🚀 Deployment Guide - Rental Management MVP

Complete guide for deploying the Rental Management platform to production.

📋 Table of Contents

Prerequisites

Before deploying, ensure you have:

- [ ] Node.js 18+ installed
- [ ] PostgreSQL 14+ database (production-ready)
- [ ] Domain name for your application
- [ ] SSL certificate (or platform-provided)
- [ ] AWS account (optional, for S3 storage)
- [ ] SMTP provider account (optional, for emails)
- [ ] Git repository access

Environment Configuration

1. Generate Secure Secrets

```
# Generate NextAuth secret
openssl rand -base64 32

# Save this for NEXTAUTH_SECRET environment variable
```

2. Required Environment Variables

Create a `.env.production` file with these **required** variables:

```
# Database - Production PostgreSQL
DATABASE_URL="postgresql://user:password@production-host:5432/rental_db?schema=public"

# NextAuth - Your production domain
NEXTAUTH_URL="https://yourdomain.com"

# NextAuth Secret - Generate with: openssl rand -base64 32
NEXTAUTH_SECRET="your-generated-secret-here"

# Environment
NODE_ENV="production"
```

3. Optional Integrations

AWS S3 (Recommended for Production)

```
AWS_REGION="us-east-1"
AWS_ACCESS_KEY_ID="AKIA..."
AWS_SECRET_ACCESS_KEY="..."
AWS_S3_BUCKET="rental-management-prod"
AWS_CLOUDFRONT_DOMAIN="d1234567890.cloudfront.net"  # Optional CDN
```

Setup steps:

1. Create S3 bucket in AWS Console
2. Enable versioning and encryption
3. Create IAM user with S3 permissions
4. Generate access keys
5. Configure CORS policy
6. Install SDK: `npm install @aws-sdk/client-s3`
7. Uncomment S3 code in `/app/api/upload/route.ts`

SMTP Email (Recommended for Production)

```
# Example: Gmail
SMTP_HOST="smtp.gmail.com"
SMTP_PORT="587"
SMTP_USER="your-app@gmail.com"
SMTP_PASSWORD="app-specific-password"
SMTP_FROM_EMAIL="noreply@yourdomain.com"
SMTP_FROM_NAME="Your Business Name"
```

Setup steps:

1. Choose email provider (Gmail, SendGrid, AWS SES)
2. Get SMTP credentials
3. Install: `npm install nodemailer @types/nodemailer`

Database Setup

Option 1: Managed PostgreSQL (Recommended)

Neon (Serverless PostgreSQL)

1. Sign up at neon.tech (https://neon.tech)

2. Create new project
3. Copy connection string
4. Add to `DATABASE_URL`

Supabase

1. Sign up at supabase.com (https://supabase.com)
2. Create new project
3. Get connection string from Settings → Database
4. Add to `DATABASE_URL`

Railway

1. Sign up at railway.app (https://railway.app)
2. Create PostgreSQL service
3. Copy connection string
4. Add to `DATABASE_URL`

Option 2: Self-Hosted PostgreSQL

```
# Connect to your production database
psql -h production-host -U postgres

# Create database
CREATE DATABASE rental_management;

# Create user
CREATE USER rental_user WITH PASSWORD 'secure_password';
GRANT ALL PRIVILEGES ON DATABASE rental_management TO rental_user;
```

Run Migrations

```
# Set DATABASE_URL to production
export DATABASE_URL="postgresql://..."

# Run migrations
npx prisma migrate deploy

# Verify
npx prisma db push
```

Optional: Seed Demo Data

```
# Only for testing production environment
npm run db:seed
```

File Storage Configuration

Local Storage (Default)

Files stored in `/public/uploads/tenant-{tenantId}/`

Pros:
- No additional setup
- Free

Cons:
- Not scalable for multiple servers
- No CDN support
- Backups required

AWS S3 (Production Recommended)

Setup:

1. **Create S3 Bucket**

```
aws s3 mb s3://rental-management-prod --region us-east-1
```

1. **Configure Bucket Policy**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::rental-management-prod/*"
    }
  ]
}
```

1. **Enable CORS**

```
[
  {
    "AllowedHeaders": ["*"],
    "AllowedMethods": ["GET", "PUT", "POST", "DELETE"],
    "AllowedOrigins": ["https://yourdomain.com"],
    "ExposeHeaders": ["ETag"]
  }
]
```

1. **Create IAM User**
   - Go to IAM → Users → Add User
   - Select “Programmatic access”
   - Attach policy: `AmazonS3FullAccess`
   - Save access keys

2. **Optional: CloudFront CDN**
   - Create CloudFront distribution
   - Point to S3 bucket
   - Add custom domain
   - Update `AWS_CLOUDFRONT_DOMAIN`

Email Configuration

Gmail (Simple, Good for Testing)

1. Enable 2-factor authentication
2. Generate App Password: myaccount.google.com/apppasswords (https://myaccount.google.com/apppasswords)
3. Use app password in `SMTP_PASSWORD`

```
SMTP_HOST="smtp.gmail.com"
SMTP_PORT="587"
SMTP_USER="your-email@gmail.com"
SMTP_PASSWORD="your-16-char-app-password"
```

Limits: ~500 emails/day

SendGrid (Recommended for Production)

1. Sign up at sendgrid.com (https://sendgrid.com)
2. Verify domain
3. Create API key
4. Configure:

```
SMTP_HOST="smtp.sendgrid.net"
SMTP_PORT="587"
SMTP_USER="apikey"
SMTP_PASSWORD="your-sendgrid-api-key"
```

Limits: 100 emails/day (free), paid plans available

AWS SES (Best for Scale)

1. Sign up for AWS SES
2. Verify domain and email addresses
3. Request production access (move out of sandbox)
4. Create SMTP credentials
5. Configure:

```
SMTP_HOST="email-smtp.us-east-1.amazonaws.com"
SMTP_PORT="587"
SMTP_USER="your-smtp-username"
SMTP_PASSWORD="your-smtp-password"
```

Limits: 62,000 emails/month (free tier), then $0.10/1,000 emails

Deployment Platforms

Vercel (Recommended for Next.js)

Pros:
- Zero-config deployment
- Automatic HTTPS
- Global CDN
- Preview deployments

Steps:

1. **Install Vercel CLI**

```
npm i -g vercel
```

1. **Login**

```
vercel login
```

1. **Deploy**

```
vercel --prod
```

1. **Configure Environment Variables**
   - Go to Vercel Dashboard → Settings → Environment Variables
   - Add all production variables

2. **Configure Domain**
   - Add custom domain in Vercel Dashboard
   - Update DNS records
   - Wait for SSL provisioning

Environment Variables in Vercel:

```
DATABASE_URL=postgresql://...
NEXTAUTH_URL=https://yourdomain.com
NEXTAUTH_SECRET=...
NODE_ENV=production
AWS_REGION=us-east-1
AWS_ACCESS_KEY_ID=...
AWS_SECRET_ACCESS_KEY=...
AWS_S3_BUCKET=...
SMTP_HOST=...
SMTP_PORT=587
SMTP_USER=...
SMTP_PASSWORD=...
SMTP_FROM_EMAIL=...
```

Railway

Steps:

1. Sign up at railway.app (https://railway.app)

2. Create new project from GitHub repo
3. Add PostgreSQL service
4. Configure environment variables
5. Deploy

Docker + VPS

Dockerfile:

```
FROM node:18-alpine AS builder
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npx prisma generate
RUN npm run build

FROM node:18-alpine AS runner
WORKDIR /app
COPY --from=builder /app/next.config.js ./
COPY --from=builder /app/public ./public
COPY --from=builder /app/.next ./.next
COPY --from=builder /app/node_modules ./node_modules
COPY --from=builder /app/package.json ./package.json
COPY --from=builder /app/prisma ./prisma

EXPOSE 3000
CMD ["npm", "start"]
```

Deploy:

```
docker build -t rental-management .
docker run -p 3000:3000 --env-file .env.production rental-management
```

Subdomain Configuration

Wildcard DNS Setup

Your application uses subdomains for multi-tenancy (e.g., `acme.yourdomain.com`).

Option 1: DNS Configuration (Recommended)

Add wildcard DNS record:

```
Type: A
Name: *
Value: Your-Server-IP
TTL: 3600
```

Or for Vercel/CloudFlare:

```
Type: CNAME
Name: *
Value: cname.vercel-dns.com
TTL: 3600
```

Option 2: Vercel Wildcard Domain

1. Go to Vercel Dashboard → Settings → Domains
2. Add domain: `yourdomain.com`
3. Add wildcard: `*.yourdomain.com`
4. Configure DNS as instructed

Testing Subdomains Locally

Edit `/etc/hosts`:

```
127.0.0.1 demo.localhost
127.0.0.1 acme.localhost
127.0.0.1 test.localhost
```

Then access: `http://demo.localhost:3000`

Security Checklist

Before going live, verify:

Application Security

- [ ] `NEXTAUTH_SECRET` is unique and secure (32+ characters)
- [ ] `NODE_ENV` is set to `production`
- [ ] Database credentials are secure and not exposed
- [ ] AWS credentials have minimal required permissions
- [ ] SMTP credentials are secure
- [ ] `.env` file is in `.gitignore` (verify!)
- [ ] No sensitive data in logs

Database Security

- [ ] Database requires authentication
- [ ] Database uses SSL/TLS connection
- [ ] Database has regular backup schedule
- [ ] Database access limited to application server
- [ ] Strong database password (16+ characters)

File Storage Security

- [ ] S3 bucket is not publicly writable
- [ ] S3 bucket has versioning enabled
- [ ] S3 bucket has encryption enabled
- [ ] CloudFront restricts access to S3 origin (if using)

Network Security

- [ ] HTTPS enforced (no HTTP)
- [ ] SSL certificate is valid
- [ ] CORS configured correctly
- [ ] Rate limiting enabled (if applicable)
- [ ] Firewall rules configured

Monitoring

- [ ] Error tracking configured (e.g., Sentry)
- [ ] Uptime monitoring enabled
- [ ] Database monitoring enabled
- [ ] Log aggregation configured

Post-Deployment

1. Verify Deployment

```
# Check application is running
curl https://yourdomain.com

# Check API health
curl https://yourdomain.com/api/health
```

2. Create First Tenant

Visit: `https://yourdomain.com/onboarding`

Or use seed script:

```
npm run db:seed
```

3. Test Core Functionality

- [ ] Tenant registration
- [ ] User login
- [ ] Add inventory item
- [ ] Create booking
- [ ] Add customer
- [ ] Generate invoice
- [ ] Download PDF

4. Configure DNS

Ensure wildcard subdomain is working:

```
# Test subdomain resolution
nslookup demo.yourdomain.com
nslookup acme.yourdomain.com
```

5. Test Email Sending

- [ ] Registration welcome email
- [ ] Booking confirmation email
- [ ] Check spam folder
- [ ] Verify sender domain

Monitoring & Maintenance

Application Monitoring

Recommended Tools:
- **Sentry** - Error tracking
- **LogRocket** - Session replay
- **Datadog** - Full observability
- **New Relic** - Performance monitoring

Database Backups

Automated Backups (Recommended)

Neon/Supabase/Railway:
- Automatic daily backups included

Self-hosted:

```
# Daily backup script
pg_dump -h host -U user -d database > backup_$(date +%Y%m%d).sql

# Automated with cron
0 2 * * * /path/to/backup-script.sh
```

Uptime Monitoring

Services:
- UptimeRobot (https://uptimerobot.com) - Free
- Pingdom (https://pingdom.com) - Paid
- Better Uptime (https://betteruptime.com) - Free tier

Performance Optimization

1. **Enable Caching**
   - CloudFront for static assets
   - Redis for session storage (future)

2. **Database Optimization**
   - Add indexes for frequent queries
   - Monitor slow queries
   - Use connection pooling

3. **Image Optimization**
   - Use WebP format
   - Implement lazy loading
   - Configure CloudFront compression

Scaling Considerations

When to Scale:
- Response time > 500ms consistently
- Database CPU > 80%
- Memory usage > 80%
- More than 1000 active tenants

Scaling Options:
1. **Vertical Scaling**: Upgrade server/database resources
2. **Horizontal Scaling**: Add more application instances
3. **Database Sharding**: Separate database per tenant (future)
4. **CDN**: Move all static assets to CDN
5. **Caching Layer**: Add Redis for sessions and queries

Troubleshooting

Common Issues

Issue: Database Connection Failed

Solution:
- Verify `DATABASE_URL` is correct
- Check database server is running
- Verify network connectivity
- Check firewall rules

Issue: Subdomain Not Working

Solution:
- Verify wildcard DNS is configured
- Check DNS propagation: `nslookup *.yourdomain.com`
- Clear browser cache
- Wait for DNS propagation (up to 48 hours)

Issue: File Upload Failed

Solution:
- Check S3 credentials
- Verify bucket permissions
- Check CORS configuration
- Verify bucket exists in correct region

Issue: Emails Not Sending

Solution:
- Verify SMTP credentials
- Check spam folder
- Verify sender domain
- Check email provider limits

Issue: NextAuth Error

Solution:
- Verify `NEXTAUTH_URL` matches deployment URL

- Check `NEXTAUTH_SECRET` is set
- Clear cookies and try again
- Check database session table

Rollback Plan

If deployment fails:

1. **Keep Previous Version Running**
   - Don't delete previous deployment until verified
2. **Database Migrations**
   - Migrations are forward-only
   - To rollback, run reverse migration:
   `bash`
   `npx prisma migrate resolve --rolled-back migration_name`
3. **Vercel Rollback**
   `bash`
   `vercel rollback`
4. **Manual Rollback**
   - Revert to previous Git commit
   - Redeploy

Support & Resources

- **Documentation**: `/README.md`, `/SETUP.md`
- **Database Schema**: `/prisma/schema.prisma`
- **API Routes**: `/app/api/`
- **Issue Tracker**: GitHub Issues

Deployment Checklist

Use this before going live:

```
Environment Setup
- [ ] All required environment variables configured
- [ ] NEXTAUTH_SECRET generated and secure
- [ ] Database connection tested
- [ ] S3 credentials configured (if using)
- [ ] SMTP credentials configured (if using)

Database
- [ ] Migrations deployed
- [ ] Backup strategy implemented
- [ ] Connection pooling configured
- [ ] SSL/TLS enabled

Security
- [ ] HTTPS enforced
- [ ] Sensitive data not exposed
- [ ] CORS configured
- [ ] Rate limiting considered
- [ ] Security headers configured

Domain & DNS
- [ ] Custom domain configured
- [ ] Wildcard subdomain configured
- [ ] SSL certificate active
- [ ] DNS propagated

Testing
- [ ] Core features tested
- [ ] Email sending verified
- [ ] File upload verified
- [ ] Subdomain routing verified
- [ ] Load testing performed

Monitoring
- [ ] Error tracking enabled
- [ ] Uptime monitoring active
- [ ] Log aggregation configured
- [ ] Database monitoring enabled

Documentation
- [ ] Deployment notes documented
- [ ] Environment variables documented
- [ ] Rollback plan documented
- [ ] Team notified
```

🎉 **Congratulations! Your Rental Management platform is ready for production.**

For questions or support, refer to the main README.md or open an issue.