AfricanMarket Production Deployment Guide

Table of Contents

Prerequisites

System Requirements

- **Operating System**: Ubuntu 20.04 LTS or newer
- **Memory**: Minimum 8GB RAM (16GB recommended)
- **Storage**: Minimum 100GB SSD
- **CPU**: Minimum 4 cores (8 cores recommended)
- **Network**: Static IP address with domain name

Required Software

- Docker & Docker Compose
- Node.js 18.x
- Yarn package manager
- Git
- Nginx
- PostgreSQL 15+
- Redis 7+
- Certbot (for SSL certificates)

Installation Commands

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Docker
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER

# Install Docker Compose
sudo curl -L "https://github.com/docker/compose/releases/latest/download/docker-com-
pose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose

# Install Node.js and Yarn
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs
npm install -g yarn

# Install additional tools
sudo apt install -y git nginx postgresql-client redis-tools certbot python3-certbot-
nginx
```

External Services Setup

1. Database (Required)

Option A: Managed Database (Recommended)
- **AWS RDS PostgreSQL**
- Instance type: db.t3.medium or larger
- Storage: 100GB GP2 (scalable)
- Multi-AZ deployment for production
- Automated backups enabled

- **Google Cloud SQL PostgreSQL**
- Machine type: db-standard-2 or larger
- Storage: 100GB SSD
- High availability configuration
- Automated backups enabled

Option B: Self-hosted PostgreSQL

```
# Install PostgreSQL
sudo apt install postgresql postgresql-contrib
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Create database and user
sudo -u postgres createdb africanmarket_production
sudo -u postgres createuser -s africanmarket_user
sudo -u postgres psql -c "ALTER USER africanmarket_user PASSWORD
'your_secure_password';"
```

2. Redis Cache (Required)

Option A: Managed Redis (Recommended)
- **AWS ElastiCache Redis**
- **Google Cloud Memorystore**
- **DigitalOcean Managed Redis**

Option B: Self-hosted Redis

```
# Install Redis
sudo apt install redis-server
sudo systemctl start redis
sudo systemctl enable redis

# Configure Redis security
sudo nano /etc/redis/redis.conf
# Set: requirepass your_redis_password
sudo systemctl restart redis
```

3. File Storage - Cloudinary (Required)

1. Sign up at Cloudinary (https://cloudinary.com)
2. Get your credentials:
   - Cloud Name
   - API Key
   - API Secret
3. Create upload preset for production use

4. Payment Processing - Stripe (Required)

1. Sign up at Stripe (https://stripe.com)
2. Get your live keys:
   - Publishable Key (pk_live_...)
   - Secret Key (sk_live_...)
3. Set up webhooks endpoint: `https://your-domain.com/api/webhooks/stripe`
4. Configure Stripe Connect for vendor payouts

5. Email Service - SendGrid (Required)

1. Sign up at SendGrid (https://sendgrid.com)
2. Create API key with Mail Send permissions
3. Set up domain authentication
4. Configure sender identity

6. SMS Service - Twilio (Optional)

1. Sign up at Twilio (https://twilio.com)
2. Get Account SID and Auth Token
3. Purchase phone number for SMS sending

7. Maps Service - Google Maps (Required)

1. Enable Google Maps JavaScript API
2. Enable Google Places API
3. Enable Google Geocoding API

4. Create API key with domain restrictions

8. Monitoring - Sentry (Recommended)

1. Sign up at Sentry (https://sentry.io)
2. Create new project for AfricanMarket
3. Get DSN for error tracking

9. Analytics - Google Analytics (Recommended)

1. Set up Google Analytics 4 property
2. Get Measurement ID (G-XXXXXXXXXX)

Infrastructure Setup

1. Server Provisioning

Recommended Providers:
- **AWS EC2**: t3.large or larger
- **Google Compute Engine**: e2-standard-4 or larger
- **DigitalOcean Droplet**: 8GB Memory / 4 vCPUs
- **Linode**: Dedicated 8GB or larger

2. Domain and DNS Setup

1. Purchase domain name
2. Configure DNS records:
   `A Record: @ -> Your Server IP`
   `  A Record: www -> Your Server IP`
   `  CNAME: api -> Your Domain`

3. Firewall Configuration

```
# Configure UFW firewall
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
sudo ufw enable
```

4. Load Balancer Setup (Optional for High Traffic)

Cloudflare Setup:
1. Add domain to Cloudflare
2. Update nameservers
3. Enable SSL/TLS (Full Strict)
4. Configure caching rules
5. Set up rate limiting

Application Deployment

1. Clone Repository

```
# Clone the application
git clone https://github.com/your-username/africanmarket.git
cd africanmarket/app

# Switch to production branch
git checkout main
```

2. Environment Configuration

```
# Copy environment template
cp .env.production.template .env.production

# Edit environment variables
nano .env.production
```

3. Required Environment Variables

Fill in your `.env.production` file with actual values:

```
# Core Application
NODE_ENV=production
NEXT_PUBLIC_APP_URL=https://your-domain.com
NEXTAUTH_URL=https://your-domain.com
NEXTAUTH_SECRET=your-super-secure-secret-key-min-32-chars

# Database
DATABASE_URL=postgresql://username:password@host:5432/africanmarket_prod

# Redis
REDIS_URL=redis://password@host:6379

# Payment Processing
STRIPE_SECRET_KEY=sk_live_your_stripe_secret_key
STRIPE_PUBLISHABLE_KEY=pk_live_your_stripe_publishable_key
STRIPE_WEBHOOK_SECRET=whsec_your_webhook_secret

# File Storage
CLOUDINARY_CLOUD_NAME=your_cloud_name
CLOUDINARY_API_KEY=your_api_key
CLOUDINARY_API_SECRET=your_api_secret

# Email Service
SENDGRID_API_KEY=SG.your_sendgrid_api_key
FROM_EMAIL=noreply@your-domain.com

# SMS Service (Optional)
TWILIO_ACCOUNT_SID=ACyour_twilio_account_sid
TWILIO_AUTH_TOKEN=your_twilio_auth_token

# External APIs
MAPBOX_ACCESS_TOKEN=pk.your_mapbox_token
GOOGLE_MAPS_API_KEY=your_google_maps_api_key

# Monitoring
SENTRY_DSN=https://your_sentry_dsn@sentry.io/project_id
GOOGLE_ANALYTICS_ID=GA_MEASUREMENT_ID

# Push Notifications
VAPID_PUBLIC_KEY=your_vapid_public_key
VAPID_PRIVATE_KEY=your_vapid_private_key
```

4. Install Dependencies and Build

```
# Install dependencies
yarn install --frozen-lockfile

# Generate Prisma client
yarn prisma generate

# Build the application
yarn build
```

5. Database Setup

```
# Run database migrations
yarn prisma migrate deploy

# Seed the database
yarn prisma db seed
```

6. SSL Certificate Setup

```
# Run SSL setup script
sudo ./security/ssl-setup.sh your-domain.com admin@your-domain.com production
```

7. Start Application with Docker

```
# Start production containers
docker-compose -f docker-compose.yml up -d

# Check container status
docker-compose ps
```

8. Configure Nginx

The SSL setup script should have configured Nginx, but verify:

```
# Test Nginx configuration
sudo nginx -t

# Restart Nginx
sudo systemctl restart nginx

# Enable auto-start
sudo systemctl enable nginx
```

Configuration

1. Production Next.js Configuration

The application includes `next.config.production.js` with optimizations:
- Image optimization enabled
- Security headers configured
- Compression enabled
- Static file caching
- Performance monitoring

2. Docker Configuration

Production Docker setup includes:
- Multi-stage builds for optimization
- Non-root user for security
- Health checks
- Resource limits
- Logging configuration

3. Database Configuration

Ensure your database has:
- Connection pooling configured
- Query logging for monitoring
- Automated backups enabled
- Performance monitoring
- SSL connections enforced

SSL and Security

1. SSL Certificate Management

```
# Test certificate renewal
sudo certbot renew --dry-run

# View certificate status
sudo certbot certificates

# Manually renew if needed
sudo certbot renew
```

2. Security Headers

Verify security headers are active:

```
curl -I https://your-domain.com | grep -E "(Strict-Transport-Security|X-Frame-Options|
X-Content-Type-Options)"
```

3. Firewall Rules

```
# Check firewall status
sudo ufw status

# View open ports
sudo netstat -tuln
```

Monitoring Setup

1. Application Monitoring

```
# Start monitoring services
node monitoring/sentry-config.js
node monitoring/performance-monitor.js
node monitoring/health-check.js
```

2. Health Check Verification

```
# Test health endpoint
curl https://your-domain.com/api/health

# Test detailed health check
curl https://your-domain.com/api/health?detailed=true
```

3. Log Management

```
# View application logs
docker-compose logs -f app

# View database logs
docker-compose logs -f postgres

# View nginx logs
sudo tail -f /var/log/nginx/access.log
sudo tail -f /var/log/nginx/error.log
```

4. Backup Configuration

```
# Test backup system
./scripts/backup.sh production full

# Verify backup files
ls -la backups/

# Test restore process (use test database)
./scripts/restore.sh backups/latest.sql.gz staging
```

Testing and Validation

1. Smoke Tests

Run these tests after deployment:

```
# Test application startup
curl -f https://your-domain.com/api/health

# Test authentication
curl -X POST https://your-domain.com/api/auth/signin \
  -H "Content-Type: application/json" \
  -d '{"email":"test@example.com","password":"password"}'

# Test database connectivity
docker exec africanmarket_postgres psql -U postgres -c "SELECT version();"
```

2. Performance Tests

```
# Install testing tools
npm install -g artillery

# Run load test
artillery run performance-test.yml
```

3. Security Tests

```
# Test SSL configuration
nmap --script ssl-enum-ciphers -p 443 your-domain.com

# Test security headers
curl -I https://your-domain.com
```

4. Functional Tests

Test these critical user journeys:
- [ ] User registration and email verification
- [ ] User login and logout
- [ ] Browse restaurants and menus
- [ ] Add items to cart and checkout
- [ ] Payment processing
- [ ] Order tracking
- [ ] Vendor order management
- [ ] Driver delivery workflow

Troubleshooting

Common Issues and Solutions

1. Application Won't Start

Symptoms: Container exits immediately or won't start
Solutions:

```
# Check logs
docker-compose logs app

# Verify environment variables
docker exec -it africanmarket_app env | grep DATABASE_URL

# Check database connectivity
docker exec -it africanmarket_app yarn prisma db ping
```

2. Database Connection Issues

Symptoms: Cannot connect to database
Solutions:

```
# Test database connection
psql $DATABASE_URL -c "SELECT version();"

# Check database server status
docker-compose ps postgres

# Verify database credentials
echo $DATABASE_URL
```

3. SSL Certificate Issues

Symptoms: HTTPS not working or certificate errors
Solutions:

```
# Check certificate status
sudo certbot certificates

# Renew certificate
sudo certbot renew

# Check nginx configuration
sudo nginx -t

# Verify DNS records
nslookup your-domain.com
```

4. High Memory Usage

Symptoms: Out of memory errors or slow performance
Solutions:

```
# Check memory usage
free -h
docker stats

# Restart application
docker-compose restart app

# Scale up server resources
# (resize your server instance)
```

5. Payment Processing Issues

Symptoms: Payment failures or webhook errors
Solutions:

```
# Check Stripe webhook logs
curl -H "Authorization: Bearer $STRIPE_SECRET_KEY" \
  "https://api.stripe.com/v1/webhook_endpoints"

# Verify webhook endpoint
curl -X POST https://your-domain.com/api/webhooks/stripe \
  -H "Content-Type: application/json"

# Check Stripe dashboard for errors
```

6. Email Delivery Issues

Symptoms: Emails not being sent or delivered

Solutions:

```
# Test SendGrid API
curl -X POST "https://api.sendgrid.com/v3/mail/send" \
  -H "Authorization: Bearer $SENDGRID_API_KEY" \
  -H "Content-Type: application/json"

# Check SendGrid activity logs
# Verify sender authentication
# Check spam folder
```

Diagnostic Commands

System Health

```
# Check disk space
df -h

# Check memory usage
free -m

# Check CPU usage
top

# Check network connectivity
ping google.com
```

Application Health

```
# Check all containers
docker-compose ps

# Check container resource usage
docker stats

# View application logs
docker-compose logs -f app

# Check application metrics
curl https://your-domain.com/api/health?detailed=true
```

Database Health

```
# Check database size
docker exec africanmarket_postgres psql -U postgres -c "SELECT
pg_size_pretty(pg_database_size('africanmarket_prod'));"

# Check active connections
docker exec africanmarket_postgres psql -U postgres -c "SELECT count(*) FROM
pg_stat_activity;"

# Check slow queries
docker exec africanmarket_postgres psql -U postgres -c "SELECT query, mean_time, calls
FROM pg_stat_statements ORDER BY mean_time DESC LIMIT 10;"
```

Maintenance

Daily Tasks

- [ ] Check application health dashboard
- [ ] Review error logs and alerts
- [ ] Monitor resource usage
- [ ] Verify backup completion
- [ ] Check security alerts

Weekly Tasks

- [ ] Review performance metrics
- [ ] Analyze user feedback
- [ ] Update security patches
- [ ] Test backup restoration
- [ ] Review monitoring alerts

Monthly Tasks

- [ ] Security audit and updates
- [ ] Performance optimization review
- [ ] Capacity planning assessment
- [ ] Database maintenance
- [ ] SSL certificate renewal check

Quarterly Tasks

- [ ] Disaster recovery testing
- [ ] Full security penetration testing
- [ ] Infrastructure cost optimization
- [ ] Team training and documentation updates
- [ ] Technology stack updates

Update Procedures

Application Updates

```
# Backup current state
./scripts/backup.sh production full

# Pull latest changes
git pull origin main

# Install dependencies
yarn install

# Run migrations
yarn prisma migrate deploy

# Build application
yarn build

# Deploy with zero downtime
docker-compose up -d --no-deps app
```

Security Updates

```
# Update system packages
sudo apt update && sudo apt upgrade

# Update Docker images
docker-compose pull

# Restart containers
docker-compose up -d
```

Database Maintenance

```
# Update database statistics
docker exec africanmarket_postgres psql -U postgres -c "ANALYZE;"

# Vacuum database
docker exec africanmarket_postgres psql -U postgres -c "VACUUM;"

# Reindex if needed
docker exec africanmarket_postgres psql -U postgres -c "REINDEX DATABASE africanmar-
ket_prod;"
```

Scaling Procedures

Vertical Scaling (Upgrading Server)

1. Schedule maintenance window
2. Create full backup
3. Stop application
4. Resize server instance
5. Start application
6. Verify functionality

Horizontal Scaling (Multiple Servers)

1. Set up load balancer (Nginx/Cloudflare)
2. Configure shared database
3. Set up shared file storage
4. Deploy application to multiple servers
5. Configure session storage (Redis)
6. Test load distribution

Monitoring and Alerting Setup

Key Metrics to Monitor

- Application uptime and response time
- Database performance and connections
- Memory and CPU usage
- Disk space utilization
- Error rates and exceptions
- User activity and business metrics

Alert Configuration

Set up alerts for:
- Application downtime > 5 minutes
- Error rate > 1%
- Response time > 3 seconds
- CPU usage > 80%
- Memory usage > 90%
- Disk space > 85%

Backup and Recovery

Automated Backup Schedule

- **Hourly**: Application logs
- **Daily**: Database incremental backup
- **Weekly**: Full database backup
- **Monthly**: Complete system backup

Recovery Testing

- **Monthly**: Test database restoration
- **Quarterly**: Test full system recovery
- **Annually**: Complete disaster recovery drill

Support and Resources

Documentation

- Next.js Deployment Guide (https://nextjs.org/docs/deployment)
- Prisma Production Guide (https://www.prisma.io/docs/guides/deployment)
- Docker Production Guide (https://docs.docker.com/config/containers/start-containers-automatically/)

Community Support

- AfricanMarket GitHub Issues (https://github.com/your-repo/issues)
- Technical Support Email
- Developer Community Slack (https://slack.africanmarket.com)

Emergency Contacts

- **Technical Emergency**: +1-XXX-XXX-XXXX
- **On-Call Engineer**: engineer@africanmarket.com
- **System Administrator**: admin@africanmarket.com

This deployment guide should be kept up to date with any changes to the application architecture, dependencies, or deployment procedures.