do Supabase
```

Opção 2: PlanetScale

```
# 1. Criar banco no PlanetScale
# 2. Configurar connection string
# 3. Executar migrações

npx prisma db push
```

4. Variáveis de Ambiente

```
# Configurar via CLI
vercel env add NEXTAUTH_SECRET
vercel env add OPENAI_API_KEY
vercel env add SENDGRID_API_KEY

# Ou via dashboard
# https://vercel.com/your-team/your-project/settings/environment-variables
```

5. Configuração de Domínio

```
# Adicionar domínio customizado
vercel domains add yourdomain.com

# Configurar DNS
# A record: @ -> 76.76.19.61
# CNAME: www -> cname.vercel-dns.com
```

AWS (Produção Enterprise)

Deploy completo na AWS com alta disponibilidade e escalabilidade.

1. Arquitetura AWS

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   CloudFront    │    │   Application   │    │   Database      │
│   (CDN)         │───▶│   Load Balancer │───▶│   RDS           │
└─────────────────┘    └─────────────────┘    └─────────────────┘
                                │
                       ┌─────────────────┐
                       │   ECS/Fargate   │
                       │   (Containers)  │
                       └─────────────────┘
```

2. Infraestrutura como Código

Terraform Configuration

```
# infrastructure/main.tf
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region = var.aws_region
}

# VPC
resource "aws_vpc" "dmca_vpc" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "dmca-guard-vpc"
  }
}

# Subnets
resource "aws_subnet" "public" {
  count             = 2
  vpc_id            = aws_vpc.dmca_vpc.id
  cidr_block        = "10.0.${count.index + 1}.0/24"
  availability_zone = data.aws_availability_zones.available.names[count.index]

  map_public_ip_on_launch = true

  tags = {
    Name = "dmca-guard-public-${count.index + 1}"
  }
}

resource "aws_subnet" "private" {
  count             = 2
  vpc_id            = aws_vpc.dmca_vpc.id
  cidr_block        = "10.0.${count.index + 10}.0/24"
  availability_zone = data.aws_availability_zones.available.names[count.index]

  tags = {
    Name = "dmca-guard-private-${count.index + 1}"
  }
}

# RDS Database
resource "aws_db_instance" "dmca_db" {
  identifier = "dmca-guard-db"

  engine         = "postgres"
  engine_version = "14.9"
  instance_class = "db.t3.micro"
  allocated_storage     = 20
  max_allocated_storage = 100
  storage_type          = "gp2"
  storage_encrypted     = true

  db_name  = "dmca_guard"
  username = var.db_username
  password = var.db_password

  vpc_security_group_ids = [aws_security_group.rds.id]
  db_subnet_group_name   = aws_db_subnet_group.dmca.name

  backup_retention_period = 7
  backup_window          = "03:00-04:00"
  maintenance_window     = "sun:04:00-sun:05:00"

  skip_final_snapshot = false
  final_snapshot_identifier = "dmca-guard-final-snapshot"

  tags = {
    Name = "dmca-guard-database"
  }
}

# ECS Cluster
resource "aws_ecs_cluster" "dmca_cluster" {
  name = "dmca-guard-cluster"

  setting {
    name  = "containerInsights"
    value = "enabled"
  }
}

# ECS Task Definition
resource "aws_ecs_task_definition" "dmca_app" {
  family                   = "dmca-guard-app"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = 512
  memory                   = 1024
  execution_role_arn       = aws_iam_role.ecs_execution_role.arn
  task_role_arn           = aws_iam_role.ecs_task_role.arn

  container_definitions = jsonencode([
    {
      name  = "dmca-guard-app"
      image = "${aws_ecr_repository.dmca_app.repository_url}:latest"

      portMappings = [
        {
          containerPort = 3000
          protocol      = "tcp"
        }
      ]

      environment = [
        {
          name  = "NODE_ENV"
          value = "production"
        },
        {
          name  = "DATABASE_URL"
          value = "postgresql://${var.db_username}:${var.db_password}@$
{aws_db_instance.dmca_db.endpoint}/${aws_db_instance.dmca_db.db_name}"
        }
      ]

      secrets = [
        {
          name      = "NEXTAUTH_SECRET"
          valueFrom = aws_ssm_parameter.nextauth_secret.arn
        },
        {
          name      = "OPENAI_API_KEY"
          valueFrom = aws_ssm_parameter.openai_key.arn
        },
        {
          name      = "SENDGRID_API_KEY"
          valueFrom = aws_ssm_parameter.sendgrid_key.arn
        }
      ]

      logConfiguration = {
        logDriver = "awslogs"
        options = {
          awslogs-group         = aws_cloudwatch_log_group.dmca_app.name
          awslogs-region        = var.aws_region
          awslogs-stream-prefix = "ecs"
        }
      }
    }
  ])
}

# Application Load Balancer
resource "aws_lb" "dmca_alb" {
  name               = "dmca-guard-alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb.id]
  subnets            = aws_subnet.public[*].id

  enable_deletion_protection = false

  tags = {
    Name = "dmca-guard-alb"
  }
}
```

3. Deploy com Docker

Dockerfile Otimizado

```dockerfile
# Dockerfile.production
FROM node:18-alpine AS base

# Install dependencies only when needed
FROM base AS deps
RUN apk add --no-cache libc6-compat
WORKDIR /app

COPY package.json yarn.lock* package-lock.json* pnpm-lock.yaml* ./
RUN \
  if [ -f yarn.lock ]; then yarn --frozen-lockfile; \
  elif [ -f package-lock.json ]; then npm ci; \
  elif [ -f pnpm-lock.yaml ]; then yarn global add pnpm && pnpm i --frozen-lockfile; \
  else echo "Lockfile not found." && exit 1; \
  fi

# Rebuild the source code only when needed
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .

# Generate Prisma client
RUN npx prisma generate

# Build application
RUN yarn build

# Production image, copy all the files and run next
FROM base AS runner
WORKDIR /app

ENV NODE_ENV production

RUN addgroup --system --gid 1001 nodejs
RUN adduser --system --uid 1001 nextjs

COPY --from=builder /app/public ./public

# Set the correct permission for prerender cache
RUN mkdir .next
RUN chown nextjs:nodejs .next

# Automatically leverage output traces to reduce image size
COPY --from=builder --chown=nextjs:nodejs /app/.next/standalone ./
COPY --from=builder --chown=nextjs:nodejs /app/.next/static ./.next/static

USER nextjs

EXPOSE 3000

ENV PORT 3000
ENV HOSTNAME "0.0.0.0"

CMD ["node", "server.js"]
```

Build e Push para ECR

```bash
#!/bin/bash
# scripts/deploy_aws.sh

set -e

# Configurações
AWS_REGION="us-east-1"
ECR_REPOSITORY="dmca-guard-app"
IMAGE_TAG="latest"

echo "  Iniciando deploy na AWS..."

# 1. Build da imagem Docker
echo "  Building Docker image..."
docker build -f Dockerfile.production -t $ECR_REPOSITORY:$IMAGE_TAG .

# 2. Login no ECR
echo "  Logging into ECR..."
aws ecr get-login-password --region $AWS_REGION | docker login --username AWS --pass-
word-stdin $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com

# 3. Tag da imagem
echo "  Tagging image..."
docker tag $ECR_REPOSITORY:$IMAGE_TAG $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazon-
aws.com/$ECR_REPOSITORY:$IMAGE_TAG

# 4. Push para ECR
echo "⬆ Pushing to ECR..."
docker push $AWS_ACCOUNT_ID.dkr.ecr.$AWS_REGION.amazonaws.com/$ECR_REPOSITORY:$IM-
AGE_TAG

# 5. Deploy via Terraform
echo "  Deploying infrastructure..."
cd infrastructure
terraform init
terraform plan
terraform apply -auto-approve

# 6. Executar migrações
echo "  Running database migrations..."
aws ecs run-task \
  --cluster dmca-guard-cluster \
  --task-definition dmca-guard-migrations \
  --launch-type FARGATE \
  --network-configuration "awsvpcConfiguration={subnets=[subnet-
xxx],securityGroups=[sg-xxx],assignPublicIp=ENABLED}"

echo "  Deploy completed successfully!"
```

4. Configuração de SSL/TLS

Certificate Manager

```
# SSL Certificate
resource "aws_acm_certificate" "dmca_cert" {
  domain_name       = var.domain_name
  validation_method = "DNS"

  subject_alternative_names = [
    "*.${var.domain_name}"
  ]

  lifecycle {
    create_before_destroy = true
  }
}

# Certificate validation
resource "aws_acm_certificate_validation" "dmca_cert" {
  certificate_arn         = aws_acm_certificate.dmca_cert.arn
  validation_record_fqdns = [for record in aws_route53_record.cert_validation : re-
cord.fqdn]
}
```

5. Monitoramento e Logs

CloudWatch Configuration

```
# CloudWatch Log Group
resource "aws_cloudwatch_log_group" "dmca_app" {
  name              = "/ecs/dmca-guard-app"
  retention_in_days = 30
}

# CloudWatch Alarms
resource "aws_cloudwatch_metric_alarm" "high_cpu" {
  alarm_name          = "dmca-guard-high-cpu"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = "2"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/ECS"
  period              = "120"
  statistic           = "Average"
  threshold           = "80"
  alarm_description   = "This metric monitors ecs cpu utilization"
  alarm_actions       = [aws_sns_topic.alerts.arn]

  dimensions = {
    ServiceName = aws_ecs_service.dmca_app.name
    ClusterName = aws_ecs_cluster.dmca_cluster.name
  }
}
```

Configurações Gerais

1. Health Checks

```
// app/api/health/route.js
import { NextResponse } from 'next/server';
import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

export async function GET() {
  try {
    // Check database connection
    await prisma.$queryRaw`SELECT 1`;

    // Check external APIs
    const openaiStatus = await checkOpenAI();
    const sendgridStatus = await checkSendGrid();

    const health = {
      status: 'healthy',
      timestamp: new Date().toISOString(),
      services: {
        database: 'healthy',
        openai: openaiStatus,
        sendgrid: sendgridStatus,
      },
      version: process.env.npm_package_version,
      uptime: process.uptime(),
    };

    return NextResponse.json(health);
  } catch (error) {
    return NextResponse.json(
      {
        status: 'unhealthy',
        error: error.message,
        timestamp: new Date().toISOString(),
      },
      { status: 503 }
    );
  }
}

async function checkOpenAI() {
  try {
    const response = await fetch('https://api.openai.com/v1/models', {
      headers: {
        'Authorization': `Bearer ${process.env.OPENAI_API_KEY}`,
      },
    });
    return response.ok ? 'healthy' : 'unhealthy';
  } catch {
    return 'unhealthy';
  }
}

async function checkSendGrid() {
  try {
    const response = await fetch('https://api.sendgrid.com/v3/user/profile', {
      headers: {
        'Authorization': `Bearer ${process.env.SENDGRID_API_KEY}`,
      },
    });
    return response.ok ? 'healthy' : 'unhealthy';
  } catch {
    return 'unhealthy';
  }
}
```

2. Configuração de CORS

```
// middleware.js
import { NextResponse } from 'next/server';

export function middleware(request) {
  // Handle CORS
  if (request.method === 'OPTIONS') {
    return new NextResponse(null, {
      status: 200,
      headers: {
        'Access-Control-Allow-Origin': process.env.ALLOWED_ORIGINS || '*',
        'Access-Control-Allow-Methods': 'GET, POST, PUT, DELETE, OPTIONS',
        'Access-Control-Allow-Headers': 'Content-Type, Authorization',
      },
    });
  }

  // Security headers
  const response = NextResponse.next();

  response.headers.set('X-Frame-Options', 'DENY');
  response.headers.set('X-Content-Type-Options', 'nosniff');
  response.headers.set('Referrer-Policy', 'strict-origin-when-cross-origin');
  response.headers.set('Permissions-Policy', 'camera=(), microphone=(), geoloca-
tion=()');

  return response;
}

export const config = {
  matcher: [
    '/((?!_next/static|_next/image|favicon.ico).*)',
  ],
};
```

3. Backup e Disaster Recovery

Script de Backup

```bash
#!/bin/bash
# scripts/backup.sh

set -e

BACKUP_DIR="/backups"
DATE=$(date +%Y%m%d_%H%M%S)
DB_NAME="dmca_guard"

echo "  Starting backup process..."

# Database backup
if [ "$ENVIRONMENT" = "railway" ]; then
    railway run pg_dump $DATABASE_URL > "$BACKUP_DIR/db_backup_$DATE.sql"
elif [ "$ENVIRONMENT" = "aws" ]; then
    aws rds create-db-snapshot \
        --db-instance-identifier dmca-guard-db \
        --db-snapshot-identifier dmca-guard-snapshot-$DATE
fi

# Upload files backup
if [ -d "uploads" ]; then
    tar -czf "$BACKUP_DIR/uploads_backup_$DATE.tar.gz" uploads/
fi

# Upload to S3 (if AWS)
if [ "$ENVIRONMENT" = "aws" ]; then
    aws s3 cp "$BACKUP_DIR/" s3://dmca-guard-backups/ --recursive
fi

echo "  Backup completed: $DATE"
```

4. Monitoramento de Performance

New Relic Integration

```javascript
// lib/monitoring/newrelic.js
import newrelic from 'newrelic';

export function trackCustomEvent(eventType, attributes) {
  if (process.env.NODE_ENV === 'production') {
    newrelic.recordCustomEvent(eventType, attributes);
  }
}

export function trackDMCARequest(data) {
  trackCustomEvent('DMCARequest', {
    platform: data.platform,
    success: data.success,
    responseTime: data.responseTime,
  });
}

export function trackUserAction(action, userId) {
  trackCustomEvent('UserAction', {
    action,
    userId,
    timestamp: Date.now(),
  });
}
```

Comparação de Provedores

Recurso	Railway	Vercel	AWS
Facilidade			
Custo	$5-20/mês	$20-100/mês	$50-500/mês
Escalabilidade			
Banco Integrado			
Auto-scaling			
Monitoramento			
Suporte			

Recomendações por Cenário

- **MVP/Startup**: Railway (simplicidade + custo)
- **Crescimento Rápido**: Vercel (performance + DX)
- **Enterprise**: AWS (controle + compliance)

Troubleshooting de Deploy

Problemas Comuns

Build Failures

```
# Verificar logs de build
railway logs --deployment

# Limpar cache
railway run yarn cache clean
railway run rm -rf .next node_modules
railway run yarn install
```

Database Connection Issues

```
# Verificar variáveis de ambiente
railway variables

# Testar conexão
railway run npx prisma db pull
```

Memory Issues

```
// next.config.js
/** @type {import('next').NextConfig} */
const nextConfig = {
  experimental: {
    outputFileTracingIncludes: {
      '/api/**/*': ['./node_modules/**/*.wasm'],
    },
  },
  // Otimizações de memória
  webpack: (config, { isServer }) => {
    if (isServer) {
      config.optimization.splitChunks = false;
    }
    return config;
  },
};

module.exports = nextConfig;
```

Próximos Passos: Após o deploy, consulte o Manual do Usuário (user_guide.md) para configurar sua primeira marca e iniciar o monitoramento.