LogRocket
Frontend Analytics

Deploying React apps to GitHub Pages

August 15, 2022 · 7 min read

The simplicity of deploying a static website with GitHub Pages is a process that can be easily transferred to React applications. With just a few steps, it's easy to host a React app on GitHub Pages for free or build it to deploy on your own custom domain or subdomain.

In this article, we'll explore how to deploy React apps on GitHub Pages. We'll also demonstrate how to create a custom domain on GitHub Pages for our static website.

Let's get started!

Jump ahead:

Prerequisites

- A GitHub account, or set one up
- Familiarity with Git commands
- Node.js installed, or you can install it here

What is GitHub Pages?

GitHub Pages is a service from GitHub that enables you to add HTML, JavaScript, and CSS files to a repository and create a hosted static website.

The website can be hosted on GitHub's `github.io` domain (e.g., `https://username.github.io/repositoryname`) or on your own custom domain. A React app can be hosted on GitHub Pages in a similar manner.

How to deploy a React application to GitHub Pages

To deploy your React application to GitHub Pages, follow these steps:

1. Set up your React application
2. Create a GitHub repository for your project
3. Push your React app to your GitHub repository

Setting up the React application

Let's get started by creating a new React application. For this tutorial, we'll be using `create-react-app` but you can set up the project however you prefer.

Open the terminal on your computer and navigate to your preferred directory. For this tutorial, we'll set up the project in the desktop directory, like so:

Create a React application using `create-react-app`:

```
npx create-react-app "your-project-name"
```

In just a few minutes, `create-react-app` will have finished setting up a new React application!

Now, let's navigate into the newly created React app project directory, like so:

```
cd "your-project-name"
```

This tutorial is limited to demonstrating how to deploy a React application to GitHub Pages, so we'll leave the current setup as it is without making any additional changes.

Creating a GitHub repository

The next step is to create a GitHub repository to store our project's files and revisions.

In your GitHub account, click the + icon in the top right and follow the prompts to set up a new repository.

After your repository has been successfully created, you should see a page that looks like this:

Awesome! Let's proceed to the next step.

Pushing the React app to the GitHub repository

Tracking and syncing changes

Initialize Git with the following command:

```
git init
```

Pushing the code to the GitHub repo

Now, we'll commit our code and push it to our branch on GitHub. To do this, simply copy and paste the code received when you created a new repository (see the above repo screenshot).


```
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/nelsonmic/testxx.git
git push -u origin main
```

Adding the GitHub Pages dependency package

Next, we'll install the `gh-pages` package in our project. The package allows us to publish build files into a `gh-pages` branch on GitHub, where they can then be hosted.

Install `gh-pages` as a dev dependency via npm:

Adding the deploy scripts

Now, let's configure the `package.json` file so that we can point our GitHub repository to the location where our React app will be deployed.

We'll also need to add `predeploy` and `deploy` scripts to the `package.json` file. The `predeploy` script is used to bundle the React application; the `deploy` script deploys the bundled file.

In the `package.json` file, add a `homepage` property that follows this structure: `http://{github-username}.github.io/{repo-name}`

Now, let's add the scripts.

In the `package.json` file, scroll down to the `scripts` property and add the following commands:

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

Here's a visual reference:

That's it! We've finished configuring the `package.json` file.

Committing changes and pushing code updates to the GitHub repo

Now, let's commit our changes and push the code to our remote repository, like so:

```
git add .
git commit -m "setup gh-pages"
git push
```

We can deploy our React application by simply running: `npm run deploy`. This will create a bundled version of our React application and push it to a `gh-pages` branch in our remote repository on GitHub.

To view our deployed React application, navigate to the **Settings** tab and click on the **Pages** menu. You should see a link to the deployed React application.

Adding a custom domain

We can deploy our React app to GitHub's domain for free, but Github Pages also supports custom subdomains Apex domains. Here are examples showing what each type of subdomain looks like:

Supported custom domain	Example
`www` subdomain	`www.logdeploy.com`
Custom subdomain	`app.logdeploy.com`
Apex domain	`logdeploy.com`

Right now, if we navigate to https://nelsonmic.github.io/logdeploy/ we'll see our recently published website. But, we could also use a custom subdomain or an Apex domain instead.

Here are the steps to set those up:

Deploying to a GitHub custom subdomain

1. Purchase a domain name from a domain service provider of your choosing (e.g., Namecheap or GoDaddy)
2. Connect the custom domain to GitHub Pages. To do so, click on the **Pages** menu on the **Settings** tab. Next, scroll down to the **Custom domain** field and

3. Ensure the `CNAME` record on your domain service provider points to the GitHub URL of the deployed website (in the case of this example, nelsonmic.github.io/logdeploy/). To do so, navigate to the **DNS management** page of the domain service provider and add a `CNAME` record that points to `username.github.io` where `username` is your GitHub username

Deploying to a GitHub Apex domain

To deploy to an Apex domain, follow the first two steps for deploying to a custom subdomain but substitute the below for the third step:

1. Navigate to the **DNS management** page of the domain service provider and add an `ALIAS` record or `ANAME` record that points your Apex domain to your GitHub Pages IP addresses, as shown:

- 185.199.108.153
- 185.199.109.153
- 185.199.110.153
- 185.199.111.153

How to deploy a React app with routing to GitHub Pages

If you've previously deployed a React app that uses React Router for routing to Netlify, you're aware that you need to configure redirects for your URLs. Without redirects, users will get a 404 error when they try to navigate to different parts of your application.

Netlify makes it simple to configure redirects and rewrite rules for your URLs. All you need to do is create a file called `_redirects` (without any extensions) in the app's public folder.

```
/*    /index.html   200
```

No matter what URL the browser requests, this rewrite rule will deliver the `index.html` file instead of returning a 404.

If we want to handle page routing when we deploy to GitHub Pages, we'll need to do something similar. Let's configure routing for our previously deployed project.

First, we need to install a router. Start by installing React Router in the project directory, like so:

```
npm install react-router-dom
```

Now, follow the below four steps.

Step 1: Connect a `HashRouter` to the application to enable client-side routing:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import { HashRouter as Router } from "react-router-dom";

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
    <Router>
      <App />
    </Router>
);
// If you want to start measuring performance in your app, pass a function
// to log results (for example, reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
```

Our `index.js` file should look like the above code block. Since GitHub Pages does not support browser history, we're employing a `HashRouter`. Our existing path does not assist GitHub Pages in determining where to direct the user (since it is a frontend route).

To solve this issue, we must replace our application's browser router with a `HashRouter`. The hash component of the URL is used by this router to keep the UI in sync with the URL.

Step 2: Create the routes:

Create a `routes` folder and the required routes. These routes can be configured in the `app.js` file. But first, let's create a `Navbar` component that can be visible on all pages.

Step 3: Create a `Navbar` component:

```
>import { Link } from "react-router-dom"
const Navbar =()=>{
      return (
            <div>
                  <Link to="/">Home</Link>
                  <Link to="/about">About</Link>
                  <Link to="/careers">Careers</Link>
            </div>
      )
}
export default Navbar;
```

Now we can add the `Navbar` component alongside the configured routes in the `app.js` file.

Step 4: Set up routes in the `app.js` file:

```
import './App.css';
import { Routes, Route} from "react-router-dom";
import About from "./routes/About";
import Careers from "./routes/Careers";
import Home from "./routes/Home";
import Navbar from './Navbar';
function App() {
  return (
    <>
      <Navbar />
      <Routes>
        <Route path="/" element={<Home />} />
        <Route path="/about" element={<About />} />
        <Route path="/careers" element={<Careers />} />
      </Routes>
    </>
  );
}
export default App;
```

Now that we're done with the setup, let's push our code, like so:

```
>git add .
git commit -m "setup gh-pages"
git push
```

Next, we simply deploy and our app should route properly:

```
>npm run deploy
```

Once these steps are completed, our deployed application will correctly route the user to any part of the application they desire.

Setting up a preview environment

When we configure Netlify deployments, we're given a preview link to view our deployment before it is merged into the main branch. Let's create the same for GitHub Pages.

We'll use a simple tool called Livecycle for this, which saves us the trouble of having to do this using GitHub Actions. Every time we make a pull request in our repository, Livecycle assists us in creating a preview environment.

To create a preview environment, follow these steps:

1. Set up a Livecycle account; I recommend signing up with your GitHub account
2. Configure a new project and connect it to GitHub
3. Once you've connected to GitHub and granted Livecycle all the access it needs, you can select the repository you want to set up the preview on

Select a template based on your project's tech stack. For our example, we'd select **Create React App NPM**, since our project was built using create-react-app:

1. Review the configuration and deploy. For our example, we don't need to add anything to the configurations:

Once deployment is successful, anytime we make a PR or push a commit to that PR we'll get a preview link.

Conclusion

GitHub Pages is easy to get started with and free to use, making it a very attractive option for developers of all skill levels.

In this article, we demonstrated how to use GitHub Pages to convert a React app

code with the world, GitHub Pages is a great option.

Nelson Michael Follow

Nelson Michael is a frontend developer from Nigeria. When he's not meddling with CSS, he spends his time writing, sharing what he knows, and playing games.

#react

11 Replies to “Deploying React apps to GitHub Pages”

Kevin Says:
September 18, 2022 at 4:51 am

Reply

The deploy script code has an error. There should be a space before -d

Lauren Saalmuller Says:
September 27, 2022 at 9:15 am

Reply

Hello Kevin,

We’ve corrected the typo. Thanks for reading LogRocket’s blog!

Nelson Michael Says:
September 27, 2022 at 9:16 am

Reply

Thank you for pointing this out.

Henryd0Henry Says:

Reply

Nelson Michael Says:
March 27, 2023 at 9:29 am

Reply

Hi Henry, thank you for reading. Can you provide more context to this problem?

Fazeelat Suleman Says:
September 27, 2022 at 6:04 am

Reply

“gh-pages-d build”, there is a space in -d. if you would not add space you can get message that “‘gh-pages-d’ is not recognized as an internal or external command, operable program or batch file. ”
should be like this
“deploy”: “gh-pages -d build”

Lauren Saalmuller Says:
September 27, 2022 at 9:14 am

Reply

Hi Fazeelat, thanks for pointing out the typo. We’ve fixed it. Thanks for reading LogRocket’s blog!

Yassine_dev1 Says:
December 15, 2022 at 2:34 pm

Reply

Thank you for the artice, very well put together.

Nelson Michael Says:
December 16, 2022 at 5:28 pm

Reply

Thank you! I’m glad it was a good read.

Thank you for putting everything very well. I could easily go through the steps and got the deploying done. A big thank you

lpskipzip Says:
May 11, 2023 at 1:55 pm

Reply

Hi Michael, thank for posting this. I'm getting an error near the beginning.
npm install gh-pages –save-dev

Unsupported engine {
npm WARN EBADENGINE package: '@csstools/selector-specificity@2.2.0',
npm WARN EBADENGINE required: { node: '^14 || ^16 || >=18' },
npm WARN EBADENGINE current: { node: 'v17.9.1', npm: '8.11.0' }

I think this is connected to problems I'm having using the command line to manage my repository on github... but maybe I fixed that and this is a new problem...?

Also, I'd like to subscribe to your blog, but I'm not sure what logrocket is and how to subscribe without paying a membership fee.

Leave a Reply

Enter your comment here...