LAWVRIKSH BLOG MANAGEMENT SYSTEM

COMPREHENSIVE DEPLOYMENT PIPELINE DOCUMENTATION

Executive Summary

This document outlines a comprehensive deployment pipeline for the LawVriksh Blog Management System, a FastAPI-based application with SQLAlchemy ORM. The pipeline encompasses development, testing, staging, and production environments with automated CI/CD processes, security measures, and monitoring solutions.

📋 Table of Contents

1. PROJECT ARCHITECTURE OVERVIEW

Technology Stack:

Backend Framework:
FastAPI 0.104.1

Database:
SQLite (Development) / PostgreSQL (Production)

ORM:
SQLAlchemy 2.0.23

Authentication:
JWT tokens with python-jose

Security:
bcrypt password hashing

API Documentation:
Auto-generated OpenAPI/Swagger

Key Components:

- Authentication Module (JWT-based)
- Posts Management (CRUD operations)
- Comments System
- Likes/Reactions System
- RESTful API endpoints

2. ENVIRONMENT STRATEGY

2.1 Development Environment

- Local development with SQLite database
- Hot-reload enabled for rapid development
- Docker Compose for local services
- Environment variables for configuration

2.2 Testing Environment

- Automated unit and integration tests
- Test database with fixtures
- API endpoint testing with pytest
- Security testing for authentication flows

2.3 Staging Environment

- Production-like environment
- PostgreSQL database
- Full integration testing
- Performance testing with load simulation

2.4 Production Environment

- High availability setup
- PostgreSQL with read replicas
- Load balancing with multiple instances
- CDN integration for static assets
- SSL/TLS encryption

3. CONTAINERIZATION STRATEGY

3.1 Dockerfile Configuration

```
FROM python:3.11-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    postgresql-client \
    && rm -rf /var/lib/apt/lists/*

# Install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY src/ ./src/
COPY main.py .

# Create non-root user
RUN useradd -r -s /bin/false appuser
USER appuser

# Expose port
EXPOSE 8000

# Health check
HEALTHCHECK --interval=30s --timeout=30s --start-period=5s --retries=3 \
    CMD python -c "import requests;
requests.get('http://localhost:8000/health')"

# Run application
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

3.2 Docker Compose for Development

```
version: '3.8'

services:
  web:
    build: .
    ports:
      - "8000:8000"
    environment:
      - DATABASE_URL=postgresql://user:password@db:5432/blogdb
    depends_on:
      - db
    volumes:
      - ./src:/app/src

  db:
    image: postgres:15-alpine
    environment:
      - POSTGRES_USER=user
      - POSTGRES_PASSWORD=password
      - POSTGRES_DB=blogdb
    ports:
      - "5432:5432"
    volumes:
      - postgres_data:/var/lib/postgresql/data

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"

volumes:
  postgres_data:
```

4. CI/CD PIPELINE CONFIGURATION

4.1 GitHub Actions Workflow

```yaml
name: CI/CD Pipeline

on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  test:
    runs-on: ubuntu-latest

    services:
      postgres:
        image: postgres:15
        env:
          POSTGRES_PASSWORD: postgres
          POSTGRES_DB: test_db
        options: >-
          --health-cmd pg_isready
          --health-interval 10s
          --health-timeout 5s
          --health-retries 5
        ports:
          - 5432:5432

    steps:
    - uses: actions/checkout@v4

    - name: Set up Python
```

```
    uses: actions/setup-python@v4
    with:
      python-version: '3.11'

  - name: Install dependencies
    run: |
      python -m pip install --upgrade pip
      pip install -r requirements.txt
      pip install pytest pytest-asyncio

  - name: Run tests
    env:
      DATABASE_URL: postgresql://postgres:postgres@localhost:5432/test_db
    run: |
      pytest tests/ -v

  - name: Run security scan
    uses: securecodewarrior/github-action-add-sarif@v1
    with:
      sarif-file: security-scan.sarif
```

5. DATABASE MIGRATION STRATEGY

5.1 Migration Tool: Alembic

```
# Initialize Alembic
alembic init alembic

# Create migration
alembic revision --autogenerate -m "Initial migration"

# Run migration
alembic upgrade head
```

5.2 Database Migration Strategy

- **Development:** Auto-migration enabled
- **Staging:** Manual migration review
- **Production:** Blue-green deployment with migration testing

5.3 Database Backup Strategy

- Daily automated backups
- Point-in-time recovery capability
- Cross-region backup replication
- Backup testing procedures

6. SECURITY IMPLEMENTATION

6.1 Authentication & Authorization

- JWT token-based authentication
- Role-based access control (RBAC)
- API rate limiting
- CORS configuration

6.2 Security Headers

- HTTPS enforcement
- Security headers (HSTS, CSP, X-Frame-Options)
- Input validation and sanitization
- SQL injection prevention

6.3 Secrets Management

- Environment variables for sensitive data
- Docker secrets for production
- AWS Secrets Manager integration
- Regular secret rotation policies

6.4 Security Scanning

- Dependency vulnerability scanning
- Container image scanning
- Static code analysis
- Penetration testing

10. ENVIRONMENT CONFIGURATIONS

10.1 Development Environment (.env.development)

```
DATABASE_URL=sqlite:///./dev.db SECRET_KEY=dev-secret-key-change-in-
production DEBUG=True CORS_ORIGINS=["http://localhost:3000",
"http://localhost:8080"]
```

10.2 Staging Environment (.env.staging)

```
DATABASE_URL=postgresql://user:password@staging-db:5432/blogdb
SECRET_KEY=staging-secret-key DEBUG=False CORS_ORIGINS=
["https://staging.lawvriksh.com"]
```

10.3 Production Environment (.env.production)

```
DATABASE_URL=postgresql://user:password@prod-db:5432/blogdb
SECRET_KEY=production-secret-key-from-vault DEBUG=False CORS_ORIGINS=
["https://lawvriksh.com", "https://www.lawvriksh.com"]
```

11. DEPLOYMENT PROCEDURES

11.1 Development Deployment

```
# Local development
docker-compose up -d

# Run migrations
alembic upgrade head

# Seed database
python scripts/seed_db.py
```

11.2 Staging Deployment

```
# Deploy to staging
git push origin develop

# Manual verification
curl -f https://staging.lawvriksh.com/health

# Run smoke tests
pytest tests/smoke/ -v
```

11.3 Production Deployment

```
# Create release
git tag -a v1.0.0 -m "Release version 1.0.0"
git push origin v1.0.0

# Deploy to production
git push origin main
```

```
# Post-deployment verification
./scripts/post-deploy-checks.sh
```

CONCLUSION

This deployment pipeline provides a robust, scalable, and secure approach to deploying the LawVriksh Blog Management System. The strategy encompasses modern DevOps practices, security best practices, and operational excellence principles. Regular reviews and updates of this pipeline will ensure continued alignment with business objectives and technological advancements.

> **Success Metrics:**
>
> - Deployment frequency
> - Lead time for changes
> - Mean time to recovery
> - Change failure rate
>
> All metrics should be continuously monitored and improved.

The pipeline is designed to support both current requirements and future growth, with built-in flexibility for scaling and enhancement.