17. GitHub Pages

정적 사이트 배포

정적 사이트 배포 개요

정적 사이트란?

- 서버에서 별도의 동적 처리 없이 그대로 제공되는 파일들로 구성된 웹사이트
- 구성 요소:
  - HTML
  - CSS
  - JavaScript
  - 이미지, 폰트, 기타 리소스

→ 서버는 요청이 오면 그대로 파일만 응답

장점

✅ 빠른 로딩 속도
✅ 서버 리소스 최소 사용
✅ 비용 저렴 (서버리스 가능)
✅ 보안성이 높음 (서버 사이드 코드 없음)
✅ CDN에 쉽게 올릴 수 있음 → 글로벌 배포 용이

배포 방법

주요 배포 플랫폼

플랫폼	특징
GitHub Pages	GitHub Repo 기반 무료 호스팅, CI/CD 지원
Netlify	무료 플랜으로도 매우 강력한 배포 지원, PR Preview 가능
Vercel	Next.js에 최적화, 정적 사이트 배포 강력
AWS S3 + CloudFront	고성능, 대규모 서비스에 적합
Cloudflare Pages	빠른 CDN 기반 정적 사이트 제공

GitHub Pages 기본 배포 방법

1️⃣ 기본 흐름

- GitHub repo → `gh-pages` 브랜치 → GitHub Pages가 자동으로 해당 브랜치의 정적 파일을 서비스

2️⃣ 설정 방법

1. GitHub repo Settings → Pages 메뉴 → Source 지정 (`gh-pages` 브랜치 or `/docs` 디렉토리 등)
2. 정적 파일 빌드 후 → `gh-pages` 브랜치에 push

3️⃣ 예시 (React 앱)

```
npm run build
# build 결과물이 ./build 에 생성됨

# gh-pages 브랜치에 deploy
npm install --save-dev gh-pages

# package.json 에 추가
"scripts": {
  "predeploy": "npm run build",
  "deploy": "gh-pages -d build"
}

# deploy 실행
npm run deploy
```

→ 자동으로 gh-pages 브랜치에 빌드 결과 push → GitHub Pages에서 서비스됨

Netlify 기본 배포 방법

1. Netlify 사이트에서 GitHub repo 연동
2. Build Command (`npm run build` 등), Publish Directory (`build/` 등) 지정
3. Push 할 때마다 Netlify가 자동으로 빌드 + 배포
4. PR Preview 기능도 제공 → PR 별로 미리보기 URL 생성됨

Vercel 기본 배포 방법

1. Vercel 사이트에서 GitHub repo 연동
2. 기본 빌드 Command / Output Directory 자동 감지
3. 자동 CI/CD 파이프라인 구성됨 → Push 할 때마다 자동 배포

배포 Best Practice

✅ CI/CD 파이프라인 구축 → 빌드/테스트 자동화 후 배포
✅ CDN 사용 → 글로벌 배포 및 빠른 응답
✅ Cache 전략 수립 → 정적 리소스 캐싱 / versioning 처리
✅ SEO 최적화 → 정적 사이트라도 메타 태그, sitemap, robots.txt 구성
✅ Custom Domain 연결 → 독자 도메인 서비스 (GitHub Pages, Netlify, Vercel 등 모두 지원)

정리

항목	내용
대상 사이트	정적 HTML/CSS/JS 기반 웹사이트
주요 플랫폼	GitHub Pages, Netlify, Vercel, AWS S3+CloudFront, Cloudflare Pages
주요 장점	빠름, 비용 저렴, 보안성 우수, 글로벌 배포 용이
추천 구성	CI/CD 자동화 + CDN 최적화 + Custom Domain 적용

결론

정적 사이트 배포는:

- 빠르고 저렴하게 글로벌 서비스 가능
- GitHub Actions 등으로 **자동화된 배포 파이프라인** 구축하면 더 강력
- 단순한 개인 사이트부터 대규모 정적 서비스까지 폭넓게 활용 가능

커스텀 도메인 연결

기본 흐름

1️⃣ 도메인 구매
2️⃣ 정적 사이트 배포 플랫폼에 **커스텀 도메인 등록**
3️⃣ 도메인 DNS 설정
4️⃣ HTTPS 적용
5️⃣ 정상 연결 확인

플랫폼별 설정

GitHub Pages

- GitHub Repo → Settings → Pages → Custom domain 입력
- DNS 설정:

```
www.example.com → CNAME username.github.io
```

or Apex 도메인:

```
example.com → A 레코드 (GitHub Pages IP 사용)
```

- GitHub Pages가 HTTPS 자동 적용

Netlify

- Netlify → Site settings → Domain management → Add domain
- DNS 설정:

```
www.example.com → CNAME sitename.netlify.app
```

- Netlify가 Let's Encrypt 통해 HTTPS 자동 적용

Vercel

- Vercel → Project Settings → Domains → Add domain
- DNS 설정:

```
www.example.com → CNAME cname.vercel-dns.com
```

- Vercel이 HTTPS 자동 적용

도메인 구매 팁

✅ DNS 관리가 쉬운 업체 추천 (Cloudflare, GoDaddy 등)
✅ 네임서버 변경 없이 레코드만 수정 가능
✅ DNS TTL → 테스트 시 5~10분, 안정화 후 1시간 이상 설정

최종 확인

- DNS 상태 확인 → dnschecker.org
- `curl -I https://www.example.com` 로 HTTPS 정상 확인
- 브라우저 접속 확인
- SEO 검사 (robots.txt, sitemap.xml 등 준비 권장)

정리

플랫폼	커스텀 도메인 연결	HTTPS 지원
GitHub Pages	CNAME or A 레코드	자동

플랫폼	커스텀 도메인 연결	HTTPS 지원
Netlify	CNAME or A 레코드	자동
Vercel	CNAME or A 레코드	자동

결론

- 커스텀 도메인 연결은 **서비스 신뢰성 & 브랜드 이미지 필수 요소**
- GitHub Pages / Netlify / Vercel 모두 매우 쉽게 적용 가능
- HTTPS도 대부분 **자동 제공** → **추가 비용 없음**
- 연결 후 반드시 DNS 확인 + HTTPS + SEO 최적화까지 마무리하면 됨

Jekyll로 블로그 만들기

개요

- **Jekyll**: Markdown, Liquid 템플릿, HTML, CSS 등을 사용해 **정적 웹사이트(블로그)** 를 자동으로 빌드하는 도구
- GitHub Pages에서 **Jekyll 기본 지원** → 별도 빌드 서버 없이도 배포 가능
- 주로 **개인 블로그, 기술 블로그, 프로젝트 문서 사이트** 등에 활용

기본 흐름

1️⃣ Jekyll 설치 → 프로젝트 초기화
2️⃣ Markdown으로 글 작성
3️⃣ 로컬에서 확인 → GitHub Pages에 배포
4️⃣ 커스텀 테마 적용 / SEO 최적화

설치 및 초기화

1️⃣ Jekyll 설치 (로컬 개발용)

macOS / Linux

```
gem install --user-install bundler jekyll
```

Windows

- Ruby 설치 필요 → RubyInstaller 사용
- 그 후:

```
gem install bundler jekyll
```

2️⃣ Jekyll 사이트 생성

```
jekyll new my-blog
cd my-blog
```

3️⃣ 로컬 서버 실행

```
bundle exec jekyll serve
```

→ http://localhost:4000 에서 확인 가능

블로그 구조

```
my-blog/
├── _config.yml          → 설정 파일
├── _posts/              → 블로그 글 (Markdown 파일)
├── _layouts/            → 레이아웃 템플릿
├── _includes/           → 공통 템플릿
├── _site/               → 최종 빌드 결과물 (정적 사이트)
├── assets/              → CSS, JS, 이미지 등
├── index.md             → 홈 페이지
```

_posts 작성 예시

파일명: `YYYY-MM-DD-title.md`

```
---
layout: post
title: "Hello World"
date: 2024-06-08
categories: blog
---

Welcome to my first post!
```

GitHub Pages 배포

1️⃣ GitHub Pages 지원

- GitHub Pages는 Jekyll 기본 지원
- 별도 빌드 서버 없이 GitHub가 자동 빌드해줌

2️⃣ 배포 방법

gh-pages 브랜치 사용

or

main 브랜치의 `/docs` 폴더 사용 → 설정에서 지정 가능

3️⃣ 설정 예시 (GitHub Pages에서):

- Settings → Pages → Source → `gh-pages` or `/docs` 선택

테마 적용

- `_config.yml` 에서 테마 지정 가능

```
theme: minima
```

- Jekyll Themes 에서 다양한 무료/유료 테마 사용 가능

테마 적용 시 주의

- GitHub Pages에서 기본 whitelist된 테마만 빌드 지원 (외부 테마는 빌드해서 `_site` 결과물만 푸시하거나 GitHub Actions 사용 가능)

확장 기능

- **플러그인 사용** 가능 (단 GitHub Pages는 제한 있음 → Actions 사용 시 해결 가능)
- SEO 플러그인
- Sitemap 생성
- RSS Feed 생성

정리

항목	내용
핵심 도구	Jekyll
지원 플랫폼	GitHub Pages에서 기본 지원
글 작성	Markdown 기반 (.md 파일)
사이트 구성	정적 HTML/CSS로 빌드됨
주요 장점	속도 빠름, 무료 호스팅, Git 기반 버전 관리 가능
주요 사용 사례	개인 블로그, 기술 블로그, 프로젝트 문서 사이트 등

결론

- Jekyll은 **GitHub Pages에서 가장 쉽게 정적 블로그 만들 수 있는 도구**
- GitHub Repo에만 푸시하면 GitHub가 자동으로 배포해주기 때문에 비용 없이 운영 가능
- 테마와 플러그인 적용으로 충분히 **디자인 커스터마이징** 가능

React 앱 배포 (gh-pages 브랜치)

기본 흐름

1️⃣ React 앱 build → 정적 파일 생성 (`build/` 디렉토리)
2️⃣ `gh-pages` 브랜치에 build 결과물을 푸시
3️⃣ GitHub Pages → `gh-pages` 브랜치 기준으로 사이트 서비스

단계별 구성

1️⃣ 사전 준비

- GitHub에 Repo 생성 → React 프로젝트 push
- GitHub Pages → **Settings** → **Pages** → **Source** → **gh-pages 브랜치** 설정 가능

2️⃣ gh-pages 패키지 설치

```
npm install --save-dev gh-pages
```

→ 이 패키지가 `gh-pages` 브랜치로 자동으로 deploy 해주는 역할

3️⃣ package.json 수정

homepage 설정

```
"homepage": "https://username.github.io/repo-name",
```

→ 자신의 GitHub username / repo name으로 정확하게 입력

scripts 추가

```
"scripts": {
  "predeploy": "npm run build",
  "deploy": "gh-pages -d build"
}
```

→ `npm run deploy` 시 자동으로:

- `npm run build` → build/ 디렉토리 생성
- `gh-pages -d build` → build 결과물을 gh-pages 브랜치에 push

4️⃣ 배포

```
npm run deploy
```

→ GitHub Pages에 사이트 자동 배포

→ GitHub Pages Settings 에서 배포 URL 확인 가능:

```
https://username.github.io/repo-name
```

주의사항

✅ React Router 사용 시 → **BrowserRouter 사용 시 404 발생 가능** → HashRouter로 전환하거나 GitHub Pages 404 fallback 설정 필요
✅ build 결과물은 자동으로 gh-pages 브랜치로 관리됨 → source 코드는 main/master 브랜치 유지
✅ 다시 deploy 시 → `npm run deploy` 만 실행하면 됨

정리

단계	내용
1	`gh-pages` 패키지 설치
2	`package.json` 에 homepage 설정 + deploy script 추가
3	`npm run deploy` 실행
4	GitHub Pages 설정에서 gh-pages 브랜치 선택 (자동 인식)
5	배포 URL 확인 (https://username.github.io/repo-name)

결론

- React 앱을 GitHub Pages에 배포하면 **서버 없이 정적 사이트로 React 앱 서비스 가능**
- GitHub Actions 없이 간단하게 `npm run deploy` 만으로 빠른 배포 가능
- React 앱에 간단한 설정(homepage + deploy script)만 추가하면 쉽게 적용 가능