

Conversación con Gemini

Exported: 15/6/2026 8:57:08

Jordi H.

Has dicho hay alguna manera de generar la documentacion desde el repo de github a cosas como readthedocs

Gemini

¡Por supuesto! De hecho, **Read the Docs (RTD)** es el estándar de la industria para hospedar documentación de código abierto. Aunque históricamente nació muy ligado a Sphinx (el generador de Python), hoy en día soporta perfectamente **MkDocs**, que es el "primo hermano" de mdBook y quizá la mejor opción para integrarse con Read the Docs de forma nativa. Aquí tienes el mapa de ruta para conectar tu repositorio

1. La opción nativa: MkDocs en Read the Docs

Si te gusta la sencillez de mdBook pero quieres la infraestructura de RTD, **MkDocs** es el camino más corto. RTD tiene soporte oficial de primer nivel para él

- Configuración: Solo necesitas un archivo mkdocs.yml en la raíz.
- Anidación: Soporta perfectamente estructuras de carpetas. En el archivo de configuración defines la navegación:YAM

CODE

```
nav:  
- Home: index.md  
- Arquitectura:  
- Servidor: arquitectura/server.md  
- Redes: arquitectura/network.md
```

- PDF: Read the Docs genera automáticamente una versión PDF (y ePub) de tu documentación cada vez que actualizas el repo, sin que tengas que configurar herramientas de LaTeX en local

2. ¿Se puede usar mdBook en Read the Docs?

Directamente no es "plug-and-play" como MkDocs, pero se puede mediante un **workflow personalizado**. RTD permite ahora usar un archivo de configuración llamado .readthedocs.yaml donde puedes definir comandos personalizados de instalación. Podrías indicarle que descargue el binario de mdBook, compile y sirva la carpeta /book. Es un enfoque más "arquitectónico" y manual

3. El flujo de trabajo (CI/CD)

El proceso para integrar tu repo de GitHub con RTD es el siguiente:

1. Conexión: Te logueas en Read the Docs con tu cuenta de GitHub.
2. Importación: Seleccionas el repositorio de tu servidor/laboratorio.
3. Webhook: RTD instalará automáticamente un webhook en tu GitHub. Cada vez que hagas git push, RTD detecta el cambio, levanta una máquina virtual (un contenedor), compila tus Markdown y publica la web
4. Versiones: Puedes tener documentación para la rama main y versiones específicas para diferentes "releases" de tu laboratorio

4. Alternativa: GitHub Pages + GitHub Actions

Si prefieres no usar un servicio externo como RTD y mantenerlo todo dentro de GitHub:

- Usas una GitHub Action que ejecute mdBook.
- El PDF y el HTML se generan en el pipeline.
- El HTML se despliega en GitHub Pages.
- El PDF se adjunta como un "artifact" o se sube a una carpeta del mismo sitio.

Resumen de herramientas para tu perfil:

Característica	MkDocs + Read the Docs	mdBook + GitHub Actions
Facilidad PDF	Automática (nativa en RTD)	Requiere plugin y motor externo
Lenguaje	Python (muy afín a ti)	Rust (binario único)
Hosting	readthedocs.io	github.io
Estética	Muy profesional/limpia	Estilo "Libro técnico"

Mi recomendación: Dado que ya trabajas con **Python** y buscas algo que gestione bien el "árbol" de ficheros y genere el PDF sin complicaciones, **MkDocs con el tema Material** desplegado en Read the Docs es la combinación más potente y con menos fricción. ¿Te gustaría que viéramos cómo estructurar el archivo `.readthedocs.yaml` para automatizarlo to

Jordi H.

Has dicho vamos a abrir más las opciones. un generador de web estática desde el repositorio? en github hay algo ya preprado

Gemini

Si buscamos ampliar el horizonte hacia los **Generadores de Sitios Estáticos (SSG)** que se integren de forma nativa con GitHub, entramos en el terreno donde la automatización es la norma. GitHub tiene una integración profunda con estas herramientas a través de **GitHub Pages**. Aquí tienes las opciones más sólidas y "preparadas" para lo que busc

1. Jekyll (El nativo de GitHub)

Es la opción por defecto. GitHub Pages nació con Jekyll integrado, lo que significa que puedes subir tus archivos Markdown y GitHub los compilará automáticamente sin que tú hagas nada

- Ventaja: No necesitas configurar Actions si usas los temas básicos. Es "subir y listo".
- Contras: Está basado en Ruby, lo cual puede resultarte ajeno si prefieres Python. Además, generar un PDF desde Jekyll no es nativo; requiere plugins adicionales o herramientas externas

2. Docusaurus (El estándar moderno)

Desarrollado por Meta (Facebook), es probablemente el SSG de documentación más potente actualmente. Está basado en **React**

- Estructura de árbol: Es excelente gestionando anidaciones profundas, barras laterales dinámicas y versiones de documentos
- Búsqueda: Incluye integración con Algolia para tener un buscador profesional en tu doc.
- Despliegue: GitHub tiene una "Action" oficial para Docusaurus que despliega a GitHub Pages en segundos.

3. MkDocs + Material Theme (La recomendación para tu perfil)

Siendo tú usuario de **Python**, esta es la opción ganadora. **Material for MkDocs** es, posiblemente, el tema de documentación más bonito y funcional del ecosistema Open Source

- Plugin de PDF: Existe el plugin `mkdocs-pdf-export-plugin` o `mkdocs-with-pdf`.
- Directorio de ilustraciones: Gestiona las imágenes de forma impecable y permite usar Mermaid.js para crear diagramas (flujos de datos, arquitectura) directamente con texto Markdown
- GitHub Actions: Hay una acción específica (`mkdocs-gh-deploy`) que hace todo el trabajo por ti.

¿Cómo se configura en GitHub?

GitHub tiene una funcionalidad llamada **GitHub Actions**. En lugar de compilar tú el sitio en local, dejas un pequeño archivo de configuración (YAML) en la carpeta `.github/workflows/`. Cuando haces un git push

1. Una máquina virtual de GitHub se enciende.
2. Instala el generador (MkDocs, mdBook o Jekyll).
3. Compila tus Markdown.
4. Genera el PDF.
5. Publica la web en tu URL de github.io y deja el PDF disponible para descarga.

Comparativa de flujo de trabajo

Herramienta	Dificultad	Lenguaje Base	Ideal para...
Jekyll	Baja	Ruby	Proyectos simples y
MkDocs	Media	Python	Documentación técnica,
Docusaurus	Media-Alta	JavaScript/React	Sitios web de
mdBook	Baja	Rust	Manuales técnicos tipo

Mi sugerencia técnica

Dado que estás montando un **homelab** y un **vector tecnológico**, **MkDocs con el tema Material** te dará el equilibrio perfecto

1. Es software libre.
2. Usa Python (tu lenguaje principal).
3. Permite insertar diagramas de arquitectura como código.
4. La comunidad es enorme y hay cientos de extensiones.

¿Te gustaría que te pase un ejemplo de cómo sería el archivo de flujo de trabajo para que GitHub haga todo este proceso de "Markdown a Web + PDF" automáticamente

Jordi H.

Has dicho puedo generar la docu a docusaurus sin prácticamente interacción?

Gemini

Sí, es totalmente posible. De hecho, el ecosistema de **Docusaurus** está diseñado para que, una vez configurada la infraestructura inicial, tu única interacción sea hacer un git push de tus archivos Markdown. Para lograr esa "interacción cero", el secreto está en delegar todo el trabajo pesado a **GitHub Actions**. Aquí te detallo cómo automatizarlo para que el sistema se encargue de la estructura de árbol, las imágenes y el despliegue

1. El "Truco" de la Estructura Automática

Docusaurus suele requerir que definas un archivo `sidebars.js`. Sin embargo, para evitar interactuar con él, puedes configurar el **autogenerador de barras laterales**. En tu archivo `sidebars.js`, simplemente pon esto:

CODE

```
module.exports = {
  mySidebar: [
    {
      type: 'autogenerated',
      dirName: '.', // Genera el árbol basado en tu estructura de carpetas
    },
  ],
};
```

Con esto, si creas una carpeta `/redes` con tres `.md` dentro, Docusaurus creará automáticamente la sección y el índice en la web sin que tú escribas una sola línea de código adicional.

2. Automatización Total con GitHub Actions

Para no tener que instalar nada en tu servidor o PC, creas un archivo en `.github/workflows/deploy.yml`. Este script hará lo siguiente cada vez que detecte cambios:

1. Levanta un contenedor con Node.js.
2. Instala Docusaurus.
3. Busca tus imágenes en el directorio de ilustraciones que mencionaste.
4. Compila el sitio estático.
5. Lo publica en GitHub Pages.

Ejemplo de Workflow (YAML):

YAML

CODE

```
name: Deploy Docusaurus
on:
  push:
    branches: [main]
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: 18
      - name: Install dependencies
        run: npm install
      - name: Build website
        run: npm run build
      - name: Deploy to GitHub Pages
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./build
```

3. Manejo de Ilustraciones

Como mencionaste que tienes un directorio de ilustraciones, Docusaurus las gestiona de forma nativa. Si tus imágenes están en `/static/img/`, puedes referenciarlas en tus Markdown así
![Arquitectura del Lab](/img/esquema.png)Docusaurus se encarga de optimizarlas y servir las correctamente en la web final

4. ¿Y el PDF único?

Aquí es donde Docusaurus requiere un pequeño paso extra. Como es un generador orientado a web (Single Page Application), no genera PDF nativo al vuelo. Tienes dos opciones para mantener la "baja interacción"

- Plugin `docusaurus-plugin-content-docs`: Permite añadir un botón de "Descargar PDF" que utiliza servicios como PrinceXML o DocRaptor
- Action de terceros: Puedes añadir un paso final en tu GitHub Action que use Wandoc o un script de Puppeteer para "escanear" tu sitio ya publicado y convertirlo en un PDF indexado de forma automática

Resumen: ¿Cómo queda tu flujo de trabajo?

1. Escribes tus notas sobre el homelab en un `.md` dentro de tu repo.
2. Haces git commit y git push.
3. Fin.
 - GitHub Actions compila la web.
 - La web se actualiza en tu URL de GitHub.
 - El árbol de navegación se ajusta solo si añadiste carpetas.

Es lo más parecido a "magia" documental que puedes conseguir manteniendo el control total del código.
¿Quieres que te ayude a preparar la estructura inicial de carpetas para que Docusaurus la reconozca a la primera?

Jordi H.

Has dicho si

Gemini

Para poner en marcha **Docusaurus** con el mínimo esfuerzo y que se gestione solo, la estructura de tu repositorio debería verse así. Esto permitirá que el autogenerador de barras laterales (el "árbol") y el directorio de ilustraciones funcionen sin que tengas que tocar código cada vez que añades contenido

1. Estructura recomendada del repositorio

Docusaurus espera una disposición específica para encontrar las cosas automáticamente: Plaintext

CODE

```
mi-repo-documentacion/  
docs/ <-- Aquí van tus .md (puedes anidar carpetas aquí)  
intro.md  
infraestructura/  
servidor.md  
redes.md  
laboratorio/  
herramientas.md  
static/ <-- Directorio de ilustraciones  
img/  
esquema-red.png  
docusaurus.config.js <-- Configuración principal  
sidebars.js <-- Aquí activamos el "auto-tree"
```

```
package.json
.github/workflows/ <-- El motor de automatización
```

2. Configuración del "Árbol" automático

En tu archivo `sidebars.js`, olvida listar archivo por archivo. Usa esto para que Docusaurus escanee tu carpeta `docs/` y cree el índice por ti:JavaScript

CODE

```
module.exports = {
  tutorialSidebar: [{type: 'autogenerated', dirName: '.'}],
};
```

3. Configuración del Proyecto (`docusaurus.config.js`)

Para que las ilustraciones funcionen sin problemas, asegúrate de que el `baseUrl` y el `url` apunten a tu dirección de GitHub Pages:JavaScript

CODE

```
module.exports = {
  title: 'Mi Homelab',
  url: 'https://tu-usuario.github.io',
  baseUrl: '/tu-repo/',
  onBrokenLinks: 'throw',
  onBrokenMarkdownLinks: 'warn',
  organizationName: 'tu-usuario',
  projectName: 'tu-repo',
  // ... resto de la config
};
```

4. Generación del PDF (Interacción Cero)

Como Docusaurus no genera el PDF por defecto, lo más eficiente para tu perfil es añadir un paso extra en la **GitHub Action**. Podemos usar una herramienta llamada `pagedjs` o un plugin específico que convierta el sitio web ya generado en un único PDF. Si quieres el PDF "indexado y profesional" que mencionabas al principio, una opción muy limpia es usar un contenedor de **Playwright** en el workflow que "navegue" por tu documentación y la imprima a PDF después de compilar

Pasos para empezar hoy mismo:

1. Inicializa el repo: Si tienes Node.js, ejecuta `npx create-docusaurus@latest my-website classic`.
2. Mueve tus Markdown: Pon tus archivos actuales en la carpeta `docs/`.
3. Configura el Workflow: Crea el archivo `.github/workflows/deploy.yml` que te pasé antes.
4. Push: En cuanto hagas el push, GitHub detectará todo y en un par de minutos tendrás tu web de documentación operativa

¿Prefieres que te ayude a detallar el paso exacto del Workflow para generar el PDF de forma automática al

Jordi H.

Has dicho puedo generar acceso privado a la web resultante? con user/pass?

Al tratarse de una web estática servida por **GitHub Pages**, no existe una base de datos ni un servidor de aplicaciones que gestione sesiones de usuario de forma nativa. Sin embargo, existen tres caminos principales para lograr esa privacidad sin perder la automatización

1. El método "Nativo" (GitHub Pro / Enterprise)

Si tu repositorio es privado y tienes una cuenta de **GitHub Pro, Team o Enterprise**, GitHub Pages ofrece una opción llamada **"Access Control"**

- **Cómo funciona:** En la configuración del repositorio, puedes restringir la visualización de la web solo a personas que tengan acceso al repositorio
- **Pros:** Es oficial, seguro y gestionado por GitHub.
- **Contras:** Requiere suscripción de pago y que los usuarios tengan cuenta en GitHub.

2. Cloudflare Pages (La mejor opción gratuita y profesional)

Si buscas algo profesional con usuario y contraseña sin pagar licencias de GitHub, la solución ideal es conectar tu repo a **Cloudflare Pages**

- **Cómo funciona:** Cloudflare importa tu código de GitHub y lo sirve. Puedes activar Cloudflare Access (parte de su plan "Zero Trust", gratuito hasta 50 usuarios)
- **Privacidad:** Permite configurar un formulario de login antes de ver la web. Puedes usar contraseñas temporales por email (OTP) o integrar proveedores como Google, GitHub o incluso un simple usuario/contraseña
- **Sincronización:** Cada vez que hagas un push en GitHub, Cloudflare recompila el Docusaurus automáticamente

3. Stork / Static Crypt (Protección por contraseña en el cliente)

Si quieres mantenerlo en GitHub Pages gratuito y solo necesitas una barrera simple (sin gestión compleja de usuarios), puedes cifrar el contenido estático

- **Herramienta:** StaticCrypt.
- **Cómo funciona:** Se añade un paso al final de tu GitHub Action que cifra tus archivos HTML con una contraseña
- **Experiencia:** Cuando alguien entra en la web, ve una caja de texto que pide la contraseña. Al introducirla, el navegador descifra el contenido en local
- **Pros:** Muy fácil de implementar y 100% gratuito.
- **Contras:** Si alguien conoce la contraseña, puede compartirla. No es una seguridad de nivel bancario, pero es un "candado" muy efectivo para documentaciones personales o de laboratorio

Mi recomendación técnica

Como ya gestionas un **homelab** y te gusta el software libre, yo optaría por **Cloudflare Pages + Cloudflare**

1. **Seguridad Real:** No solo ocultas la web, sino que está protegida por una capa de red global.
2. **Facilidad:** El flujo de CI/CD es idéntico: tú solo haces push a GitHub y Cloudflare se encarga del resto.
3. **PDF Privado:** Puedes dejar el link de descarga del PDF dentro de la web protegida, asegurando que solo los autorizados bajen tu documentación completa

¿Te interesa que profundicemos en cómo configurar el **StaticCrypt** dentro de tu GitHub Action para que sea totalmente automático? Es la opción que requiere menos cambios de infraestructura

Has dicho puedo usar uno de los repositorios como "central" (de hecho, solamente contiene ficheros markdown e ilustraciones), y además incorporar las documentaciones de otros repositorios de código fuente ? muchos de mis repos tienen una sección docs con el informe técnico, de uso, u otras informaciones

Gemini

Es una arquitectura excelente y muy común en entornos profesionales: se conoce como **Documentación en Publicación Centralizada**. Dado que tu repositorio central será el "cerebro" (el que tiene la estructura de Docusaurus/MkDocs) y los demás repositorios contienen la información técnica específica, tienes dos formas principales de unificar todo sin tener que copiar y pegar archivos manualmente.

1. La opción nativa de Git: Git Submodules

Es la solución más pura desde el punto de vista de la arquitectura de sistemas. Consiste en "anclar" otros repositorios dentro de una carpeta de tu repositorio central

- **Cómo funciona:** En tu repo central, ejecutas `git submodule add [URL-repo-fuente] docs/proyecto-A`.
- **Automatización:** Cuando tu GitHub Action vaya a compilar la web, le indicas que primero "traiga" el contenido de los submódulos
- **Sincronización:** El repo central siempre apuntará a una versión (commit) específica del repo de código. Si actualizas el informe técnico en el repo de código, solo tienes que hacer un `git submodule update` en el central para que la web se actualice

2. El enfoque de "Multi-repo" con GitHub Actions (Recomendado)

Esta es la opción más flexible y la que mejor encaja con la idea de "interacción cero". En lugar de que el repo central "tire" de los demás, configuramos el flujo para que los datos "fluyan" hacia el centro. **El flujo sería este**

1. Tienes tu Repo Central con la estructura de Docusaurus.
2. En tus Repos de Código, configuras una GitHub Action que, cada vez que edites algo en la carpeta `/docs`, haga un "push" automático solo de esos archivos hacia una carpeta específica del Repo Central
3. El Repo Central, al recibir esos archivos nuevos, dispara su propia compilación y actualiza la web.

3. Usar Plugins específicos (MkDocs "Remote" / Docusaurus "Remote Content")

Existen herramientas diseñadas específicamente para este "puzzle":

- **MkDocs Remote Content:** Existe un plugin llamado `mkdocs-remote-content` que te permite definir en el archivo de configuración URLs de otros repositorios. Al compilar, MkDocs descarga los Markdown de esos sitios externos y los renderiza como si fueran locales
- **Docusaurus Remote Content:** Hay plugins similares (como `docusaurus-plugin-remote-content`) que descargan los archivos durante el proceso de build

Comparativa para tu escenario

Característica	Git Submodules	GitHub Actions (Push)	Plugins Remote Content
Control	Total (tú eliges qué versión)	Automático al 100%	Dinámico al compilar
Complejidad	Media (Git puede ser)	Alta al inicio (configurar)	Baja (solo configuración)
Uso de Red	Solo al clonar	Frecuente (cada push)	En cada compilación

Mi recomendación para tu Homelab

Si quieres que el mantenimiento sea mínimo, usa los **Plugins de Remote Content**.

Simplemente le dices a tu Docusaurus central: "Busca el fichero README.md de este otro repositorio e insértalo en la sección 'Servidores'". De esta forma, mantienes el código y su documentación juntos (como deb e ser), pero el usuario final ve una web única, coherente y perfectamente indexada. ¿Te gustaría ver cómo se configuraría una sección de "Remote Content" en el fichero de Docusaurus para traer un archivo de otro re

Jordi H.

Has dicho ya tengo funcionando una primera aproximacion. muy bien el docusaurus. ahora quiero que la url no sea de esas inventadas por github. que la pueda decidir yo o que tenga otra opcio

Gemini

¡Excelente! Me alegra que **Docusaurus** ya esté corriendo. Ahora, para personalizar la URL y profesionalizar el acceso, tienes dos caminos principales según si ya posees un dominio propio o si buscas una opción gratuita a con mejor estética. Aquí tienes las opciones para dejar atrás la URL por defecto (`usuario.github.io/repo/`)

1. Usar un Dominio Propio (Custom Domain)

Si tienes un dominio (ej. `mi-laboratorio.com` o `hernan.dev`), puedes vincularlo directamente a tu documentación en GitHub

- Configuración en GitHub: En tu repositorio, ve a Settings > Pages. En la sección "Custom domain", escribe tu dominio o subdominio (ej. `docs.mi-laboratorio.com`)
- Configuración DNS: Deberás ir al panel de tu registrador de dominios (donde lo compraste) y añadir: Un registro CNAME apuntando a `tu-usuario.github.io`. O registros A apuntando a las IPs oficiales de GitHub si usas el dominio raíz
- SSL/HTTPS: GitHub generará automáticamente el certificado SSL (Let's Encrypt) para tu dominio de forma gratuita

2. Cloudflare Pages (Recomendado por estética y privacidad)

Como mencionamos antes, si mueves el despliegue a **Cloudflare Pages** (que se conecta a tu repo de GitHub con un clic)

- URL más limpia: Te darán una URL del tipo `mi-docs.pages.dev`, que suele ser más corta y profesional que la de GitHub
- Dominio personalizado gratis: Cloudflare facilita enormemente la gestión del SSL y la seguridad.
- Ventaja extra: Desde aquí puedes activar el Acceso Privado (login con email o clave) de forma mucho más robusta que en GitHub Pages

3. Cambiar el "BaseURL" de GitHub Pages

Si no quieres comprar un dominio pero quieres que la URL sea más corta dentro de GitHub, puedes configurar el repositorio como tu "User Site"

- Si renombas tu repositorio a `tu-usuario.github.io`, la documentación pasará de estar en `tu-usuario.github.io/proyecto/` a estar directamente en la raíz
: `[https://tu-usuario.github.io](https://tu-usuario.github.io)`

Ajuste CRÍTICO en Docusaurus

Independientemente de la opción que elijas, **debes actualizar tu archivo `docusaurus.config.js`**, de lo contrario los estilos CSS y las imágenes se romperán al cambiar la URL: JavaScrip

CODE

```
// docusaurus.config.js
module.exports = {
  url: 'https://tu-nuevo-dominio.com', // O 'https://mi-docs.pages.dev'
  baseUrl: '/', // Si ahora está en la raíz del dominio, cámbialo de '/repo/' a '/'
  // ...
};
```

Resumen de pasos para el cambio:

1. Decide el nombre: ¿Comprarás un .com/.es o prefieres un subdominio gratuito de Cloudflare?
 2. Configura el Hosting: En los Settings de GitHub o el panel de Cloudflare.
 3. Actualiza el código: Cambia url y baseUrl en tu docusaurus.config.js y haz push.
- Si optas por un **dominio propio**, ¿necesitas que te ayude con los registros DNS específicos que debes crear?

Jordi H.

Has dicho estoy dentro de una organization. cada repo genera una url inventada con palabras aleatorias. a mi me ha tocado effective-adventure-xxxxxx.page

Gemini

Parece que GitHub ha activado por defecto el aislamiento de subdominios para tu organización, lo que genera esos nombres aleatorios por seguridad (para evitar ataques de tipo cross-cookie entre diferentes proyectos de la misma organización). Como estás dentro de una **GitHub Organization**, tienes tres niveles para solucionar esto y conseguir una URL que no parezca generada por un generador de contraseñas

1. El método "Limpio": Custom Domain (Recomendado)

Si tu organización ya tiene un dominio corporativo (ej. empresa.com), puedes crear un subdominio específico para tu documentación (ej. docs-laboratorio.empresa.com)

1. En GitHub: Ve a los Settings de la Organización > Pages. Allí puedes añadir el dominio raíz de la empresa y verificarlo
2. En el Repo: Ve a Settings > Pages del repositorio de Docusaurus y escribe el subdominio que quieras usar.
3. DNS: Pide al administrador de sistemas que apunte ese CNAME a tu-org.github.io.

2. Renombrar el Repositorio (Si usas el dominio de la Org)

Si no tienes un dominio propio, GitHub Pages usa por defecto la estructura:

`https://<org-name>.github.io/<repo-name>/`. Si tu URL actual es algo como

`effective-adventure-xxxxxx.pages.github.io`, es probable que la organización esté usando **GitHub Enterprise** con subdominios únicos por repositorio. Para cambiar el nombre "adventure", simplemente

1. Ve a Settings > General.
2. Cambia el Repository Name por algo descriptivo como doc-homelab.
3. La URL debería actualizarse automáticamente a algo tipo `doc-homelab-xxxxxx.pages.github.io`. Sigue teniendo el sufijo aleatorio por seguridad, pero al menos el cuerpo de la URL es legible

3. La opción "Pro": Cloudflare Pages

Dado que mencionaste que te gusta tener control y eres arquitecto de sistemas, **Cloudflare Pages** es la solución más elegante para saltarse las restricciones de nombres de las organizaciones de GitHub

1. Conectas Cloudflare a tu repositorio de la organización.
2. Cloudflare te permite elegir el nombre del proyecto. Si eliges mi-laboratorio, tu URL será https://mi-laboratorio.pages.dev
3. Es gratuito, mucho más rápido que GitHub Pages y elimina por completo las palabras aleatorias.

¡Ojo con el archivo de configuración!

En el momento en que cambies esa URL "inventada" por una propia o una de Cloudflare, **Docusaurus dejará de** si no actualizas el docusaurus.config.js. Asegúrate de cambiar estos valores para que coincidan con tu nueva identidad: JavaScript

CODE

```
// docusaurus.config.js
module.exports = {
  url: 'https://tu-nueva-url.com', // La nueva URL (sin el path)
  baseUrl: '/', // Si la nueva URL apunta directamente a la raíz de la doc
  // ...
};
```

¿Tienes permisos de administrador en la organización o prefieres una solución que puedas gestionar tú solo sin

Jordi H.

Has dicho explicame mas sobre cloudflare pages

Gemini

Cloudflare Pages es una plataforma de alojamiento Jamstack (sitios web estáticos y aplicaciones frontend) pensada para desarrolladores y arquitectos de sistemas. Funciona bajo un principio de **"interacción cero"** tras su configuración inicial, integrándose directamente con tu flujo de Git (GitHub o GitLab). A diferencia de los hostings tradicionales o de soluciones como GitHub Pages, la gran ventaja de Cloudflare es que tu documentación no se aloja en un servidor central, sino que se distribuye directamente en su infraestructura global de red (Edge Network) que cuenta con cientos de puntos de presencia (PoPs) en todo el mundo.

1. ¿Cómo funciona la arquitectura con Docusaurus?

El flujo de trabajo es extremadamente limpio y elimina la necesidad de mantener servidores web o configurar pipelines complejos de CI/CD

CODE

```
[ Tu PC / Servidor ] --( git push )--> [ GitHub Repository ]
```

(Webhook automático)

```
[ Cloudflare Pages ]
```

1. Levanta contenedor
2. Compila Docusaurus
3. Despliega en el Edge

```
[ Red Global / Users ]
```

1. El disparador: Escribes tu documentación en Markdown dentro de tu repositorio (ya sea personal o de organización) y haces un git push
2. La compilación (Build): Cloudflare detecta el cambio mediante un webhook, levanta un entorno aislado, instala las dependencias de Node.js y ejecuta el comando de producción (npm run build o yarn build de tu Docusaurus)
3. Distribución en el Edge: El resultado estático (HTML, CSS, JS e imágenes) se envía inmediatamente a toda la red global de Cloudflare. Tu web se sirve siempre desde el nodo más cercano al usuario que la solicita, lo que reduce la latencia a prácticamente cero

2. Ventajas clave frente a GitHub Pages (Especialmente en Organizaciones)

- URLs limpias y predecibles: Evitas los subdominios aleatorios de seguridad que te impone GitHub Enterprise. En Cloudflare tú eliges el nombre del proyecto; si lo llamas docs-laboratorio, tu URL gratuita será <https://docs-laboratorio.pages.de>
- Dominios personalizados sin fricción: Si decides asignarle un dominio propio (como docs.mi-homelab.com), Cloudflare gestiona la delegación de DNS, la mitigación de ataques DDoS y la renovación del certificado o SSL (TLS) de forma 100% automatizada
- Previsualizaciones por Rama (Preview Deployments): Si creas una rama en Git para probar una nueva estructura de documentación, Cloudflare compilará una URL única y temporal para esa rama (ej: <https://rama-prueba.docs-laboratorio.pages.dev>). Así puedes validar cómo queda el árbol antes de fusionarlo a la rama principal (main)

3. Seguridad y Control de Acceso (Tu Homelab Privado)

Dado que uno de tus objetivos es controlar quién accede, Cloudflare te permite aplicar una capa de seguridad profesional **de forma gratuita para hasta 50 usuarios** a través de su suite **Cloudflare Access (Zero Trust)**. Puede bloquear el acceso a tu subdominio .pages.dev o a tu dominio personalizado mediante varias políticas

- One-Time PIN (OTP): El usuario introduce su correo electrónico y Cloudflare le envía un código de un solo uso para poder entrar. No necesitas gestionar contraseñas
- Proveedores de Identidad (IdP): Puedes restringir el acceso para que solo los usuarios que se autenticuen con su cuenta de GitHub, Google o GitLab puedan ver la documentación
- Restricción por IP o País: Si solo quieres acceder desde la IP de tu homelab o desde redes específicas, puedes crear una regla de cortafuegos en tres clics

4. El archivo de configuración que necesitas para Docusaurus

Al delegar la compilación a Cloudflare, tu archivo docusaurus.config.js se vuelve muy sencillo porque ya no depende de rutas anidadas de repositorios. Siempre se sirve desde la raíz del dominio: JavaScript

CODE

```
// docusaurus.config.js
module.exports = {
  title: 'Documentación Central',
  tagline: 'Vector Tecnológico y Homelab',
  url: 'https://tu-proyecto.pages.dev', // O tu dominio personalizado posterior
  baseUrl: '/', // Al ser un dominio dedicado, la base siempre es la raíz
  onBrokenLinks: 'throw',
  onBrokenMarkdownLinks: 'warn',

  // Tu configuración de barras laterales automáticas y temas...
};
```

Para ponerlo en marcha, basta con ir al panel de Cloudflare, seleccionar **Workers & Pages**, darle a **Create a Connect to Git** y **seleccionar tu repositorio**. En la configuración del Build eliges el ajuste preestablecido de Docusaurus

aurus**, y la plataforma se encargará de rellenar los comandos necesarios por ti. ¿Te gustaría que viéramos cómo se estructura la política de acceso de Zero Trust en Cloudflare para proteger esa URL?

Jordi H.

Has dicho no tengo cuenta en cloudflare. como empiezo?

Gemini

Crear una cuenta y conectar tu Docusaurus es un proceso rápido. Cloudflare tiene un modelo freemium muy generoso, por lo que todo lo que vamos a hacer (alojamiento, SSL, subdominio limpio y la capa de seguridad de Access) te costará exactamente 0 euros. Sigue estos pasos para desplegar tu documentación desde cero.

Paso 1: Crear la cuenta en Cloudflare

1. Ve a dash.cloudflare.com/sign-up.
2. Introduce tu correo electrónico y una contraseña.
3. No necesitas introducir ninguna tarjeta de crédito ni seleccionar planes de pago. Al terminar el registro, entrarás directamente al panel principal (Dashboard)

Paso 2: Conectar tu repositorio de GitHub

Una vez dentro del panel, vamos a desplegar el Docusaurus:

1. En la barra lateral izquierda, haz clic en Workers & Pages.
2. Haz clic en el botón azul Create (Crear) y luego selecciona la pestaña Pages.
3. Haz clic en Connect to Git.
4. Selecciona GitHub. Te pedirá autenticarte con tu cuenta de GitHub y dar permisos. Puedes darle acceso a todos tus repositorios o únicamente al repositorio central de documentación (lo cual es más recomendable por seguridad).
5. Selecciona el repositorio de tu organización o cuenta personal y haz clic en Begin setup.

Paso 3: Configurar el Build de Docusaurus

Cloudflare detectará automáticamente la tecnología de tu repositorio, pero asegúrate de que los campos queden de la siguiente manera:

- Project name: Este nombre definirá tu URL gratuita. Si pones lab-docs, tu URL será <https://lab-docs.pages.dev>. (Elige el nombre que quieras para jubilar la URL aleatoria de GitHub)
- Production branch: main (o la rama principal de tu repositorio).
- Framework preset: Selecciona Docusaurus v2+ (o v3 si usas la última).
- Build command: Se rellenará solo con `npm run build` o `yarn build`.
- Output directory: Se rellenará solo con `build`.

Haz clic en **Save and Deploy**. Cloudflare levantará el contenedor, compilará tus Markdown e ilustraciones, y en unos 2-3 minutos te dará tu nueva URL funcional.

Paso 4: Ajuste obligatorio en tu código

Para que no se rompan los enlaces internos ni los estilos debido al cambio de hosting, vuelve a tu repositorio de GitHub y edita el archivo `docusaurus.config.js` con los datos de la URL que te acaba de dar Cloudflare: `JavaScript`

CODE

```
// docusaurus.config.js
module.exports = {
  title: 'Tu Título',
  url: 'https://tu-nombre-de-proyecto.pages.dev', // La URL que te dio Cloudflare
  baseUrl: '/', // Al ser un dominio propio de Cloudflare, se sirve en la raíz
  // ... resto de tu configuración
};
```

Haz git commit y git push. Cloudflare detectará este cambio y volverá a compilar la web automáticamente en segundos

Paso 5: Añadir el candado (Hacer la web privada con Cloudflare Access)

Ahora que la web está publicada bajo una URL limpia, vamos a restringir el acceso para que solo tú (o quien decidas) pueda verla

1. En el panel de Cloudflare, ve a la sección Zero Trust (está abajo en la barra lateral izquierda).
2. Si es la primera vez que entras, te pedirá elegir un nombre de equipo (por ejemplo, mi-homelab-team).
Elige el plan gratuito de 0\$ (no pide tarjeta)
3. Una vez dentro de la consola de Zero Trust, ve a Access > Applications.
4. Haz clic en Add an Application y selecciona Self-hosted.
5. Configura la aplicación:
 - Application name: Docs Privados.
 - Application URL: Selecciona el subdominio de tu proyecto de Pages (ej. lab-docs.pages.dev).
6. Haz clic en Next. Ahora crearemos la regla de acceso (Policy):
 - Policy name: Solo yo.
 - Action: Allow.
 - Configure a rule: En el selector de Selector, elige Emails. En el campo de la derecha, escribe tu correo electrónico personal
7. Haz clic en Next y luego en Save application.

¡Listo! A partir de este momento, si alguien intenta entrar en tu URL de Cloudflare Pages, se encontrará con una pantalla de autenticación. Cloudflare te enviará un código PIN de 6 dígitos a tu correo electrónico para dejarte pasar. Seguridad de nivel empresarial para tu documentación técnica sin mantener infraestructura. ¿Pudiste conectar el repositorio con éxito en este primer intento?
