Deployment Guide

Version: 0.1.0
Generated: 7/22/2025

Deployment Overview

This document provides detailed instructions for deploying the Blog Web Application to various environments. The application is designed to be deployed as a static site that connects to backend API services.

Prerequisites

Before deploying the application, ensure you have:

- Node.js v16.14.0 or higher installed on your build environment
- npm v8.3.0 or higher or yarn v1.22.0 or higher
- Access to the target deployment environment (hosting service credentials)
- Backend API services properly configured and accessible
- Environment variables for the target environment
- SSL certificate for production deployments (required for PWA features)
- Domain name configured with DNS settings pointing to your hosting provider

Environment Configuration

The application requires specific environment variables for each deployment environment:

Production Environment Variables

```
# API Configuration
NEXT_PUBLIC_API_URL=https://api.blogapp.com/v1
NEXT_PUBLIC_SOCKET_URL=https://api.blogapp.com
NEXT_PUBLIC_ASSET_URL=https://cdn.blogapp.com

# Analytics and Monitoring
NEXT_PUBLIC_GOOGLE_ANALYTICS_ID=UA-XXXXXXXXX-X
NEXT_PUBLIC_SENTRY_DSN=https://xxxxxxx.ingest.sentry.io/xxxxxxx

# Authentication
NEXT_PUBLIC_GOOGLE_CLIENT_ID=production-google-client-
id.apps.googleusercontent.com
NEXT_PUBLIC_APPLE_CLIENT_ID=com.blogapp.production

# Feature Flags
NEXT_PUBLIC_ENABLE_EXPERIMENTAL_FEATURES=false
NEXT_PUBLIC_ENABLE_MOCK_API=false
NEXT_PUBLIC_MAINTENANCE_MODE=false
```

Staging Environment Variables

```
# API Configuration
NEXT_PUBLIC_API_URL=https://api-staging.blogapp.com/v1
NEXT_PUBLIC_SOCKET_URL=https://api-staging.blogapp.com
NEXT_PUBLIC_ASSET_URL=https://cdn-staging.blogapp.com

# Analytics and Monitoring
NEXT_PUBLIC_GOOGLE_ANALYTICS_ID=UA-XXXXXXXX-X
NEXT_PUBLIC_SENTRY_DSN=https://xxxxxxx.ingest.sentry.io/staging-xxxxxxx

# Authentication
NEXT_PUBLIC_GOOGLE_CLIENT_ID=staging-google-client-id.apps.googleusercontent.com
NEXT_PUBLIC_APPLE_CLIENT_ID=com.blogapp.staging
```

```
# Feature Flags
NEXT_PUBLIC_ENABLE_EXPERIMENTAL_FEATURES=true
NEXT_PUBLIC_ENABLE_MOCK_API=false
NEXT_PUBLIC_MAINTENANCE_MODE=false
```

Building for Production

To create an optimized production build, follow these steps:

- Ensure all environment variables are properly configured in your .env.production file
- Run linting and tests to ensure code quality: npm run lint && npm test
- Generate the production build: npm run build
- Verify the build by running it locally: npm run start
- The optimized production files will be created in the "build" directory

```
# Example build script for CI/CD pipeline
npm ci --production
npm run lint
npm test
npm run build
```

The production build includes:

- Minified JavaScript bundles with code splitting
- Optimized CSS with vendor prefixes
- Compressed static assets
- Service worker for offline functionality
- Web App Manifest for PWA capabilities
- Static HTML for initial rendering
- Source maps for error tracking (optional, can be disabled)

Deployment Options

Deploying to Netlify

Netlify provides an easy way to deploy the application with continuous integration:

- Create a netlify.toml file in the project root:

```
[build]
  command = "npm run build"
  publish = "build"

[context.production]
  environment = { NODE_VERSION = "16.14.0" }

[[redirects]]
  from = "/*"
  to = "/index.html"
  status = 200
```

- Connect your GitHub repository to Netlify:
- 1. Log in to Netlify and click "New site from Git"
- 2. Select GitHub and authorize Netlify
- 3. Select your repository
- 4. Configure build settings using the netlify.toml file
- 5. Configure environment variables in the Netlify UI (Settings > Build & deploy > Environment)
- 6. Click "Deploy site"
- Set up a custom domain:
- 1. Go to Domain settings in Netlify
- 2. Click "Add custom domain"

- 3. Enter your domain and follow instructions to configure DNS

Deploying to Vercel

Vercel is optimized for Next.js applications and offers similar ease of use:

- Create a vercel.json file in the project root:

```
{
  "version": 2,
  "builds": [
    { "src": "package.json", "use": "@vercel/static-build" }
  ],
  "routes": [
    { "src": "/static/(.*)", "dest": "/static/$1" },
    { "src": "/favicon.ico", "dest": "/favicon.ico" },
    { "src": "/asset-manifest.json", "dest": "/asset-manifest.json" },
    { "src": "/manifest.json", "dest": "/manifest.json" },
    { "src": "/service-worker.js", "headers": { "cache-control": "s-maxage=0" },
"dest": "/service-worker.js" },
    { "src": "/(.*)", "dest": "/index.html" }
  ]
}
```

- Deploy to Vercel:
- 1. Install Vercel CLI: npm i -g vercel
- 2. Login to Vercel: vercel login
- 3. Deploy: vercel --prod
- Alternatively, connect your GitHub repository to Vercel:
- 1. Log in to Vercel and click "New Project"
- 2. Select your repository
- 3. Configure build settings and environment variables
- 4. Click "Deploy"

Deploying to AWS S3 + CloudFront

For more control and scalability, deploy to AWS using S3 for storage and CloudFront for content delivery:

- Create an S3 bucket:
- 1. Go to AWS S3 console and create a new bucket
- 2. Enable "Static website hosting" in bucket properties
- 3. Set the index document to "index.html" and error document to "index.html"
- 4. Update the bucket policy to allow public read access:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::your-bucket-name/*"
    }
  ]
}
```

- Deploy the build to S3:
- 1. Build the project: npm run build
- 2. Deploy to S3 using AWS CLI:

```
aws s3 sync build/ s3://your-bucket-name --delete
```

- Set up CloudFront:

- 1. Create a new CloudFront distribution
- 2. Set the origin domain to your S3 website endpoint
- 3. Configure cache behavior:
- • Default TTL: 86400 (1 day)
- • Compress objects automatically: Yes
- • Forward query strings: Yes
- 4. Set the default root object to "index.html"
- 5. Configure error pages:
- • HTTP Error Code: 403, 404
- • Response Page Path: /index.html
- • HTTP Response Code: 200
- 6. Create and attach SSL certificate using AWS Certificate Manager
- 7. Set up custom domain in CloudFront settings

Deploying with Docker

For containerized deployments, use Docker with Nginx as a web server:

- Create a Dockerfile in the project root:

```
# Build stage
FROM node:16-alpine as build
WORKDIR /app
COPY package*.json ./
RUN npm ci
COPY . .
RUN npm run build

# Production stage
FROM nginx:alpine
COPY --from=build /app/build /usr/share/nginx/html
COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Create an nginx.conf file in an nginx directory:

```
server {
  listen 80;
  server_name _;
  root /usr/share/nginx/html;
  index index.html;

  # Enable gzip
  gzip on;
  gzip_types text/plain text/css application/json application/javascript text/xml
application/xml application/xml+rss text/javascript;

  location / {
    try_files $uri $uri/ /index.html;
  }

  # Cache static assets
  location /static/ {
    expires 1y;
    add_header Cache-Control "public, max-age=31536000";
  }
}
```

- Build and run the Docker container:
- 1. Build the Docker image: docker build -t blogapp .
- 2. Run the container: docker run -p 8080:80 blogapp
- 3. Access the application at http://localhost:8080
- For production deployment:
- 1. Push the image to a container registry (Docker Hub, AWS ECR, etc.)

- 2. Deploy to your container orchestration platform (Kubernetes, ECS, etc.)

Continuous Integration/Continuous Deployment

Set up CI/CD pipelines to automate testing and deployment:

GitHub Actions

Create a workflow file at .github/workflows/deploy.yml:

```
name: Deploy

on:
  push:
    branches: [ main ]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 16
          cache: 'npm'
      - name: Install dependencies
        run: npm ci
      - name: Run linter
        run: npm run lint
      - name: Run tests
        run: npm test

  deploy:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 16
          cache: 'npm'
      - name: Install dependencies
        run: npm ci
      - name: Build
        run: npm run build
        env:
          NEXT_PUBLIC_API_URL: ${{ secrets.NEXT_PUBLIC_API_URL }}
          # Add other environment variables
      - name: Deploy to AWS S3
        uses: jakejarvis/s3-sync-action@master
        with:
          args: --delete
        env:
          AWS_S3_BUCKET: ${{ secrets.AWS_S3_BUCKET }}
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
          SOURCE_DIR: "build"
      - name: Invalidate CloudFront cache
        uses: chetan/invalidate-cloudfront-action@master
        env:
          DISTRIBUTION: ${{ secrets.CLOUDFRONT_DISTRIBUTION_ID }}
          PATHS: "/*"
          AWS_REGION: "us-east-1"
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

Post-Deployment Verification

After deploying, perform these checks to ensure everything is working correctly:

- Verify all pages load correctly and are responsive on different devices
- Check that API endpoints are connected and returning data
- Test authentication flows (login, signup, password reset)
- Ensure static assets (images, fonts, etc.) are loading properly
- Verify service worker registration and offline functionality
- Run Lighthouse audit to check performance, accessibility, SEO, and PWA compliance
- Check console for any errors or warnings
- Monitor error tracking service (Sentry) for any unexpected errors
- Test critical user flows (posting, commenting, navigation)

Rollback Procedures

If issues are detected after deployment, follow these rollback procedures:

Netlify/Vercel Rollback

- **Go to the Deployments section in your hosting dashboard**
- **Find the last known good deployment**
- **Click "Restore deployment" or equivalent option**
- **Verify the rollback was successful**

AWS S3/CloudFront Rollback

- **Deploy the previous version from CI/CD or manually:**
- **aws s3 sync s3://your-backup-bucket/previous-version/ s3://your-production-bucket/ --delete**
- **Invalidate CloudFront cache:**
- **aws cloudfront create-invalidation --distribution-id YOUR_DISTRIBUTION_ID --paths "/*"**