🚀 Guia de Deploy - Timming LoveU

Este guia completo apresenta todas as opções de deploy para a aplicação Timming LoveU em ambiente de produção.

📋 Índice

🎯 Pré-requisitos

Software Necessário

- Node.js 18.x ou superior
- PostgreSQL 15.x ou superior
- Git
- Docker & Docker Compose (opcional, para deploy via containers)
- npm ou yarn

Contas Necessárias

- [ ] Conta na Vercel (para deploy na Vercel)
- [ ] Banco de dados PostgreSQL configurado
- [ ] Domínio próprio (opcional)

✅ Checklist Pré-Deploy

Execute os seguintes passos antes do deploy:

1. Verificações de Código

```
# Verificar tipos TypeScript
npm run type-check

# Executar linter
npm run lint

# Executar testes
npm test

# Verificar cobertura de testes
npm run test:coverage
```

2. Verificações de Segurança

- [ ] Variável `NEXTAUTH_SECRET` configurada com valor forte
- [ ] Credenciais de banco de dados seguras
- [ ] Headers de segurança configurados no `next.config.js`
- [ ] HTTPS habilitado em produção
- [ ] Rate limiting configurado (se aplicável)

3. Otimizações

- [ ] Imagens otimizadas
- [ ] Build de produção testado localmente
- [ ] Cache configurado adequadamente
- [ ] Prisma Client gerado

4. Banco de Dados

- [ ] Backup do banco de dados realizado
- [ ] Migrations testadas
- [ ] Índices criados para queries frequentes
- [ ] Connection pooling configurado

🌐 Opções de Deploy

1️⃣ Deploy na Vercel (Recomendado)

A Vercel oferece a melhor experiência para aplicações Next.js, com deploy contínuo e escalabilidade automática.

Passo a Passo

1. Preparar o Repositório Git

```
# Commit todas as mudanças
git add .
git commit -m "Preparar para deploy em produção"
git push origin main
```

2. Configurar Projeto na Vercel
1. Acesse vercel.com (https://vercel.com) e faça login
2. Clique em “Add New Project”
3. Importe seu repositório do GitHub/GitLab/Bitbucket
4. Configure as variáveis de ambiente (veja seção abaixo)

3. Configurar Variáveis de Ambiente na Vercel

No dashboard da Vercel, adicione as seguintes variáveis:

```
# Database
DATABASE_URL=postgresql://user:password@host:5432/database

# NextAuth
NEXTAUTH_SECRET=seu-secret-aqui
NEXTAUTH_URL=https://seu-dominio.vercel.app

# (Opcional) Outras configurações
NODE_ENV=production
```

4. Configurar Build Settings
- **Framework Preset**: Next.js
- **Build Command**: `prisma generate && next build`
- **Output Directory**: `.next` (padrão)
- **Install Command**: `npm install`

5. Deploy

```
# Deploy via CLI (opcional)
npm install -g vercel
vercel login
vercel --prod
```

6. Executar Migrations

```
# Conecte-se ao banco de produção e execute
npx prisma migrate deploy
```

Configuração de Domínio Personalizado

1. No dashboard da Vercel, vá em “Settings” > “Domains”
2. Adicione seu domínio personalizado
3. Configure os registros DNS conforme instruído
4. Atualize `NEXTAUTH_URL` para usar o novo domínio

2️⃣ Deploy com Docker

Deploy usando containers Docker oferece portabilidade e facilita o gerenciamento de dependências.

Passo a Passo

1. Preparar Variáveis de Ambiente

Crie um arquivo `.env.production`:

```
# Database
DATABASE_URL=postgresql://timming:senha@db:5432/timming_loveu
POSTGRES_USER=timming
POSTGRES_PASSWORD=senha_segura_aqui
POSTGRES_DB=timming_loveu

# NextAuth
NEXTAUTH_SECRET=gere-com-openssl-rand-base64-32
NEXTAUTH_URL=https://seu-dominio.com

# Application
NODE_ENV=production
PORT=3000
```

2. Build da Imagem Docker

```
# Build da imagem
npm run docker:build

# Ou manualmente
docker build -t timming-loveu:latest .
```

3. Iniciar Aplicação com Docker Compose

```
# Iniciar todos os serviços
docker-compose up -d

# Ver logs
docker-compose logs -f app

# Verificar status
docker-compose ps
```

4. Executar Migrations

```
# Executar migrations no container
docker-compose exec app npm run prisma:migrate
```

5. Health Check

```
# Verificar se a aplicação está saudável
curl http://localhost:3000/api/health
```

Docker Compose - Serviços Incluídos

- **app**: Aplicação Next.js
- **db**: PostgreSQL database
- **redis**: Cache e rate limiting (opcional)
- **nginx**: Reverse proxy (opcional, use profile `with-nginx`)

Para incluir Nginx:

```
docker-compose --profile with-nginx up -d
```

Comandos Úteis Docker

```
# Parar serviços
docker-compose down

# Reiniciar serviços
docker-compose restart

# Ver logs de um serviço específico
docker-compose logs -f app

# Acessar shell do container
docker-compose exec app sh

# Remover tudo (incluindo volumes)
docker-compose down -v
```

3️⃣ Deploy Manual (VPS/Cloud)

Para deploy em servidores VPS (DigitalOcean, AWS EC2, Google Cloud, etc.)

Passo a Passo

1. Preparar Servidor

```
# Atualizar sistema
sudo apt update && sudo apt upgrade -y

# Instalar Node.js 18
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt install -y nodejs

# Instalar PostgreSQL
sudo apt install -y postgresql postgresql-contrib

# Instalar PM2 (gerenciador de processos)
sudo npm install -g pm2
```

2. Configurar PostgreSQL

```
# Acessar PostgreSQL
sudo -u postgres psql

# Criar database e usuário
CREATE DATABASE timming_loveu;
CREATE USER timming WITH PASSWORD 'senha_segura';
GRANT ALL PRIVILEGES ON DATABASE timming_loveu TO timming;
\q
```

3. Clonar Repositório

```
# Criar diretório da aplicação
cd /var/www
sudo git clone https://github.com/seu-usuario/timming-loveu.git
cd timming-loveu

# Configurar permissões
sudo chown -R $USER:$USER /var/www/timming-loveu
```

4. Configurar Variáveis de Ambiente

```
# Criar arquivo .env
nano .env

# Adicionar variáveis (copie do .env.example)
DATABASE_URL=postgresql://timming:senha@localhost:5432/timming_loveu
NEXTAUTH_SECRET=$(openssl rand -base64 32)
NEXTAUTH_URL=https://seu-dominio.com
NODE_ENV=production
```

5. Instalar Dependências e Build

```
# Instalar dependências
npm ci --only=production

# Gerar Prisma Client
npx prisma generate

# Executar migrations
npx prisma migrate deploy

# Build da aplicação
npm run build
```

6. Configurar PM2

```
# Criar arquivo ecosystem.config.js
cat > ecosystem.config.js << EOF
module.exports = {
  apps: [{
    name: 'timming-loveu',
    script: 'npm',
    args: 'start',
    instances: 'max',
    exec_mode: 'cluster',
    env: {
      NODE_ENV: 'production',
      PORT: 3000
    }
  }]
}
EOF

# Iniciar aplicação com PM2
pm2 start ecosystem.config.js

# Configurar PM2 para iniciar no boot
pm2 startup
pm2 save
```

7. Configurar Nginx (Reverse Proxy)

```
# Instalar Nginx
sudo apt install -y nginx

# Criar configuração
sudo nano /etc/nginx/sites-available/timming-loveu

# Adicionar configuração:
server {
    listen 80;
    server_name seu-dominio.com;

    location / {
        proxy_pass http://localhost:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

# Ativar site
sudo ln -s /etc/nginx/sites-available/timming-loveu /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

8. Configurar SSL com Let’s Encrypt

```
# Instalar Certbot
sudo apt install -y certbot python3-certbot-nginx

# Obter certificado SSL
sudo certbot --nginx -d seu-dominio.com

# Renovação automática já está configurada
```

9. Configurar Firewall

```
# Permitir apenas portas necessárias
sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
```

10. Verificar Deploy

```
# Verificar status PM2
pm2 status

# Ver logs
pm2 logs timming-loveu

# Verificar health check
curl https://seu-dominio.com/api/health
```

🗄️ Configuração de Banco de Dados

PostgreSQL em Produção

Opções de Hospedagem

1. **Vercel Postgres** (integrado com Vercel)
2. **Supabase** (gratuito com limites)
3. **Railway** (fácil setup)
4. **Amazon RDS**
5. **Google Cloud SQL**
6. **DigitalOcean Managed Databases**

Connection String Format

```
postgresql://USER:PASSWORD@HOST:PORT/DATABASE?schema=public
```

Connection Pooling (Recomendado)

Para ambientes serverless (Vercel), use connection pooling:

```
datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
  directUrl = env("DIRECT_DATABASE_URL") // Para migrations
}
```

Configurar variáveis:

```
# Com Supabase Pooler
DATABASE_URL="postgresql://user:pass@pooler.supabase.com:6543/postgres?pgbouncer=true"
DIRECT_DATABASE_URL="postgresql://user:pass@db.supabase.com:5432/postgres"
```

Backups Automáticos

Script de backup (adicione ao crontab):

```
#!/bin/bash
# backup-db.sh

DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_DIR="/var/backups/timming-loveu"
DB_NAME="timming_loveu"

mkdir -p $BACKUP_DIR

pg_dump $DB_NAME | gzip > $BACKUP_DIR/backup_$DATE.sql.gz

# Manter apenas últimos 7 dias
find $BACKUP_DIR -name "backup_*.sql.gz" -mtime +7 -delete
```

Adicionar ao crontab:

```
# Backup diário às 2h da manhã
0 2 * * * /path/to/backup-db.sh
```

🔐 Variáveis de Ambiente

Variáveis Obrigatórias

Variável	Descrição	Exemplo
`DATABASE_URL`	Connection string PostgreSQL	`postgresql://user:pass@host:5432/db`
`NEXTAUTH_SECRET`	Secret para JWT do NextAuth	`openssl rand -base64 32`
`NEXTAUTH_URL`	URL da aplicação	`https://timming-loveu.com`
`NODE_ENV`	Ambiente de execução	`production`

Variáveis Opcionais

Variável	Descrição	Padrão
`PORT`	Porta da aplicação	`3000`
`DATABASE_URL_DIRECT`	URL direta do DB (para migrations)	-
`REDIS_URL`	URL do Redis (cache)	-
`SENTRY_DSN`	Sentry para error tracking	-
`SMTP_*`	Configurações de email	-

Gerando NEXTAUTH_SECRET

```
# Linux/Mac
openssl rand -base64 32

# Ou online
node -e "console.log(require('crypto').randomBytes(32).toString('base64'))"
```

📊 Monitoramento e Manutenção

Health Checks

A aplicação inclui um endpoint de health check:

```
# Verificar status
curl https://seu-dominio.com/api/health

# Resposta esperada
{
  "status": "healthy",
  "timestamp": "2024-10-23T10:00:00.000Z",
  "uptime": 12345,
  "responseTime": "45ms",
  "checks": {
    "database": "healthy"
  },
  "version": "1.0.0",
  "environment": "production"
}
```

Logs

Com PM2:

```
# Ver logs em tempo real
pm2 logs timming-loveu

# Logs com filtro
pm2 logs timming-loveu --lines 100

# Limpar logs
pm2 flush
```

Com Docker:

```
# Ver logs
docker-compose logs -f app

# Últimas 100 linhas
docker-compose logs --tail=100 app
```

Monitoramento de Performance

Ferramentas Recomendadas:
- **Vercel Analytics** (se usando Vercel)
- **Sentry** (error tracking)
- **New Relic** (APM)
- **Datadog** (infraestrutura)
- **UptimeRobot** (uptime monitoring)

Atualizações

Processo de Atualização:

1. **Backup do banco de dados**

```
pg_dump timming_loveu > backup_antes_update.sql
```

1. **Atualizar código**

```
git pull origin main
npm ci
npm run build
```

1. **Executar migrations**

```
npx prisma migrate deploy
```

1. **Reiniciar aplicação**

```
# PM2
pm2 restart timming-loveu

# Docker
docker-compose restart app

# Vercel
# Push no Git dispara deploy automático
```

🔧 Troubleshooting

Problemas Comuns

1. Erro de Conexão com Banco de Dados

Sintoma: `Can't reach database server`

Soluções:

```
# Verificar se PostgreSQL está rodando
sudo systemctl status postgresql

# Testar conexão
psql $DATABASE_URL

# Verificar firewall
sudo ufw status
```

2. Erro 502 Bad Gateway

Sintoma: Nginx retorna 502

Soluções:

```
# Verificar se a aplicação está rodando
pm2 status

# Reiniciar aplicação
pm2 restart timming-loveu

# Verificar logs
pm2 logs
```

3. Build Falha

Sintoma: `Error: Build failed`

Soluções:

```
# Limpar cache
rm -rf .next node_modules
npm install
npm run build

# Verificar erros TypeScript
npm run type-check
```

4. Prisma Client Error

Sintoma: `PrismaClient is unable to be run in the browser`

Solução:

```
# Regenerar Prisma Client
npx prisma generate

# Limpar e rebuild
rm -rf node_modules/.prisma
npm run build
```

5. Memória Insuficiente

Sintoma: `JavaScript heap out of memory`

Soluções:

```
# Aumentar limite de memória Node.js
export NODE_OPTIONS="--max-old-space-size=4096"
npm run build

# Ou adicionar no package.json:
"build": "NODE_OPTIONS='--max-old-space-size=4096' next build"
```

Scripts de Debug

Verificar status geral:

```
#!/bin/bash
# debug-status.sh

echo "=== System Status ==="
echo "Disk Space:"
df -h

echo "\n=== Memory Usage ==="
free -h

echo "\n=== PM2 Status ==="
pm2 status

echo "\n=== Database Connection ==="
psql $DATABASE_URL -c "SELECT version();"

echo "\n=== Health Check ==="
curl http://localhost:3000/api/health
```

📝 Checklist Final Pós-Deploy

- [ ] Aplicação acessível via HTTPS
- [ ] Health check retorna status 200
- [ ] Banco de dados conectado e funcionando
- [ ] Autenticação funcionando (login/logout)
- [ ] Upload de mídia funcionando
- [ ] Testes de carga realizados
- [ ] Backups configurados
- [ ] Monitoring configurado
- [ ] SSL/TLS válido
- [ ] DNS configurado corretamente
- [ ] Logs sendo coletados
- [ ] Performance otimizada

🆘 Suporte

Em caso de problemas:

1. Verificar logs da aplicação
2. Verificar endpoint `/api/health`
3. Consultar este guia de troubleshooting
4. Revisar documentações:
   - Next.js Deployment (https://nextjs.org/docs/deployment)
   - Prisma Deployment (https://www.prisma.io/docs/guides/deployment)
   - Vercel Documentation (https://vercel.com/docs)

📚 Recursos Adicionais

- Next.js Production Checklist (https://nextjs.org/docs/going-to-production)
- Prisma Best Practices (https://www.prisma.io/docs/guides/performance-and-optimization)
- Security Best Practices (https://cheatsheetseries.owasp.org/)
- Docker Best Practices (https://docs.docker.com/develop/dev-best-practices/)

Última atualização: Outubro 2024
Versão: 1.0.0