Hotel Management System - Deployment Guide

Overview

This guide provides step-by-step instructions for deploying the Hotel Management System in various environments, from local development to production Kubernetes clusters.

Prerequisites

System Requirements

- **Operating System**: Linux (Ubuntu 20.04+ recommended), macOS, or Windows with WSL2
- **Memory**: Minimum 8GB RAM (16GB recommended for production)
- **Storage**: Minimum 20GB free space
- **Network**: Internet connection for downloading dependencies

Required Software

- **Docker**: Version 20.10 or higher
- **Docker Compose**: Version 2.0 or higher
- **Node.js**: Version 18 or higher (for local development)
- **npm**: Version 8 or higher
- **Git**: Latest version
- **kubectl**: For Kubernetes deployments
- **k6**: For load testing (optional)

Installation Methods

Method 1: Docker Compose (Recommended for Development)

Step 1: Clone the Repository

```
git clone <repository-url>
cd hotel-management-app
```

Step 2: Environment Configuration

Create environment files for each service:

Backend Services (.env files):

```
# auth-service/.env
MONGODB_URI=mongodb://admin:password@mongodb:27017/hotel-auth?
authSource=admin
JWT_SECRET=your-super-secret-jwt-key-change-in-production
REDIS_URL=redis://redis:6379
PORT=3001

# room-service/.env
MONGODB_URI=mongodb://admin:password@mongodb:27017/hotel-rooms?
authSource=admin
PORT=3002

# reservation-service/.env
MONGODB_URI=mongodb://admin:password@mongodb:27017/hotel-
reservations?authSource=admin
ROOM_SERVICE_URL=http://room-service:3002
PORT=3003
```

Step 3: Build and Start Services

```
# Build all services
docker-compose build

# Start all services
docker-compose up -d

# Check service status
docker-compose ps
```

Step 4: Verify Installation

```
# Check if all services are running
curl http://localhost:3001/graphql
curl http://localhost:3002/graphql
curl http://localhost:3003/graphql
curl http://localhost:80  # Frontend
```

Step 5: Initialize Database (Optional)

```
# Connect to MongoDB and create initial admin user
docker exec -it hotel-mongodb mongo -u admin -p password --
authenticationDatabase admin

# In MongoDB shell:
use hotel-auth
db.users.insertOne({
  email: "admin@hotel.com",
  password: "$2b$10$hashed_password_here",
  firstName: "Admin",
  lastName: "User",
  role: "admin",
  createdAt: new Date(),
  updatedAt: new Date()
})
```

Method 2: Kubernetes Deployment (Production)

Step 1: Prepare Kubernetes Cluster

Ensure you have a running Kubernetes cluster with: - Minimum 3 nodes - 8GB RAM per node - Storage class for persistent volumes

Step 2: Create Namespace

```
kubectl create namespace hotel-management
kubectl config set-context --current --namespace=hotel-
management
```

Step 3: Apply Secrets

```
# Update secrets with your values
kubectl apply -f k8s/secrets.yaml
```

Step 4: Deploy Infrastructure

```
# Deploy MongoDB
kubectl apply -f k8s/mongodb.yaml

# Deploy Redis
kubectl apply -f k8s/redis.yaml

# Wait for infrastructure to be ready
kubectl wait --for=condition=ready pod -l app=mongodb --
timeout=300s
kubectl wait --for=condition=ready pod -l app=redis --
timeout=300s
```

Step 5: Deploy Microservices

```
# Deploy Auth Service
kubectl apply -f k8s/auth-service.yaml

# Deploy Room Service
kubectl apply -f k8s/room-service.yaml

# Deploy Reservation Service
kubectl apply -f k8s/reservation-service.yaml

# Wait for services to be ready
kubectl wait --for=condition=ready pod -l app=auth-service --
timeout=300s
kubectl wait --for=condition=ready pod -l app=room-service --
timeout=300s
kubectl wait --for=condition=ready pod -l app=reservation-
service --timeout=300s
```

Step 6: Deploy Frontend

```
# Deploy Frontend
kubectl apply -f k8s/frontend.yaml

# Get external IP
kubectl get service frontend
```

Step 7: Configure Ingress (Optional)

```
# ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
```

```yaml
metadata:
  name: hotel-management-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: hotel.yourdomain.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend
            port:
              number: 80
      - path: /api/auth
        pathType: Prefix
        backend:
          service:
            name: auth-service
            port:
              number: 3001
      - path: /api/rooms
        pathType: Prefix
        backend:
          service:
            name: room-service
            port:
              number: 3002
      - path: /api/reservations
        pathType: Prefix
        backend:
          service:
            name: reservation-service
            port:
              number: 3003
```

```
kubectl apply -f ingress.yaml
```

Method 3: Local Development Setup

Step 1: Install Dependencies

```bash
# Install Node.js dependencies for each service
cd backend/auth-service && npm install
cd ../room-service && npm install
cd ../reservation-service && npm install
cd ../../frontend/hotel-frontend-react && npm install
```

Step 2: Start Infrastructure Services

```
# Start MongoDB, Redis using Docker
docker run -d --name mongodb -p 27017:27017 -e
MONGO_INITDB_ROOT_USERNAME=admin -e
MONGO_INITDB_ROOT_PASSWORD=password mongo:latest
docker run -d --name redis -p 6379:6379 redis:alpine
```

Step 3: Start Backend Services

```
# Terminal 1: Auth Service
cd backend/auth-service
npm run start:dev

# Terminal 2: Room Service
cd backend/room-service
npm run start:dev

# Terminal 3: Reservation Service
cd backend/reservation-service
npm run start:dev
```

Step 4: Start Frontend

```
# Terminal 4: Frontend
cd frontend/hotel-frontend-react
npm run dev
```

Configuration

Environment Variables

Authentication Service

- `MONGODB_URI` : MongoDB connection string
- `JWT_SECRET` : Secret key for JWT token signing
- `REDIS_URL` : Redis connection URL
- `PORT` : Service port (default: 3001)

Room Service

- `MONGODB_URI` : MongoDB connection string
- `PORT` : Service port (default: 3002)

Reservation Service

- `MONGODB_URI` : MongoDB connection string
- `ROOM_SERVICE_URL` : Room service URL for inter-service communication
- `PORT` : Service port (default: 3003)

Database Configuration

MongoDB Collections

- `hotel-auth.users` : User accounts and authentication data
- `hotel-rooms.rooms` : Room inventory and details
- `hotel-reservations.reservations` : Booking and reservation data

Indexes (Recommended for Production)

```
 // Users collection
db.users.createIndex({ email: 1 }, { unique: true })
db.users.createIndex({ role: 1 })

// Rooms collection
db.rooms.createIndex({ roomNumber: 1 }, { unique: true })
db.rooms.createIndex({ availability: 1 })
db.rooms.createIndex({ roomType: 1 })

// Reservations collection
db.reservations.createIndex({ userId: 1 })
db.reservations.createIndex({ roomId: 1 })
db.reservations.createIndex({ checkInDate: 1, checkOutDate: 1 })
db.reservations.createIndex({ status: 1 })
```

Monitoring and Logging

Prometheus and Grafana Setup

```
 # Start monitoring stack
docker-compose up -d prometheus grafana

# Access Grafana
# URL: http://localhost:3000
# Username: admin
# Password: admin
```

Log Aggregation

For production deployments, consider implementing: - **ELK Stack** (Elasticsearch, Logstash, Kibana) - **Fluentd** for log collection - **Jaeger** for distributed tracing

Security Considerations

Production Security Checklist

- [ ] Change default passwords and secrets
- [ ] Enable HTTPS/TLS encryption
- [ ] Configure firewall rules
- [ ] Set up VPN for database access
- [ ] Enable audit logging
- [ ] Implement rate limiting
- [ ] Configure CORS properly
- [ ] Use secrets management (e.g., HashiCorp Vault)
- [ ] Enable container security scanning
- [ ] Set up backup and disaster recovery

Network Security

```yaml
# Network policy example
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: hotel-management-network-policy
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
```

```
    - protocol: TCP
      port: 3001
```

Troubleshooting

Common Issues

Service Won't Start

```
# Check logs
docker-compose logs <service-name>
kubectl logs -l app=<service-name>

# Check resource usage
docker stats
kubectl top pods
```

Database Connection Issues

```
# Test MongoDB connection
docker exec -it hotel-mongodb mongo -u admin -p password --
authenticationDatabase admin

# Check network connectivity
kubectl exec -it <pod-name> -- nslookup mongodb
```

Frontend Not Loading

```
# Check if backend services are accessible
curl http://localhost:3001/graphql
curl http://localhost:3002/graphql
curl http://localhost:3003/graphql

# Check browser console for errors
# Verify CORS configuration
```

Performance Tuning

Database Optimization

- Enable MongoDB sharding for large datasets
- Configure appropriate connection pooling
- Implement read replicas for read-heavy workloads

Application Optimization

- Enable Redis caching for frequently accessed data
- Implement GraphQL query complexity analysis
- Configure horizontal pod autoscaling in Kubernetes

Backup and Recovery

Database Backup

```
# MongoDB backup
docker exec hotel-mongodb mongodump --uri="mongodb://
admin:password@localhost:27017/?authSource=admin" --out=/backup

# Restore from backup
docker exec hotel-mongodb mongorestore --uri="mongodb://
admin:password@localhost:27017/?authSource=admin" /backup
```

Kubernetes Backup

```
# Backup Kubernetes resources
kubectl get all -o yaml > hotel-management-backup.yaml

# Backup persistent volumes
kubectl get pv,pvc -o yaml > hotel-management-storage-
backup.yaml
```

Scaling

Horizontal Scaling

```
# Scale microservices
kubectl scale deployment auth-service --replicas=5
kubectl scale deployment room-service --replicas=3
kubectl scale deployment reservation-service --replicas=4

# Auto-scaling configuration
kubectl autoscale deployment auth-service --cpu-percent=70 --
min=2 --max=10
```

Load Balancing

Configure load balancers for: - Frontend (NGINX or cloud load balancer) - API Gateway (Kong, Ambassador, or Istio) - Database (MongoDB replica sets)

Support and Maintenance

Health Checks

All services expose health check endpoints: - `GET /health` - Basic health check - `GET /metrics` - Prometheus metrics

Monitoring Alerts

Set up alerts for: - High CPU/memory usage - Database connection failures - API response time degradation - Error rate increases

Regular Maintenance

- Update dependencies monthly
- Review and rotate secrets quarterly
- Perform security audits
- Monitor and optimize database performance
- Review and update backup procedures