KingdomQuest DevOps & Deployment Guide

This comprehensive guide covers the complete DevOps workflow, CI/CD pipeline, and deployment strategies for the KingdomQuest application.

Table of Contents

- Quick Start Guide
- CI/CD Pipeline
- Environment Management
- Deployment Procedures
- Production Infrastructure
- Monitoring & Maintenance
- Troubleshooting
- Security Best Practices

Quick Start Guide

Prerequisites

- **Node.js**: 18.x or 20.x (recommended: 20.x)
- **Package Manager**: pnpm 8.15.1+ (preferred) or npm
- **Git**: Latest version
- **Environment Variables**: Copy `.env.example` to `.env.local` and configure

Local Development Setup

1. **Clone and Install**
   `bash git clone <repository-url> cd kingdom-quest pnpm install`
2. **Environment Configuration**
   `bash cp .env.example .env.local # Edit .env.local with your development credentials`
3. **Database Setup**
   ```bash
   # Initialize Supabase locally (optional)
   pnpm dlx supabase start

# Or use remote Supabase project
# Configure NEXT_PUBLIC_SUPABASE_URL and NEXT_PUBLIC_SUPABASE_ANON_KEY
```

1. **Start Development Server**
   `bash pnpm run dev`
2. **Run Tests**
   ```bash
   # Unit tests
   pnpm run test

# E2E tests
pnpm run test:e2e

# All tests with coverage
pnpm run test:coverage
```

CI/CD Pipeline

Our GitHub Actions workflow provides comprehensive automated testing and deployment with the following stages:

Pipeline Overview

```
┌─────────────────────┐
│   Security Scan      │
│   • Dependency Check  │
│   • Vulnerability Scan │
├─────────────────────┤
│        Lint           │
│   • ESLint           │
│   • Prettier         │
│   • TypeScript       │
├─────────────────────┤
│        Test           │
│   • Unit Tests       │
│   • Component Tests  │
│   • Coverage Report  │
├─────────────────────┤
│       Build          │
│   • Next.js Build    │
│   • Size Analysis    │
│   • Artifact Upload  │
├─────────────────────┤
│        E2E           │
│   • Playwright       │
│   • Cross-Browser    │
│   • Visual Testing   │
├─────────────────────┤
│      Deploy         │
│   • Preview (PRs)    │
│   • Staging (dev)    │
│   • Production (main) │
└─────────────────────┘
```

Pipeline Features

Performance Optimizations:
- Matrix testing across Node.js 18.x and 20.x
- Parallel job execution for maximum efficiency
- Intelligent caching (pnpm store, Next.js build cache)
- Conditional job execution based on branch and file changes

Quality Gates:
- Zero-tolerance linting policy (max-warnings: 0)
- Comprehensive test coverage reporting
- Cross-browser E2E testing (Chromium, Firefox, WebKit)
- Build size analysis and optimization alerts

Security & Compliance:
- Automated dependency vulnerability scanning
- Code quality analysis with Super Linter
- Secrets scanning and validation
- Security policy enforcement

Workflow Triggers

Trigger	Branches	Jobs Executed
Push	`main`, `develop`, `staging`	All jobs + Production/Staging deploy
Pull Request	`main`, `develop`	All jobs + Preview deploy
Manual	Any branch	All jobs (no deploy)

Environment Variables Setup

Required GitHub Secrets:

```
# Supabase Configuration
NEXT_PUBLIC_SUPABASE_URL
NEXT_PUBLIC_SUPABASE_ANON_KEY
SUPABASE_SERVICE_ROLE_KEY

# Google Maps
NEXT_PUBLIC_GOOGLE_MAPS_API_KEY

# Deployment
VERCEL_TOKEN
VERCEL_ORG_ID
VERCEL_PROJECT_ID
VERCEL_STAGING_PROJECT_ID
VERCEL_TEAM_ID

# Testing (Optional)
E2E_SUPABASE_URL
E2E_SUPABASE_ANON_KEY
CODECOV_TOKEN
```

Environment Management

Environment Strategy

We implement a three-tier environment approach:

Environment	Branch	URL	Purpose
Development	feature/*	localhost:3000	Local development
Preview	PR branches	pr-{number}.vercel.app	Feature testing
Staging	develop	staging-kingdom-quest.vercel.app	Integration testing
Production	main	kingdom-quest.vercel.app	Live application

Environment Configuration

Development (.env.local)

```
# Use development/test credentials
NEXT_PUBLIC_SUPABASE_URL=https://dev-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=dev_key
NEXT_PUBLIC_GOOGLE_MAPS_API_KEY=dev_maps_key
NODE_ENV=development
```

Staging

```
# Use staging-specific resources
NEXT_PUBLIC_SUPABASE_URL=https://staging-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=staging_key
NEXT_PUBLIC_GOOGLE_MAPS_API_KEY=staging_maps_key
NODE_ENV=production
```

Production

```
# Use production resources with restrictions
NEXT_PUBLIC_SUPABASE_URL=https://prod-project.supabase.co
NEXT_PUBLIC_SUPABASE_ANON_KEY=prod_key
NEXT_PUBLIC_GOOGLE_MAPS_API_KEY=prod_maps_key
NODE_ENV=production
```

Secrets Management

Best Practices:

1. **Separate Projects**: Use different Supabase projects for each environment
2. **Key Rotation**: Regularly rotate API keys and secrets
3. **IP Restrictions**: Enable IP-based restrictions where possible
4. **Minimal Permissions**: Grant only necessary permissions to each key
5. **Monitoring**: Track API usage and set up alerts for unusual activity

Validation Script:

```
# Validate environment variables
npx tsx scripts/validate-env.ts
```

Deployment Procedures

Automatic Deployments

Pull Request (Preview)

1. Create PR against `main` or `develop`
2. CI/CD pipeline runs automatically
3. Preview environment deployed on success
4. PR comment added with preview URL
5. Environment cleaned up when PR is closed

Staging Deployment

1. Merge changes to `develop` branch
2. Automatic deployment to staging environment

3. Integration tests run against staging
4. Manual approval required for production

Production Deployment

1. Merge `develop` into `main` via PR
2. Automatic deployment to production
3. Health checks and smoke tests
4. Rollback capabilities available

Manual Deployments

Emergency Hotfix

```
# Create hotfix branch from main
git checkout main
git pull origin main
git checkout -b hotfix/critical-fix

# Make changes and commit
git add .
git commit -m "fix: critical issue"
git push origin hotfix/critical-fix

# Create PR directly to main
# After approval, merge triggers production deployment
```

Local Preview

```
# Build and preview locally
pnpm run build
pnpm run start

# Deploy to personal Vercel account
vercel --token=YOUR_TOKEN
```

Rollback Procedures

Vercel Rollback

```
# List recent deployments
vercel list --token=YOUR_TOKEN

# Promote previous deployment
vercel promote DEPLOYMENT_URL --token=YOUR_TOKEN
```

Git Rollback

```
# Revert last commit and deploy
git revert HEAD
git push origin main
```

Production Infrastructure

Current Architecture (Vercel + Supabase)

```
┌───────────────────┐
│   Vercel (CDN)      │
│   • Next.js App       │
│   • Static Assets     │
│   • Edge Functions    │
├───────────────────┤
│   Supabase (BaaS)   │
│   • PostgreSQL DB   │
│   • Authentication    │
│   • File Storage      │
│   • Edge Functions    │
└───────────────────┘
```

Future Architecture (Fly.io + PostgreSQL + S3)

```
┌─────────────────────────────────┐
│           Cloudflare (CDN)           │
├─────────────────────────────────┤
│    Fly.io (App Platform)            │
│    • Next.js Application             │
│    • API Routes                      │
│    • Auto-scaling                    │
├─────────────────────────────────┤
│    Fly.io PostgreSQL                 │
│    • Primary Database                │
│    • Automated Backups              │
│    • Connection Pooling             │
├─────────────────────────────────┤
│    AWS S3 / Cloudflare R2          │
│    • File Storage                   │
│    • Image Processing               │
│    • CDN Integration               │
└─────────────────────────────────┘
```

Infrastructure Setup Guide

Vercel Configuration (Current)

Project Setup:

```
# Install Vercel CLI
npm install -g vercel

# Link to existing project
vercel link --project=kingdom-quest

# Configure environment variables
vercel env add NEXT_PUBLIC_SUPABASE_URL production
vercel env add NEXT_PUBLIC_SUPABASE_ANON_KEY production
vercel env add NEXT_PUBLIC_GOOGLE_MAPS_API_KEY production

# Deploy
vercel --prod
```

Vercel Configuration (vercel.json):

```json
{
  "buildCommand": "pnpm run build",
  "outputDirectory": ".next",
  "framework": "nextjs",
  "installCommand": "pnpm install",
  "env": {
    "PNPM_VERSION": "8.15.1"
  },
  "functions": {
    "app/api/**/*.ts": {
      "maxDuration": 30
    }
  },
  "headers": [
    {
      "source": "/(.*)",
      "headers": [
        {
          "key": "X-Content-Type-Options",
          "value": "nosniff"
        },
        {
          "key": "X-Frame-Options",
          "value": "DENY"
        },
        {
          "key": "X-XSS-Protection",
          "value": "1; mode=block"
        }
      ]
    }
  ]
}
```

Fly.io Setup (Future)

Application Configuration (fly.toml):

```
app = "kingdom-quest"
primary_region = "iad"

[build]
  dockerfile = "Dockerfile.production"

[env]
  NODE_ENV = "production"
  NEXT_TELEMETRY_DISABLED = "1"

[http_service]
  internal_port = 3000
  force_https = true
  auto_stop_machines = true
  auto_start_machines = true
  min_machines_running = 1
  processes = ["app"]

[[vm]]
  cpu_kind = "performance"
  cpus = 2
  memory_mb = 4096

[deploy]
  release_command = "npm run db:migrate"
```

Database Setup:

```
# Create PostgreSQL cluster
flyctl postgres create --name kingdom-quest-db --region iad

# Attach to application
flyctl postgres attach --app kingdom-quest kingdom-quest-db

# Configure connection pooling
flyctl postgres config update --max-connections 200 kingdom-quest-
db
```

Alternative: Render Setup

Web Service:
- **Runtime**: Node.js 20
- **Build Command**: `pnpm install && pnpm run build`
- **Start Command**: `pnpm start`
- **Auto-Deploy**: Yes (from main branch)

Database:
- **Type**: PostgreSQL 15
- **Plan**: Starter (7/month)orPro(20/month)
- **Backup**: Daily automated backups

Monitoring & Maintenance

Health Checks

API Health Endpoint (`/api/health`):

```
// app/api/health/route.ts
export async function GET() {
  const checks = {
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    database: await checkDatabase(),
    supabase: await checkSupabase(),
    external_apis: await checkExternalAPIs()
  };

  const isHealthy = Object.values(checks).every(check =>
    typeof check === 'boolean' ? check : true
  );

  return Response.json(checks, {
    status: isHealthy ? 200 : 503
  });
}
```

Monitoring Checklist:
- [ ] Application uptime and response times
- [ ] Database connection and query performance
- [ ] Supabase service availability
- [ ] Google Maps API quota and usage
- [ ] Error rates and types
- [ ] Build and deployment success rates
- [ ] User authentication and session management

Performance Monitoring

Metrics to Track:
- **Web Vitals**: LCP, FID, CLS, TTFB
- **Build Performance**: Build time, bundle size
- **API Performance**: Response times, error rates
- **Database Performance**: Query times, connection pool usage

Tools:
- **Vercel Analytics**: Built-in performance monitoring
- **Sentry**: Error tracking and performance monitoring
- **LogRocket**: Session recording and debugging
- **Google Analytics**: User behavior and conversion tracking

Maintenance Tasks

Weekly:
- [ ] Review error logs and fix critical issues
- [ ] Check API usage and quotas
- [ ] Monitor build and deployment success rates
- [ ] Review security alerts and dependencies

Monthly:
- [ ] Update dependencies and security patches
- [ ] Review and rotate API keys
- [ ] Analyze performance metrics and optimize
- [ ] Backup and test disaster recovery procedures

Quarterly:
- [ ] Comprehensive security audit
- [ ] Infrastructure cost optimization review
- [ ] Performance benchmark comparison
- [ ] Documentation and runbook updates

Log Management

Centralized Logging:

```
// lib/logger.ts
import winston from 'winston';

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  transports: [
    new winston.transports.Console(),
    new winston.transports.File({ filename: 'app.log' })
  ]
});

export default logger;
```

Log Levels:
- **ERROR**: Critical issues requiring immediate attention
- **WARN**: Important issues that should be addressed
- **INFO**: General application events
- **DEBUG**: Detailed information for troubleshooting

Troubleshooting

Common Issues

Build Failures

TypeScript Errors:

```
# Check types without building
npx tsc --noEmit

# Fix common issues
npm run lint:fix
npm run format
```

Dependency Issues:

```
# Clear cache and reinstall
rm -rf node_modules pnpm-lock.yaml .next
pnpm install
```

Environment Variable Issues:

```
# Validate environment variables
npx tsx scripts/validate-env.ts

# Check Next.js configuration
npm run build -- --debug
```

Deployment Issues

Vercel Deployment Failures:

```
# Check deployment logs
vercel logs --follow

# Redeploy with debug info
vercel --debug

# Check function logs
vercel logs --scope=functions
```

Database Connection Issues:

```
# Test Supabase connection
curl -H "apikey: YOUR_ANON_KEY" "YOUR_SUPABASE_URL/rest/v1/"

# Check database migrations
pnpm dlx supabase db push
```

Performance Issues

Slow Page Loads:

1. Check bundle size: `npm run analyze`
2. Optimize images: Use Next.js Image component
3. Enable compression: Verify gzip/brotli compression
4. Database optimization: Review query performance

High Memory Usage:

1. Profile memory usage: Use browser dev tools
2. Check for memory leaks: Review component cleanup
3. Optimize images: Reduce image sizes and use WebP
4. Bundle analysis: Remove unused dependencies

Debug Commands

Development Debugging:

```
# Start with debugging enabled
NODE_OPTIONS="--inspect" npm run dev

# Enable verbose logging
DEBUG="*" npm run dev

# Analyze bundle
npm run build && npm run analyze
```

Production Debugging:

```
# Check production build locally
npm run build && npm run start

# Verify environment variables
node -e "console.log(process.env)"

# Test API endpoints
curl -v http://localhost:3000/api/health
```

Support Contacts

Technical Issues:
- GitHub Issues: Repository issue tracker
- Documentation: `/docs` directory
- Team Chat: Internal communication channel

Service Dependencies:
- **Vercel Support**: Vercel Dashboard
- **Supabase Support**: Supabase Dashboard
- **Google Maps Support**: Google Cloud Console

Security Best Practices

Environment Security

Secrets Management:
1. Never commit secrets to version control
2. Use different API keys for each environment
3. Implement key rotation schedule
4. Enable IP restrictions where possible
5. Monitor API usage for anomalies

Access Control:
1. Use principle of least privilege
2. Enable 2FA for all service accounts
3. Regular access reviews and cleanup
4. Separate service accounts per environment
5. Log and monitor all administrative actions

Application Security

Authentication & Authorization:

```
// Secure API route example
export async function GET(request: Request) {
  const token =
request.headers.get('Authorization')?.replace('Bearer ', '');

  if (!token) {
    return new Response('Unauthorized', { status: 401 });
  }

  try {
    const { data: user } = await supabase.auth.getUser(token);
    if (!user) {
      return new Response('Invalid token', { status: 401 });
    }

    // Process authenticated request
    return Response.json({ data: 'secure data' });
  } catch (error) {
    return new Response('Authentication failed', { status: 401 });
  }
}
```

Data Validation:

```
// Input validation with Zod
import { z } from 'zod';

const userSchema = z.object({
  name: z.string().min(1).max(100),
  email: z.string().email(),
  age: z.number().min(0).max(120)
});

export async function POST(request: Request) {
  try {
    const data = await request.json();
    const validatedData = userSchema.parse(data);

    // Process validated data
    return Response.json({ success: true });
  } catch (error) {
    return new Response('Invalid input', { status: 400 });
  }
}
```

Security Headers:

```
// Security middleware
export function middleware(request: NextRequest) {
  const response = NextResponse.next();

  response.headers.set('X-Content-Type-Options', 'nosniff');
  response.headers.set('X-Frame-Options', 'DENY');
  response.headers.set('X-XSS-Protection', '1; mode=block');
  response.headers.set('Referrer-Policy', 'strict-origin-when-
cross-origin');

  return response;
}
```

Infrastructure Security

Network Security:
- Enable HTTPS everywhere (force SSL)
- Implement proper CORS policies
- Use secure headers and CSP
- Regular security scanning and updates

Database Security:
- Enable Row Level Security (RLS) in Supabase
- Use parameterized queries (prevent SQL injection)
- Regular database security audits
- Encrypted connections and data at rest

Monitoring & Incident Response:
- Set up security event logging
- Implement anomaly detection
- Create incident response procedures
- Regular security drills and updates

Document Version: 1.0
Last Updated: 2025-08-26
Next Review: 2025-11-26

For questions or updates to this documentation, please create an issue in the repository or contact the development team.