Maritime Reservation System - Deployment Guide

Overview

This deployment guide provides comprehensive instructions for deploying the Maritime Reservation System to production environments. The guide covers infrastructure setup, application deployment, configuration management, and operational procedures.

Prerequisites

System Requirements

- **Operating System:** Ubuntu 22.04 LTS or CentOS 8+
- **Memory:** Minimum 8GB RAM (16GB recommended)
- **Storage:** Minimum 100GB SSD storage
- **CPU:** Minimum 4 cores (8 cores recommended)
- **Network:** High-speed internet connection with static IP

Required Software

- Docker 24.0+
- Docker Compose 2.0+
- Node.js 20.x
- Python 3.11+
- PostgreSQL 15+
- Redis 7.0+
- Nginx 1.22+

Infrastructure Setup

1. Server Preparation

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

# Install required packages
```

```
sudo apt install -y curl wget git unzip nginx postgresql-client
redis-tools

# 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-compose-$(uname -s)-$(uname -m)" -o /usr/local/
bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

2. Database Setup

```
 # Install PostgreSQL
sudo apt install -y postgresql postgresql-contrib

# Create database and user
sudo -u postgres psql << EOF
CREATE DATABASE maritime_reservations;
CREATE USER maritime_user WITH PASSWORD 'secure_password_here';
GRANT ALL PRIVILEGES ON DATABASE maritime_reservations TO
maritime_user;
ALTER USER maritime_user CREATEDB;
\q
EOF
```

3. Redis Setup

```
 # Install Redis
sudo apt install -y redis-server

# Configure Redis
sudo systemctl enable redis-server
sudo systemctl start redis-server

# Secure Redis
echo "requirepass your_redis_password_here" | sudo tee -a /etc/
redis/redis.conf
sudo systemctl restart redis-server
```

Application Deployment

1. Clone Repository

```
# Clone the application repository
git clone https://github.com/your-org/maritime-reservation-
system.git
cd maritime-reservation-system
```

2. Environment Configuration

Create production environment files:

Backend Environment (.env.prod):

```
# Database Configuration
DATABASE_URL=postgresql://
maritime_user:secure_password_here@localhost:5432/
maritime_reservations
REDIS_URL=redis://:your_redis_password_here@localhost:6379/0

# Security Configuration
SECRET_KEY=your_very_secure_secret_key_here
JWT_SECRET_KEY=your_jwt_secret_key_here
ENCRYPTION_KEY=your_encryption_key_here

# API Configuration
API_V1_STR=/api/v1
PROJECT_NAME="Maritime Reservations"
BACKEND_CORS_ORIGINS=["https://your-domain.com"]

# Payment Configuration
STRIPE_PUBLISHABLE_KEY=pk_live_your_stripe_key
STRIPE_SECRET_KEY=sk_live_your_stripe_secret
STRIPE_WEBHOOK_SECRET=whsec_your_webhook_secret

PAYPAL_CLIENT_ID=your_paypal_client_id
PAYPAL_CLIENT_SECRET=your_paypal_client_secret
PAYPAL_MODE=live

# Ferry Operator APIs
CTN_API_KEY=your_ctn_api_key
CTN_API_URL=https://api.ctn.com.tn
GNV_API_KEY=your_gnv_api_key
GNV_API_URL=https://api.gnv.it

# Email Configuration
SMTP_HOST=smtp.your-provider.com
```

```
SMTP_PORT=587
SMTP_USER=your_email@your-domain.com
SMTP_PASSWORD=your_email_password
EMAILS_FROM_EMAIL=noreply@your-domain.com

# Monitoring
SENTRY_DSN=your_sentry_dsn_here
LOG_LEVEL=INFO
```

Frontend Environment (.env.production):

```
 REACT_APP_API_URL=https://api.your-domain.com
REACT_APP_STRIPE_PUBLISHABLE_KEY=pk_live_your_stripe_key
REACT_APP_PAYPAL_CLIENT_ID=your_paypal_client_id
REACT_APP_GOOGLE_ANALYTICS_ID=your_ga_id
REACT_APP_SENTRY_DSN=your_frontend_sentry_dsn
```

3. Backend Deployment

```
 # Navigate to backend directory
cd maritime_reservation_backend

# Create virtual environment
python3 -m venv venv
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Run database migrations
alembic upgrade head

# Create initial admin user
python scripts/create_admin.py

# Test the application
python -m pytest tests/

# Start the application with Gunicorn
gunicorn main:app -w 4 -k uvicorn.workers.UvicornWorker --bind
0.0.0.0:8000
```

4. Frontend Deployment

```
 # Navigate to frontend directory
cd ../maritime-reservation-frontend
```

```
# Install dependencies
npm install

# Build for production
npm run build

# Copy build files to web server
sudo cp -r dist/* /var/www/html/
```

Docker Deployment (Recommended)

1. Docker Compose Configuration

Create `docker-compose.prod.yml`:

```
 version: '3.8'

services:
  db:
    image: postgres:15
    environment:
      POSTGRES_DB: maritime_reservations
      POSTGRES_USER: maritime_user
      POSTGRES_PASSWORD: secure_password_here
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./backups:/backups
    ports:
      - "5432:5432"
    restart: unless-stopped

  redis:
    image: redis:7-alpine
    command: redis-server --requirepass your_redis_password_here
    volumes:
      - redis_data:/data
    ports:
      - "6379:6379"
    restart: unless-stopped

  backend:
    build:
      context: ./maritime_reservation_backend
      dockerfile: Dockerfile.prod
    environment:
      - DATABASE_URL=postgresql://
maritime_user:secure_password_here@db:5432/maritime_reservations
```

```
      - REDIS_URL=redis://:your_redis_password_here@redis:6379/0
    env_file:
      - .env.prod
    ports:
      - "8000:8000"
    depends_on:
      - db
      - redis
    volumes:
      - ./logs:/app/logs
    restart: unless-stopped

  frontend:
    build:
      context: ./maritime-reservation-frontend
      dockerfile: Dockerfile.prod
    ports:
      - "3000:80"
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/ssl:/etc/nginx/ssl
      - ./logs/nginx:/var/log/nginx
    depends_on:
      - backend
      - frontend
    restart: unless-stopped

volumes:
  postgres_data:
  redis_data:
```

2. Nginx Configuration

Create `nginx/nginx.conf`:

```
 events {
    worker_connections 1024;
}

http {
    upstream backend {
        server backend:8000;
    }
    upstream frontend {
        server frontend:80;
    }

    # Rate limiting
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;
    limit_req_zone $binary_remote_addr zone=general:10m
rate=30r/s;

    server {
        listen 80;
        server_name your-domain.com www.your-domain.com;
        return 301 https://$server_name$request_uri;
    }

    server {
        listen 443 ssl http2;
        server_name your-domain.com www.your-domain.com;

        ssl_certificate /etc/nginx/ssl/fullchain.pem;
        ssl_certificate_key /etc/nginx/ssl/privkey.pem;
        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-
GCM-SHA512;
        ssl_prefer_server_ciphers off;

        # Security headers
        add_header X-Frame-Options DENY;
        add_header X-Content-Type-Options nosniff;
        add_header X-XSS-Protection "1; mode=block";
        add_header Strict-Transport-Security "max-age=63072000;
includeSubDomains; preload";

        # API routes
        location /api/ {
            limit_req zone=api burst=20 nodelay;
            proxy_pass http://backend;
            proxy_set_header Host $host;
            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;
        }

        # Admin routes
        location /admin {
            limit_req zone=general burst=10 nodelay;
            proxy_pass http://frontend;
            proxy_set_header Host $host;
            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;
        }

        # Frontend routes
        location / {
            limit_req zone=general burst=20 nodelay;
            proxy_pass http://frontend;
            proxy_set_header Host $host;
            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;
        }

        # Static files caching
        location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg)$ {
            expires 1y;
            add_header Cache-Control "public, immutable";
        }
    }
}
```

3. Deploy with Docker

```
# Build and start services
docker-compose -f docker-compose.prod.yml up -d

# Check service status
docker-compose -f docker-compose.prod.yml ps

# View logs
docker-compose -f docker-compose.prod.yml logs -f
```

SSL Certificate Setup

Using Let's Encrypt (Recommended)

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

# Obtain SSL certificate
sudo certbot --nginx -d your-domain.com -d www.your-domain.com

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

```
# Set up automatic renewal
echo "0 12 * * * /usr/bin/certbot renew --quiet" | sudo crontab
-
```

Monitoring and Logging

1. Application Monitoring

```
 # Install monitoring tools
pip install prometheus-client grafana-api

# Configure Prometheus metrics endpoint
# Add to main.py:
from prometheus_client import Counter, Histogram,
generate_latest
from fastapi import Response

REQUEST_COUNT = Counter('requests_total', 'Total requests',
['method', 'endpoint'])
REQUEST_LATENCY = Histogram('request_duration_seconds',
'Request latency')

@app.get("/metrics")
async def metrics():
    return Response(generate_latest(), media_type="text/plain")
```

2. Log Management

```
 # Configure log rotation
sudo tee /etc/logrotate.d/maritime-reservations << EOF
/var/log/maritime-reservations/*.log {
    daily
    missingok
    rotate 52
    compress
    delaycompress
    notifempty
    create 644 www-data www-data
    postrotate
        systemctl reload nginx
    endscript
}
EOF
```

3. Health Checks

Create `scripts/health_check.sh`:

```
#!/bin/bash

# Check backend health
BACKEND_STATUS=$(curl -s -o /dev/null -w "%{http_code}" http://
localhost:8000/health)
if [ $BACKEND_STATUS -ne 200 ]; then
    echo "Backend health check failed: $BACKEND_STATUS"
    exit 1
fi

# Check database connection
DB_STATUS=$(PGPASSWORD=secure_password_here psql -h localhost -
U maritime_user -d maritime_reservations -c "SELECT 1;" 2>/dev/
null)
if [ $? -ne 0 ]; then
    echo "Database health check failed"
    exit 1
fi

# Check Redis connection
REDIS_STATUS=$(redis-cli -a your_redis_password_here ping 2>/
dev/null)
if [ "$REDIS_STATUS" != "PONG" ]; then
    echo "Redis health check failed"
    exit 1
fi

echo "All health checks passed"
```

Backup and Recovery

1. Database Backup

Create `scripts/backup_db.sh`:

```
#!/bin/bash

BACKUP_DIR="/backups"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="$BACKUP_DIR/maritime_reservations_$DATE.sql"

# Create backup directory
mkdir -p $BACKUP_DIR
```

```
# Perform backup
PGPASSWORD=secure_password_here pg_dump -h localhost -U
maritime_user maritime_reservations > $BACKUP_FILE

# Compress backup
gzip $BACKUP_FILE

# Remove backups older than 30 days
find $BACKUP_DIR -name "*.sql.gz" -mtime +30 -delete

echo "Backup completed: $BACKUP_FILE.gz"
```

2. Automated Backups

```
 # Add to crontab
echo "0 2 * * * /path/to/scripts/backup_db.sh" | crontab -
```

3. Recovery Procedure

```
 # Stop application
docker-compose -f docker-compose.prod.yml stop backend

# Restore database
PGPASSWORD=secure_password_here psql -h localhost -U
maritime_user -d maritime_reservations < backup_file.sql

# Start application
docker-compose -f docker-compose.prod.yml start backend
```

Security Hardening

1. Firewall Configuration

```
 # Configure UFW firewall
sudo ufw default deny incoming
sudo ufw default allow outgoing
sudo ufw allow ssh
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable
```

2. System Security

```
# Disable root login
sudo sed -i 's/PermitRootLogin yes/PermitRootLogin no/' /etc/
ssh/sshd_config

# Configure fail2ban
sudo apt install -y fail2ban
sudo systemctl enable fail2ban
sudo systemctl start fail2ban
```

3. Application Security

```
# Set proper file permissions
sudo chown -R www-data:www-data /var/www/html
sudo chmod -R 755 /var/www/html

# Secure configuration files
sudo chmod 600 .env.prod
sudo chown root:root .env.prod
```

Performance Optimization

1. Database Optimization

```
-- Create indexes for better performance
CREATE INDEX idx_bookings_user_id ON bookings(user_id);
CREATE INDEX idx_bookings_departure_date ON
bookings(departure_date);
CREATE INDEX idx_bookings_status ON bookings(status);
CREATE INDEX idx_ferries_route ON ferries(departure_port,
arrival_port);

-- Configure PostgreSQL for production
-- Add to postgresql.conf:
shared_buffers = 256MB
effective_cache_size = 1GB
maintenance_work_mem = 64MB
checkpoint_completion_target = 0.9
wal_buffers = 16MB
default_statistics_target = 100
random_page_cost = 1.1
effective_io_concurrency = 200
```

2. Application Optimization

```bash
# Configure Gunicorn for production
gunicorn main:app \
  --workers 4 \
  --worker-class uvicorn.workers.UvicornWorker \
  --bind 0.0.0.0:8000 \
  --max-requests 1000 \
  --max-requests-jitter 100 \
  --timeout 30 \
  --keep-alive 2
```

Troubleshooting

Common Issues

1. **Database Connection Issues** ```bash # Check PostgreSQL status sudo systemctl status postgresql

# Check connection PGPASSWORD=secure_password_here psql -h localhost -U maritime_user -d maritime_reservations -c "SELECT version();" ```

1. **Redis Connection Issues** ```bash # Check Redis status sudo systemctl status redis-server

# Test connection redis-cli -a your_redis_password_here ping ```

1. **SSL Certificate Issues** ```bash # Check certificate validity openssl x509 -in /etc/nginx/ssl/fullchain.pem -text -noout

# Renew certificate sudo certbot renew ```

Log Locations

- **Application logs:** `/var/log/maritime-reservations/`
- **Nginx logs:** `/var/log/nginx/`
- **PostgreSQL logs:** `/var/log/postgresql/`
- **System logs:** `/var/log/syslog`

Maintenance Procedures

Regular Maintenance Tasks

1. **Weekly Tasks**
2. Review application logs
3. Check system resource usage
4. Verify backup integrity
5. Update security patches
6. **Monthly Tasks**
7. Database maintenance (VACUUM, ANALYZE)
8. Review performance metrics
9. Update dependencies
10. Security audit
11. **Quarterly Tasks**
12. Full system backup test
13. Disaster recovery test
14. Performance optimization review
15. Security penetration testing

This deployment guide provides comprehensive instructions for setting up and maintaining a production Maritime Reservation System. Follow these procedures carefully and adapt them to your specific infrastructure requirements.