# Deployment & Operations Guide - Spotify Follow-Swarm

## Production Architecture

### Infrastructure Overview

```
┌─────────────────────────────────────────────────────────────┐
│                    CloudFlare CDN                           │
└─────────────────────────────────────────────────────────────┘
                         │
                         ▼
┌─────────────────────────────────────────────────────────────┐
│                 AWS/GCP Load Balancer                       │
└─────────────────────────────────────────────────────────────┘
                         │
            ┌────────────┴────────────┐
            ▼                         ▼
    ┌───────────────┐         ┌───────────────┐
    │  Web Servers  │         │  API Servers  │
    │  (React App)  │         │ (Node/Python) │
    └───────────────┘         └───────────────┘
                                     │
                  ┌──────────────────┼──────────────────┐
                  ▼                  ▼                  ▼
        ┌──────────────┐  ┌──────────────┐  ┌──────────────┐
        │  PostgreSQL  ││    Redis     ││ Queue Worker │
        │  (Primary)   ││   (Cache)    ││   Servers    │
        └──────────────┘  └──────────────┘  └──────────────┘
                │
        ┌──────────────┐
        │  PostgreSQL  │
        │(Read Replica)│
        └──────────────┘
```

## Docker Configuration

### Docker Compose (Development)

yaml

```yaml
# docker-compose.yml
version: '3.8'

services:
  postgres:
    image: postgres:14
    environment:
      POSTGRES_DB: spotify_swarm
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: ${DB_PASSWORD}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data

  api:
    build:
      context: ./backend
      dockerfile: Dockerfile
    environment:
      DATABASE_URL: postgresql://admin:${DB_PASSWORD}@postgres:5432/spotify_swarm
      REDIS_URL: redis://redis:6379
      SPOTIFY_CLIENT_ID: ${SPOTIFY_CLIENT_ID}
      SPOTIFY_CLIENT_SECRET: ${SPOTIFY_CLIENT_SECRET}
    depends_on:
      - postgres
      - redis
    ports:
      - "3001:3001"
    volumes:
      - ./backend:/app
      - /app/node_modules

  worker:
    build:
      context: ./backend
      dockerfile: Dockerfile.worker
    environment:
      DATABASE_URL: postgresql://admin:${DB_PASSWORD}@postgres:5432/spotify_swarm
      REDIS_URL: redis://redis:6379
    depends_on:
      - postgres
      - redis
      - api
    volumes:
      - ./backend:/app
      - /app/node_modules

  frontend:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    environment:
      REACT_APP_API_URL: http://localhost:3001
    ports:
      - "3000:3000"
    volumes:
      - ./frontend:/app
      - /app/node_modules

volumes:
  postgres_data:
  redis_data:
```

## Production Dockerfiles

### Backend API Dockerfile

```dockerfile
```

```dockerfile
# backend/Dockerfile
FROM node:18-alpine AS builder

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production

FROM node:18-alpine

WORKDIR /app
COPY --from=builder /app/node_modules ./node_modules
COPY . .

# Run as non-root user
RUN addgroup -g 1001 -S nodejs && \
  adduser -S nodejs -u 1001
USER nodejs

EXPOSE 3001
CMD ["node", "server.js"]
```

### Worker Dockerfile

dockerfile

```dockerfile
# backend/Dockerfile.worker
FROM node:18-alpine

WORKDIR /app
COPY package*.json ./
RUN npm ci --only=production
COPY . .

# Run as non-root user
RUN addgroup -g 1001 -S nodejs && \
  adduser -S nodejs -u 1001
USER nodejs

CMD ["node", "worker.js"]
```

## Kubernetes Deployment

## API Deployment

yaml

```yaml
# k8s/api-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: api
  template:
    metadata:
      labels:
        app: api
    spec:
      containers:
      - name: api
        image: spotify-swarm-api:latest
        ports:
        - containerPort: 3001
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: app-secrets
              key: database-url
        - name: REDIS_URL
          valueFrom:
            secretKeyRef:
              name: app-secrets
              key: redis-url
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        livenessProbe:
          httpGet:
            path: /health
      port: 3001
    initialDelaySeconds: 30
    periodSeconds: 10
  readinessProbe:
    httpGet:
      path: /ready
      port: 3001
    initialDelaySeconds: 5
    periodSeconds: 5
```

## Worker Deployment

yaml

```yaml
# k8s/worker-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: worker-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: worker
  template:
    metadata:
      labels:
        app: worker
    spec:
      containers:
      - name: worker
        image: spotify-swarm-worker:latest
        env:
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: app-secrets
              key: database-url
        - name: REDIS_URL
          valueFrom:
            secretKeyRef:
              name: app-secrets
              key: redis-url
        resources:
          requests:
            memory: "512Mi"
            cpu: "500m"
          limits:
            memory: "1Gi"
            cpu: "1000m"
```

## Environment Configuration

### Production Environment Variables

```bash
# .env.production
# Application
NODE_ENV=production
APP_URL=https://spotifyswarm.com
API_URL=https://api.spotifyswarm.com

# Database
DATABASE_URL=postgresql://user:pass@db.amazonaws.com:5432/spotify_swarm
DATABASE_POOL_MIN=2
DATABASE_POOL_MAX=10

# Redis
REDIS_URL=redis://redis.amazonaws.com:6379
REDIS_PASSWORD=your_redis_password

# Spotify API
SPOTIFY_CLIENT_ID=your_client_id
SPOTIFY_CLIENT_SECRET=your_client_secret
SPOTIFY_REDIRECT_URI=https://spotifyswarm.com/auth/callback

# Security
JWT_SECRET=your_jwt_secret_key
ENCRYPTION_KEY=your_encryption_key
SESSION_SECRET=your_session_secret

# Monitoring
SENTRY_DSN=https://key@sentry.io/project
NEW_RELIC_LICENSE_KEY=your_license_key
DATADOG_API_KEY=your_api_key

# Email
SENDGRID_API_KEY=your_sendgrid_key
EMAIL_FROM=noreply@spotifyswarm.com

# Payments
STRIPE_SECRET_KEY=sk_live_your_key
STRIPE_WEBHOOK_SECRET=whsec_your_secret
```

## CI/CD Pipeline

## GitHub Actions Workflow

yaml

```yaml
# .github/workflows/deploy.yml
name: Deploy to Production

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

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Setup Node.js
        uses: actions/setup-node@v2
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test

      - name: Run linting
        run: npm run lint

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Build Docker images
        run: |
          docker build -t spotify-swarm-api:${{ github.sha }} ./backend
          docker build -t spotify-swarm-worker:${{ github.sha }} ./backend -f Dockerfile.worker
          docker build -t spotify-swarm-frontend:${{ github.sha }} ./frontend
      - name: Push to Registry
        env:
          DOCKER_REGISTRY: ${{ secrets.DOCKER_REGISTRY }}
        run: |
          echo ${{ secrets.DOCKER_PASSWORD }} | docker login -u ${{ secrets.DOCKER_USERNAME }} --pass
          docker push spotify-swarm-api:${{ github.sha }}
          docker push spotify-swarm-worker:${{ github.sha }}
          docker push spotify-swarm-frontend:${{ github.sha }}

  deploy:
    needs: build
    runs-on: ubuntu-latest
    if: github.ref == 'refs/heads/main'
    steps:
      - name: Deploy to Kubernetes
        env:
          KUBE_CONFIG: ${{ secrets.KUBE_CONFIG }}
        run: |
          echo "$KUBE_CONFIG" | base64 -d > kubeconfig
          export KUBECONFIG=kubeconfig
          kubectl set image deployment/api-deployment api=spotify-swarm-api:${{ github.sha }}
          kubectl set image deployment/worker-deployment worker=spotify-swarm-worker:${{ github.sha }}
          kubectl rollout status deployment/api-deployment
          kubectl rollout status deployment/worker-deployment
```

## Monitoring & Alerting

### Health Check Endpoints

```javascript
```

```
// health-checks.js
app.get('/health', (req, res) => {
  res.status(200).json({ status: 'healthy' });
});

app.get('/ready', async (req, res) => {
  try {
    // Check database
    await db.query('SELECT 1');

    // Check Redis
    await redis.ping();

    // Check Spotify API
    const token = await getSystemToken();
    if (!token) throw new Error('No system token');

    res.status(200).json({
      status: 'ready',
      services: {
        database: 'connected',
        redis: 'connected',
        spotify: 'authenticated'
      }
    });
  } catch (error) {
    res.status(503).json({
      status: 'not ready',
      error: error.message
    });
  }
});

app.get('/metrics', async (req, res) => {
  // Prometheus metrics endpoint
  res.set('Content-Type', register.contentType);
  res.end(await register.metrics());
});
```

### Monitoring Stack Configuration

yaml

```yaml
# monitoring/prometheus.yml
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'api'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_label_app]
        action: keep
        regex: api
      - source_labels: [__meta_kubernetes_pod_ip]
        target_label: __address__
        replacement: $1:3001

  - job_name: 'worker'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_label_app]
        action: keep
        regex: worker
```

## Alert Rules

yaml

```yaml
# monitoring/alerts.yml
groups:
  - name: spotify_swarm
    rules:
      - alert: HighErrorRate
        expr: rate(http_requests_total{status=~"5.."}[5m]) > 0.05
        for: 5m
        annotations:
          summary: High error rate detected
          description: "Error rate is {{ $value }} errors per second"

      - alert: SlowAPIResponse
        expr: histogram_quantile(0.95, rate(http_request_duration_seconds_bucket[5m])) > 1
        for: 10m
        annotations:
          summary: API response time is slow
          description: "95th percentile response time is {{ $value }} seconds"

      - alert: QueueBacklog
        expr: queue_size{status="pending"} > 1000
        for: 15m
        annotations:
          summary: Large queue backlog
          description: "{{ $value }} jobs pending in queue"

      - alert: RateLimitNearLimit
        expr: rate(spotify_api_calls_total[1h]) > 2700
        for: 5m
        annotations:
          summary: Approaching Spotify rate limit
          description: "Current rate: {{ $value }} calls per hour"
```

## Database Management

### Backup Strategy

```bash
```

```
#!/bin/bash
# backup.sh

# Daily backup script
BACKUP_DIR="/backups"
DATE=$(date +%Y%m%d_%H%M%S)
DB_NAME="spotify_swarm"

# Create backup
pg_dump $DATABASE_URL > $BACKUP_DIR/backup_$DATE.sql

# Compress backup
gzip $BACKUP_DIR/backup_$DATE.sql

# Upload to S3
aws s3 cp $BACKUP_DIR/backup_$DATE.sql.gz s3://spotify-swarm-backups/

# Delete local backups older than 7 days
find $BACKUP_DIR -name "backup_*.sql.gz" -mtime +7 -delete

# Delete S3 backups older than 30 days
aws s3 ls s3://spotify-swarm-backups/ | \
  while read -r line; do
    createDate=$(echo $line | awk {'print $1" "$2'})
    createDate=$(date -d"$createDate" +%s)
    olderThan=$(date -d"30 days ago" +%s)
    if [[ $createDate -lt $olderThan ]]; then
      fileName=$(echo $line | awk {'print $4'})
      aws s3 rm s3://spotify-swarm-backups/$fileName
    fi
  done
```

## Migration Strategy

```
javascript
```

```
// migrations/run.js
const { Pool } = require('pg');
const fs = require('fs');
const path = require('path');

async function runMigrations() {
  const pool = new Pool({ connectionString: process.env.DATABASE_URL });

  // Create migrations table
  await pool.query(`
    CREATE TABLE IF NOT EXISTS migrations (
      id SERIAL PRIMARY KEY,
      filename VARCHAR(255) UNIQUE NOT NULL,
      executed_at TIMESTAMP DEFAULT NOW()
    )
  `);

  // Get pending migrations
  const files = fs.readdirSync(path.join(__dirname, 'sql'));
  const executed = await pool.query('SELECT filename FROM migrations');
  const executedFiles = executed.rows.map(r => r.filename);

  const pending = files.filter(f => !executedFiles.includes(f)).sort();

  // Run pending migrations
  for (const file of pending) {
    console.log(`Running migration: ${file}`);
    const sql = fs.readFileSync(path.join(__dirname, 'sql', file), 'utf8');

    await pool.query('BEGIN');
    try {
      await pool.query(sql);
      await pool.query('INSERT INTO migrations (filename) VALUES ($1)', [file]);
      await pool.query('COMMIT');
      console.log(`✓ Migration ${file} completed`);
    } catch (error) {
      await pool.query('ROLLBACK');
      console.error(`✗ Migration ${file} failed:`, error);
      process.exit(1);
    }
  }
  await pool.end();
}

runMigrations();
```

## Security Checklist

### Pre-Deployment Security Audit

- [ ] All dependencies updated to latest secure versions
- [ ] Environment variables properly secured
- [ ] Database connections use SSL
- [ ] API endpoints have rate limiting
- [ ] Input validation on all user inputs
- [ ] XSS protection headers configured
- [ ] CSRF tokens implemented
- [ ] Content Security Policy configured
- [ ] HTTPS enforced everywhere
- [ ] Secrets rotated from development

### Runtime Security Monitoring

```
javascript
```

```javascript
// security-monitor.js
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
const mongoSanitize = require('express-mongo-sanitize');

// Security middleware
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'unsafe-inline'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:", "https:"],
    },
  },
}));

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
});
app.use('/api/', limiter);

// Input sanitization
app.use(mongoSanitize());

// Security logging
app.use((req, res, next) => {
  if (req.path.includes('admin') || req.method !== 'GET') {
    logger.info({
      type: 'security',
      method: req.method,
      path: req.path,
      ip: req.ip,
      user: req.user?.id
    });
  }
  next();
});
```

## Performance Optimization

### Caching Strategy

```javascript
```

```
// cache-manager.js
class CacheManager {
  constructor(redis) {
    this.redis = redis;
    this.ttl = {
      user: 3600,       // 1 hour
      follows: 300,     // 5 minutes
      analytics: 1800,  // 30 minutes
      static: 86400     // 24 hours
    };
  }

  async get(key, fetcher, ttl) {
    // Try cache first
    const cached = await this.redis.get(key);
    if (cached) {
      return JSON.parse(cached);
    }

    // Fetch and cache
    const data = await fetcher();
    await this.redis.setex(
      key,
      ttl || this.ttl.static,
      JSON.stringify(data)
    );

    return data;
  }

  async invalidate(pattern) {
    const keys = await this.redis.keys(pattern);
    if (keys.length > 0) {
      await this.redis.del(...keys);
    }
  }
}
```

## Database Optimization

```
sql
```

```
-- Indexes for performance
CREATE INDEX idx_follows_follower_status ON follows(follower_id, status);
CREATE INDEX idx_follows_followed_status ON follows(followed_id, status);
CREATE INDEX idx_queue_jobs_status ON queue_jobs(status, scheduled_for);
CREATE INDEX idx_analytics_user_created ON analytics(user_id, created_at);

-- Partitioning for analytics table
CREATE TABLE analytics_2024_01 PARTITION OF analytics
FOR VALUES FROM ('2024-01-01') TO ('2024-02-01');

-- Query optimization views
CREATE MATERIALIZED VIEW user_stats AS
SELECT
  u.id,
  COUNT(DISTINCT f1.followed_id) as following_count,
  COUNT(DISTINCT f2.follower_id) as follower_count,
  MAX(f1.completed_at) as last_follow_at
FROM users u
LEFT JOIN follows f1 ON f1.follower_id = u.id AND f1.status = 'completed'
LEFT JOIN follows f2 ON f2.followed_id = u.id AND f2.status = 'completed'
GROUP BY u.id;

-- Refresh materialized view periodically
CREATE OR REPLACE FUNCTION refresh_user_stats()
RETURNS void AS $$
BEGIN
  REFRESH MATERIALIZED VIEW CONCURRENTLY user_stats;
END;
$$ LANGUAGE plpgsql;
```

## Disaster Recovery

### Rollback Procedure

```bash
```

```bash
#!/bin/bash
# rollback.sh

PREVIOUS_VERSION=$1

if [ -z "$PREVIOUS_VERSION" ]; then
  echo "Usage: ./rollback.sh <version>"
  exit 1
fi

echo "Rolling back to version $PREVIOUS_VERSION"

# Rollback Kubernetes deployments
kubectl set image deployment/api-deployment api=spotify-swarm-api:$PREVIOUS_VERSION
kubectl set image deployment/worker-deployment worker=spotify-swarm-worker:$PREVIOUS_VERSION

# Wait for rollout
kubectl rollout status deployment/api-deployment
kubectl rollout status deployment/worker-deployment

# Clear cache
redis-cli FLUSHALL

echo "Rollback complete"
```

### Incident Response Plan

1. **Detection** - Alert triggered via monitoring
2. **Assessment** - Check dashboards and logs
3. **Communication** - Notify team and update status page
4. **Mitigation** - Apply immediate fix or rollback
5. **Resolution** - Deploy permanent fix
6. **Post-mortem** - Document and learn from incident

## Scaling Guidelines

### Horizontal Scaling Triggers

- CPU usage > 70% for 5 minutes

- Memory usage > 80% for 5 minutes
- Request latency p95 > 1 second
- Queue depth > 5000 jobs

### Vertical Scaling Recommendations

- **API Servers:** Start with 2 CPU, 4GB RAM
- **Workers:** Start with 4 CPU, 8GB RAM
- **Database:** Start with db.t3.medium (2 vCPU, 4GB)
- **Redis:** Start with cache.t3.micro (2 vCPU, 0.5GB)