Practical Implementation of the Docker with React.js Web Application

To demonstrate the practical use of Docker, here we will use it to run a React.js web application.

- Now normally to run a React.js web application, we need to install Node.js and then install the React.js framework. But with Docker, we can run the React.js web application without installing Node.js or React.js framework.

  First we need to create a React.js web application. To do that I have already installed Node.js and React.js framework on my machine.

  To install Node for your device you can download it from here.

- Here I have used Vite.js to create a React.js web application.

  Now to create a React.js web application, we will use the vite plugin `react` to create a React.js web application.

  ```
  npm create vite@latest . -- --template react
  ```

  Output:

  ```
  ✔ Current directory is not empty. Remove existing files and continue? …
  yes

  Scaffolding project in
  /Users/neelanjanmukherji/Learning/Programming/docker/react-app...

  Done. Now run:

  npm install
  npm run dev
  ```

- Then we need to install the dependencies for the React.js web application.

  ```
  npm install
  ```

  > Here we have used `npm` to install the dependencies. But you can also use `yarn` to install the dependencies.

  Output:

  ```
  added 242 packages, and audited 243 packages in 7s


  82 packages are looking for funding
  ```

```
    run "npm fund' for details


2 moderate severity vulnerabilities

To address all issues (including breaking changes) , run:
    mpm audit fix --force


Run 'npm audit' for details.
```

> Note: The vulnerabilities are not a big issues and can vary from machine to machine.

> **This is the main reason why we use Docker to run the React.js web application. So that we don't have to install Node.js and React.js framework on our machine and we can run the React.js web application without any issues.**

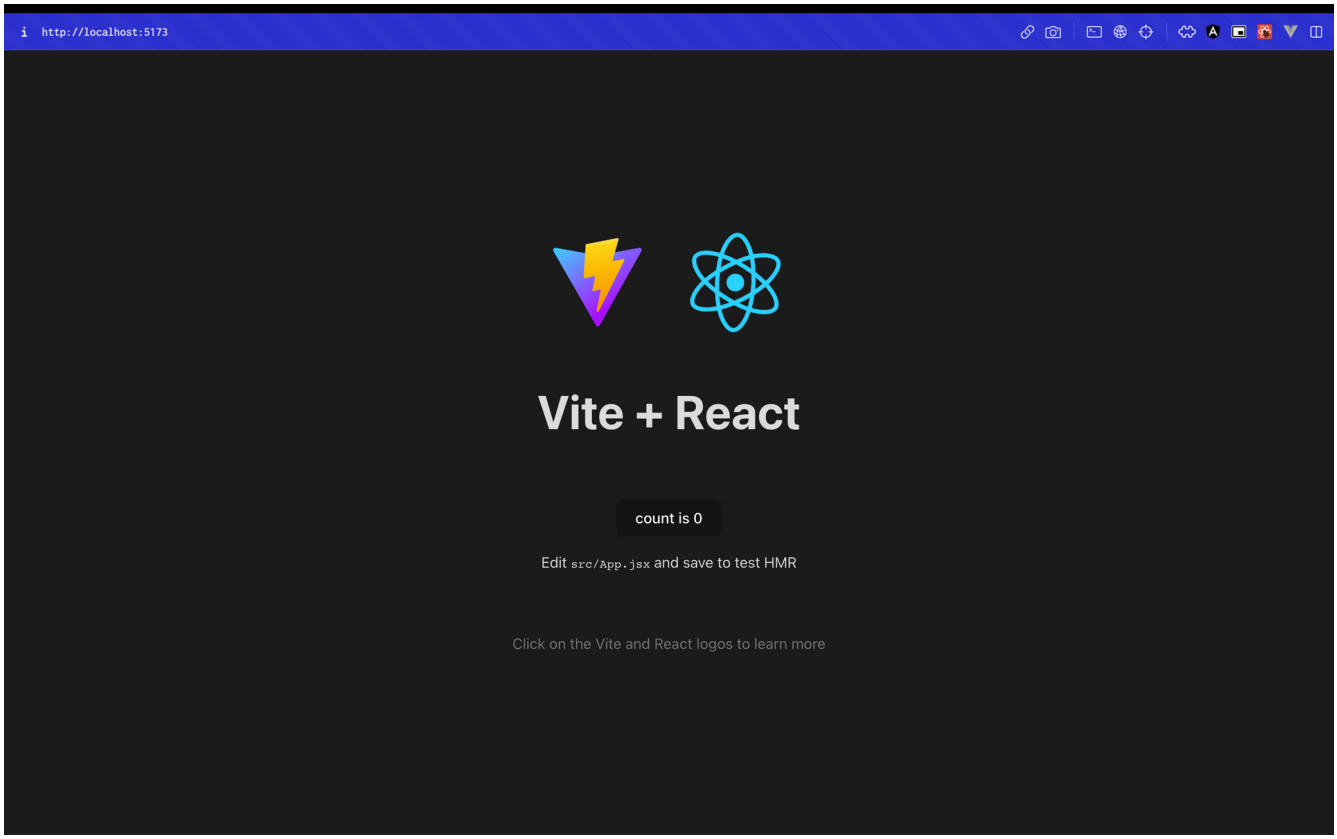
- Now we can run the React.js web application.

```
npm run dev
```

Output:

```
> react-app@0.0.0 dev
> vite


VITE v4.4.2  ready in 984 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  press h to show help
```

- The React.js web application will look something like this.



- But let's edit the React.js web application a little bit.

  This is the App.jsx file.

```
import { useState } from 'react'
import reactLogo from './assets/react.svg'
import myLogo from './assets/myLogo.svg'
import dockerLogo from './assets/docker.svg'
import viteLogo from './assets/vite.svg'
import gitLogo from './assets/git.svg'
import githubLogo from './assets/github-2.svg'
import './App.css'

function App() {
const [count, setCount] = useState(0)

return (
    <>
    <div>
        <a href="https://vitejs.dev" target="_blank" rel="noreferrer">
            <img src={viteLogo} className="logo" alt="Vite logo" />
        </a>
        <a href="https://react.dev" target="_blank" rel="noreferrer">
            <img src={reactLogo} className="logo react" alt="React logo"
/>
        </a>
        <a href="https://github.com/Maverick7274" target="_blank"
```

```
rel="noreferrer">
            <img src={myLogo} className="logo mylogo" alt="My logo" />
        </a>
        <a href="https://www.docker.com/" target="_blank"
rel="noreferrer">
            <img src={dockerLogo} className="logo docker" alt="Docker
logo" />
        </a>
        <a href="https://git-scm.com/" target="_blank" rel="noreferrer">
        <img src={gitLogo} className="logo git" alt="git logo" />
        </a>
        <a href="https://github.com/" target="_blank" rel="noreferrer">
            <img src={githubLogo} className="logo github" alt="GitHub
logo" />
        </a>
        <a href="https://overleaf.com/" target="_blank" rel="noreferrer">
            <img
src="https://images.ctfassets.net/nrgyaltdicpt/h9dpHuVys19B1sOAWvbP6/5f8d
4c6d051f63e4ba450befd56f9189/ologo_square_colour_light_bg.svg"
className="logo overleaf" alt="GitHub logo" />
        </a>
    </div>
    <h1>Vite + React + Git + Docker + GitHub</h1>
    <div className="card">
        <button onClick={() => setCount((count) => count + 1)}>
        count is {count}
        </button>
        <p>This is a Test for Docker</p>
        <div className='link-tab'>
        <a href='https://www.overleaf.com/read/rzrdkkhdgpph'
rel='noreferrer' target='_blank'>
            Overleaf Report
        </a>
        <a href='https://github.com/Maverick7274/NTCC-Report-2023'
rel='noreferrer' target='_blank'>
            Main NTCC GitHub Repository
        </a>
        <a href='https://github.com/Maverick7274/NTCC-Git.git'
rel='noreferrer' target='_blank'>
            Git & GitHub Practical Experiment Repository
        </a>
        <a href='https://github.com/Maverick7274/NTCC-Docker.git'
rel='noreferrer' target='_blank'>
```

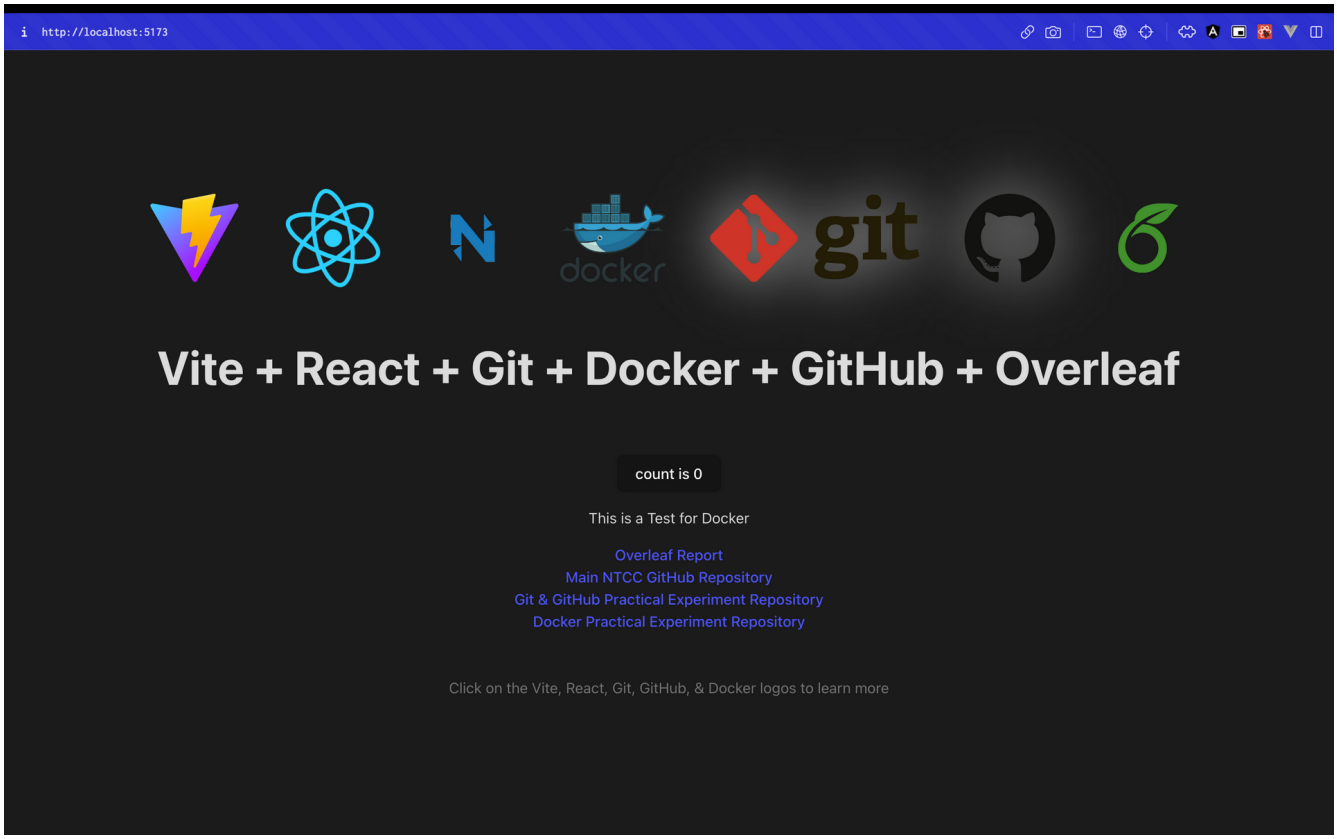
```
            Docker Practical Experiment Repository
        </a>
        </div>
    </div>
    <p className="read-the-docs">
        Click on the Vite, React, Git, GitHub, & Docker logos to learn
more
    </p>
    </>
)
}

export default App
```

This is the App.css file.

```
#root {
    max-width: 1280px;
    margin: 0 auto;
    padding: 2rem;
    text-align: center;
}

.logo {
    height: 6em;
    padding: 1.5em;
    will-change: filter;
    transition: filter 300ms;
}
.logo:hover {
    filter: drop-shadow(0 0 2em #646cffaa);
}
.logo.react:hover {
    filter: drop-shadow(0 0 2em #61dafbaa);
}
.logo.mylogo:hover {
    filter: drop-shadow(0 0 2em #97daf6aa);
}
.logo.docker:hover {
    filter: drop-shadow(0 0 2em #22A0C8aa);
}
.logo.git {
    filter: drop-shadow(0 0 2em #d0d0d0);
}
```

```
.logo.git:hover {
    filter: drop-shadow(0 0 2em #DE4C36aa);
}
.logo.github {
    filter: drop-shadow(0 0 2em #d0d0d0);
}
.logo.github:hover {
    filter: drop-shadow(0 0 2em #121110aa);
}

@keyframes logo-spin {
    from {
        transform: rotate(0deg);
    }
    to {
        transform: rotate(360deg);
    }
}

@media (prefers-reduced-motion: no-preference) {
    a:nth-of-type(2) .logo {
        animation: logo-spin infinite 20s linear;
    }
}

.card {
    padding: 2em;
}

.read-the-docs {
    color: #888;
}

.link-tab {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
}
```

Now the React.js web application will look something like this.



> **Note:** It is not necessary to edit or understand the React.js web application. You can skip this step.

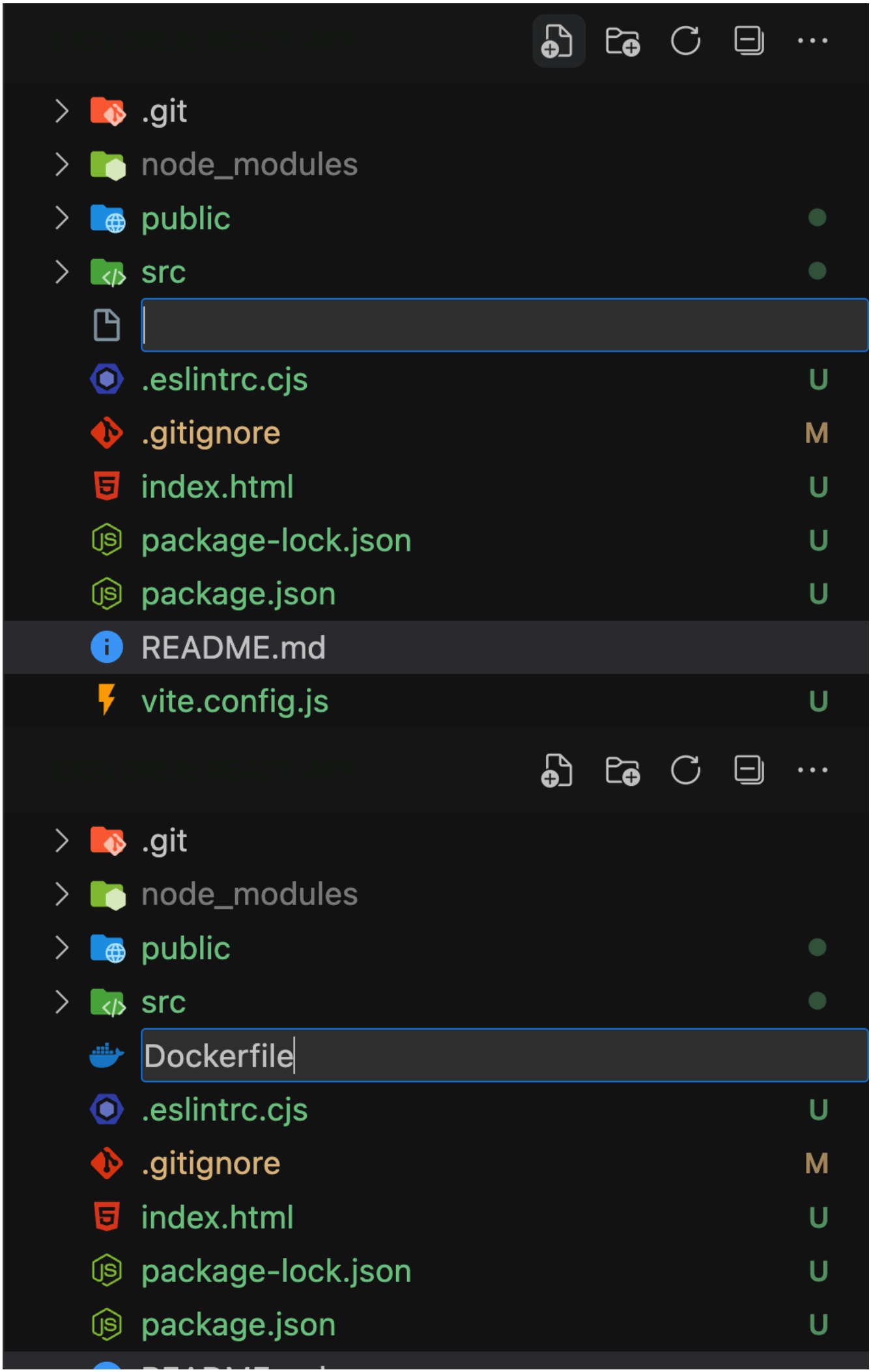
> **Note:** You can edit the React.js web application as you like.

- Now to add the Dockerfile, we need to create a file named `Dockerfile` in the root directory of the React.js web application.

```
touch Dockerfile
```

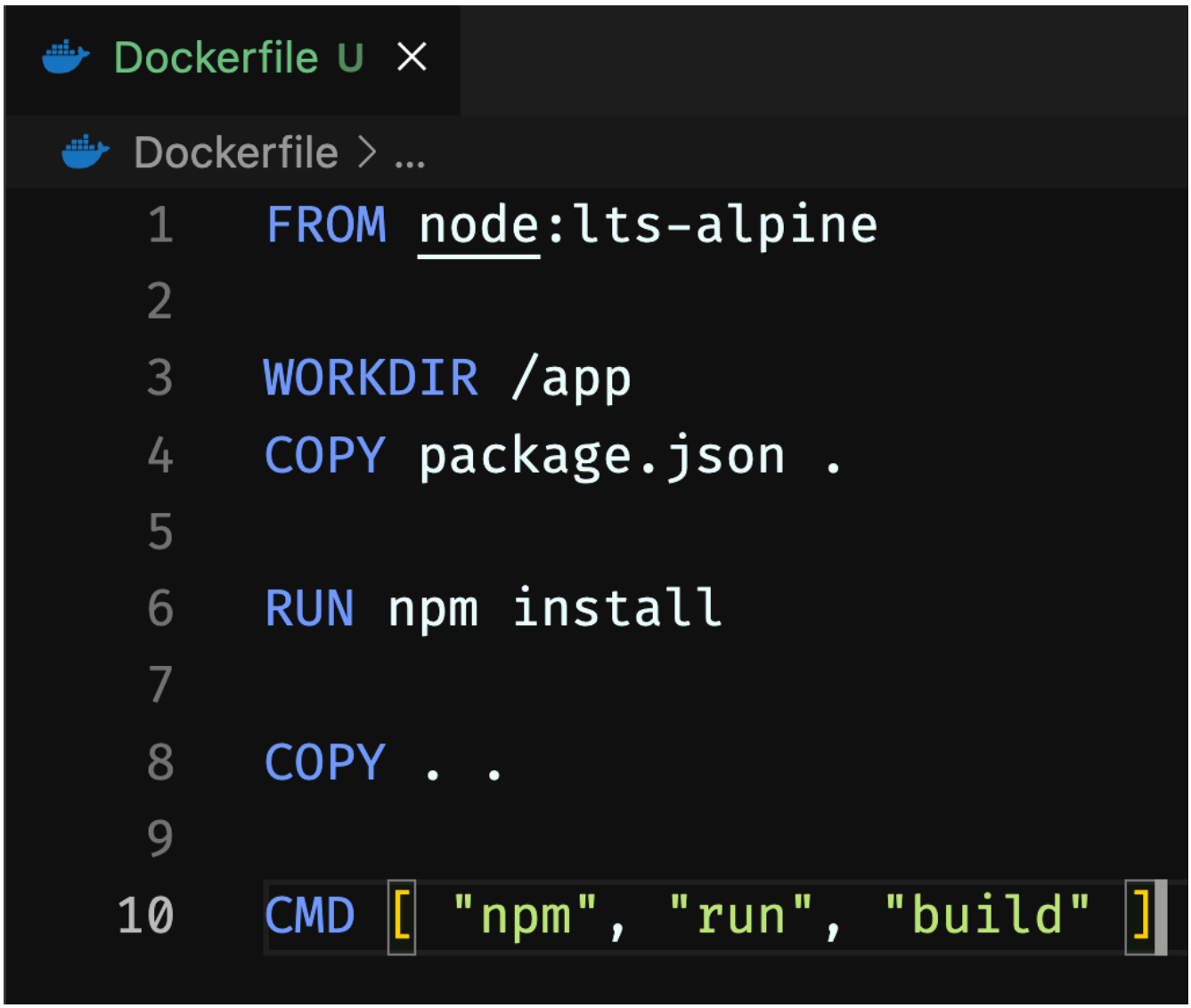
OR

Just create a file named `Dockerfile` in the root directory of the React.js web application.


.git
node_modules
public
src
.eslintrc.cjs
U
.gitignore
M
index.html
U
package-lock.json
U
package.json
U
README.md
vite.config.js
U
.git
node_modules
public
src
Dockerfile
.eslintrc.cjs
U
.gitignore
M
index.html
U
package-lock.json
U
package.json
U

- Now we need to add the following code to the `Dockerfile`.

```
FROM node:lts-alpine

WORKDIR /app
COPY package.json .

RUN npm install

COPY . .

CMD [ "npm", "run", "build" ]
```



  - Here we have used the `node:lts-alpine` image as the base image.
  - Then we have set the working directory to `/app`.
  - Then we have copied the `package.json` file to the working directory.

- Then we have installed the dependencies.
- Then we have copied the whole React.js web application to the working directory.
- Then we have used the `CMD` command to run the `npm run build` command.

> Note: Here we have used the `npm run build` command to build the React.js web application.

> If you want to run the React.js web application in development mode then you can use the `npm run dev` command.

> It is recommended to use the `npm run build` command to build the React.js web application.

- Now we need to build the Docker image.

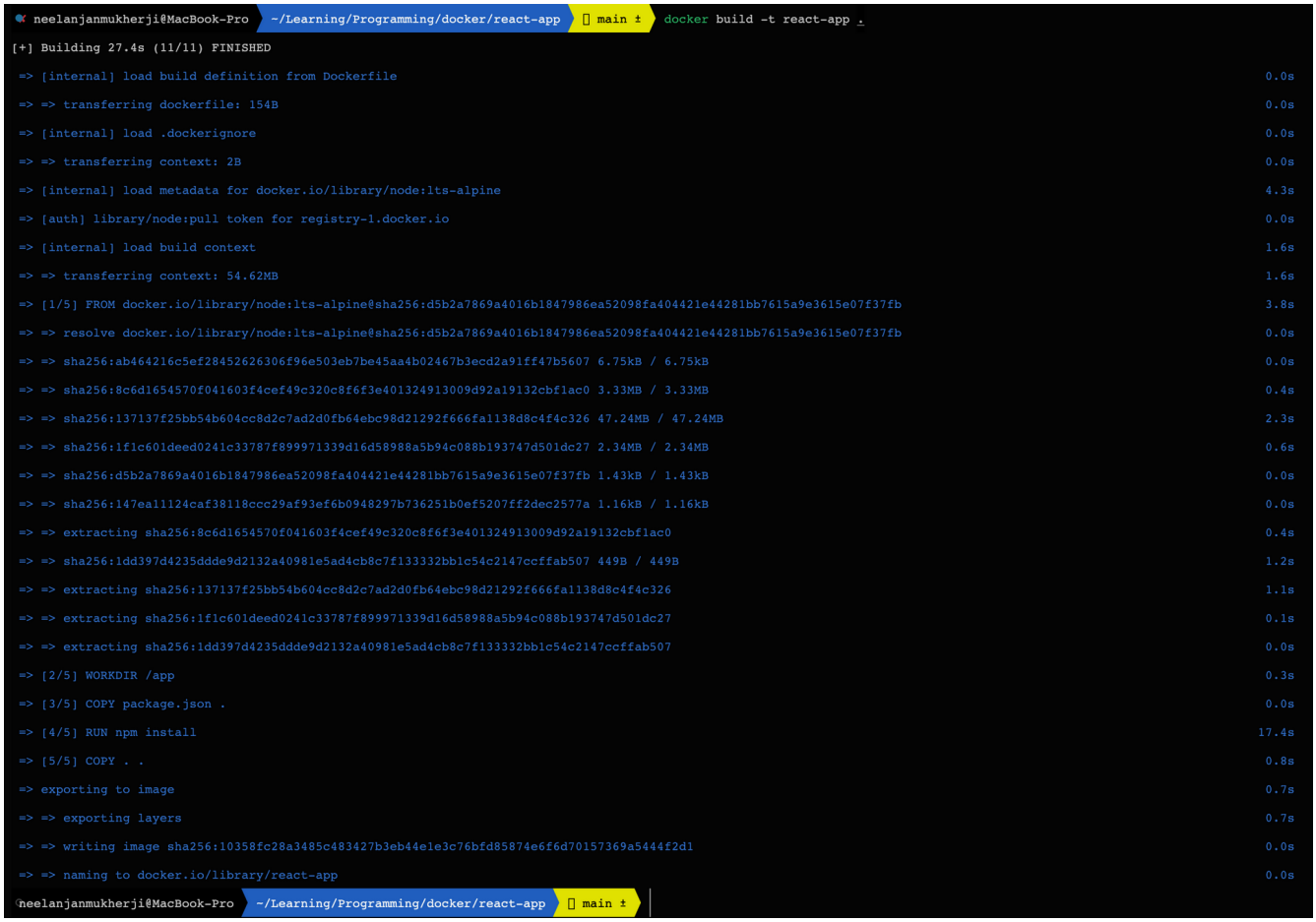
```
docker build -t react-app .
```

Output:

```
[+] Building 27.4s (11/11) FINISHED
=> [internal] load build definition from Dockerfile
0.0s
=> => transferring dockerfile: 154B
0.0s
=> [internal] load .dockerignore
0.0s
=> => transferring context: 2B
0.0s
=> [internal] load metadata for docker.io/library/node:lts-alpine
4.3s
=> [auth] library/node:pull token for registry-1.docker.io
0.0s
=> [internal] load build context
1.6s
=> => transferring context: 54.62MB
1.6s
=> [1/5] FROM docker.io/library/node:lts-
alpine@sha256:d5b2a7869a4016b1847986ea52098fa404421e44281bb7615a9e3615e07
f37fb                                                    3.8s
=> => resolve docker.io/library/node:lts-
alpine@sha256:d5b2a7869a4016b1847986ea52098fa404421e44281bb7615a9e3615e07
f37fb                                                    0.0s
=> =>
sha256:ab464216c5ef28452626306f96e503eb7be45aa4b02467b3ecd2a91ff47b5607
6.75kB / 6.75kB
```

```
0.0s
=> =>
sha256:8c6d1654570f041603f4cef49c320c8f6f3e401324913009d92a19132cbf1ac0
3.33MB / 3.33MB
0.4s
=> =>
sha256:137137f25bb54b604cc8d2c7ad2d0fb64ebc98d21292f666fa1138d8c4f4c326
47.24MB / 47.24MB
2.3s
=> =>
sha256:1f1c601deed0241c33787f899971339d16d58988a5b94c088b193747d501dc27
2.34MB / 2.34MB
0.6s
=> =>
sha256:d5b2a7869a4016b1847986ea52098fa404421e44281bb7615a9e3615e07f37fb
1.43kB / 1.43kB
0.0s
=> =>
sha256:147ea11124caf38118ccc29af93ef6b0948297b736251b0ef5207ff2dec2577a
1.16kB / 1.16kB
0.0s
=> => extracting
sha256:8c6d1654570f041603f4cef49c320c8f6f3e401324913009d92a19132cbf1ac0
0.4s
=> =>
sha256:1dd397d4235ddde9d2132a40981e5ad4cb8c7f133332bb1c54c2147ccffab507
449B / 449B
1.2s
=> => extracting
sha256:137137f25bb54b604cc8d2c7ad2d0fb64ebc98d21292f666fa1138d8c4f4c326
1.1s
=> => extracting
sha256:1f1c601deed0241c33787f899971339d16d58988a5b94c088b193747d501dc27
0.1s
=> => extracting
sha256:1dd397d4235ddde9d2132a40981e5ad4cb8c7f133332bb1c54c2147ccffab507
0.0s
=> [2/5] WORKDIR /app
0.3s
=> [3/5] COPY package.json .
0.0s
=> [4/5] RUN npm install
17.4s
```

```
=> [5/5] COPY . .
0.8s
=> exporting to image
0.7s
=> => exporting layers
0.7s
=> => writing image
sha256:10358fc28a3485c483427b3eb44e1e3c76bfd85874e6f6d70157369a5444f2d1
0.0s
=> => naming to docker.io/library/react-app
0.0s
```



> Note: Here we have used the `-t` flag to tag the Docker image with the name `react-app`.

Notice that we followed similar commands like `npm install` in running the React.js web application in our local machine. That is the beauty of Docker. We can now run the React.js web application in the same way in any machine.

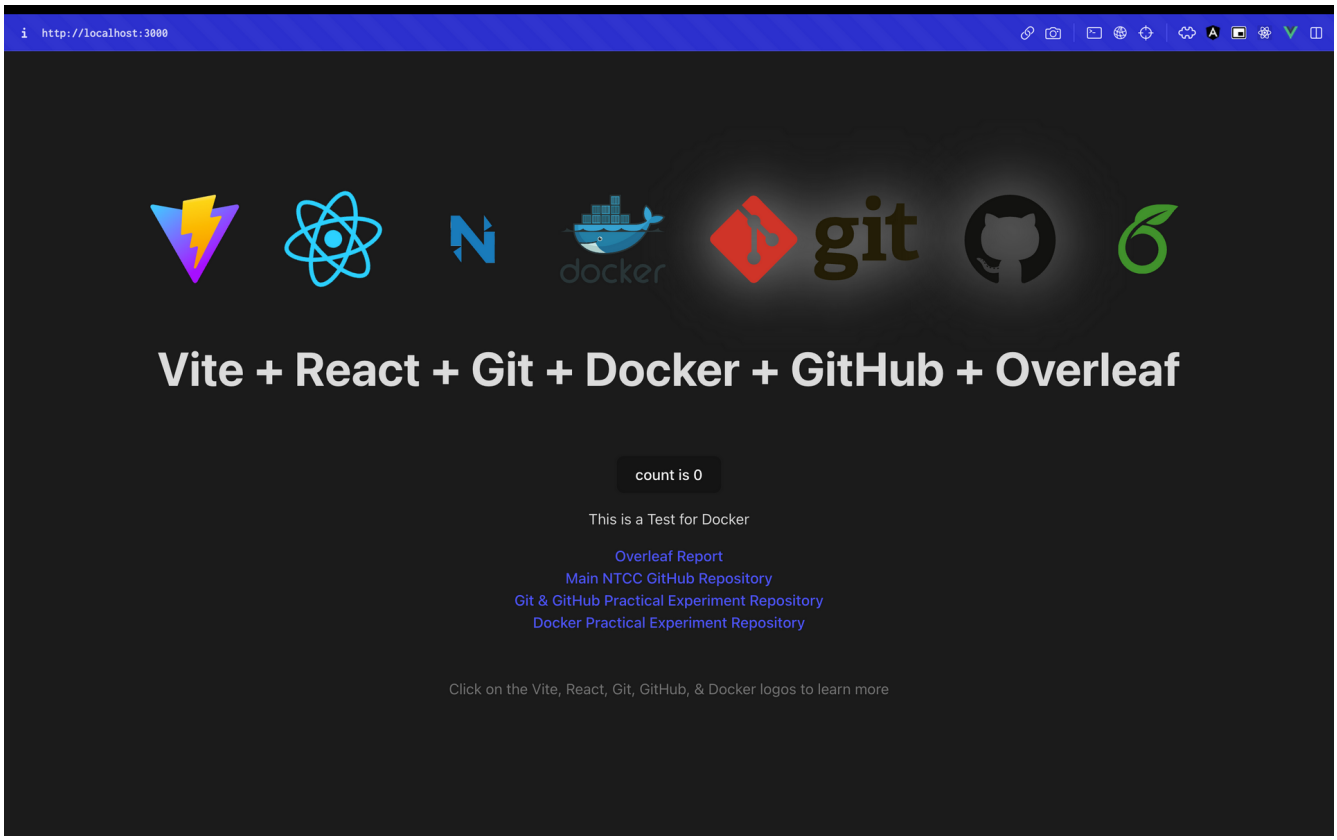
- Now we need to run the Docker image.

```
docker run -it -p 3000:3000 react-app
```

Output:

```
> react-app@0.0.0 build
> vite build

vite v4.4.2 building for production...
✓ 38 modules transformed.
dist/index.html                          0.45 kB │ gzip:  0.30 kB
dist/assets/vite-4a748afd.svg            1.50 kB │ gzip:  0.77 kB
dist/assets/github-2-fa93a6d5.svg        2.06 kB │ gzip:  1.09 kB
dist/assets/git-79cc113c.svg             2.21 kB │ gzip:  1.15 kB
dist/assets/react-35ef61ed.svg           4.13 kB │ gzip:  2.14 kB
dist/assets/docker-85301acb.svg          9.57 kB │ gzip:  3.67 kB
dist/assets/myLogo-a77f6266.svg          9.93 kB │ gzip:  4.59 kB
dist/assets/index-2e679e36.css           1.89 kB │ gzip:  0.85 kB
dist/assets/index-6712cc1a.js          145.09 kB │ gzip: 46.56 kB
✓ built in 1.11s
```

- Now we can open the React.js web application in the browser by navigating to `http://localhost:3000`.



> Note: Here we have used the `-p` flag to map the port `3000` of the Docker container to the port `3000` of the local machine.

- Now we have successfully deployed the React.js web application in the Docker container.
- Now we will Deploy this website on the Internet using GitHub Actions and Docker.
- First push the code to GitHub.

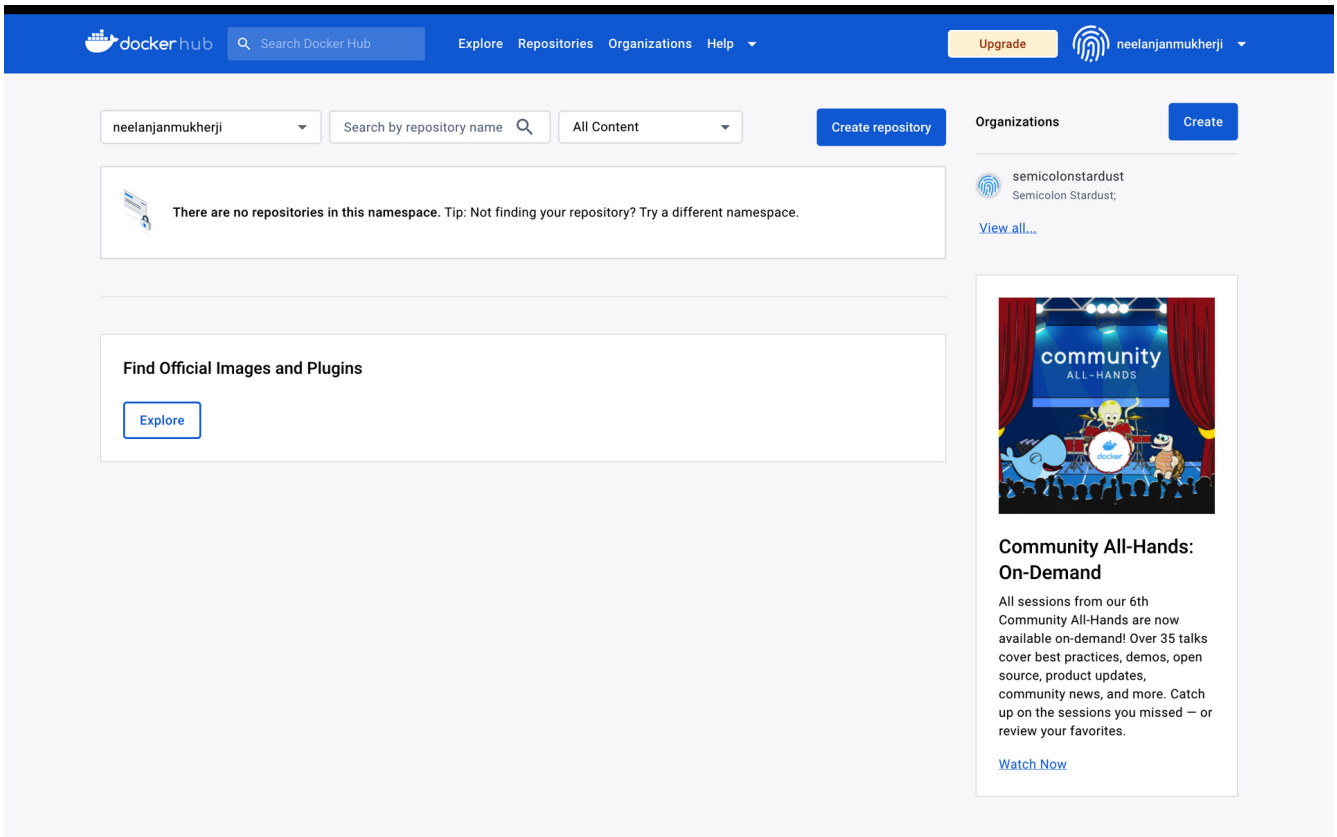
```
git add .
git commit -m "Dockerized the React.js web application"
git push origin main
```

Output:

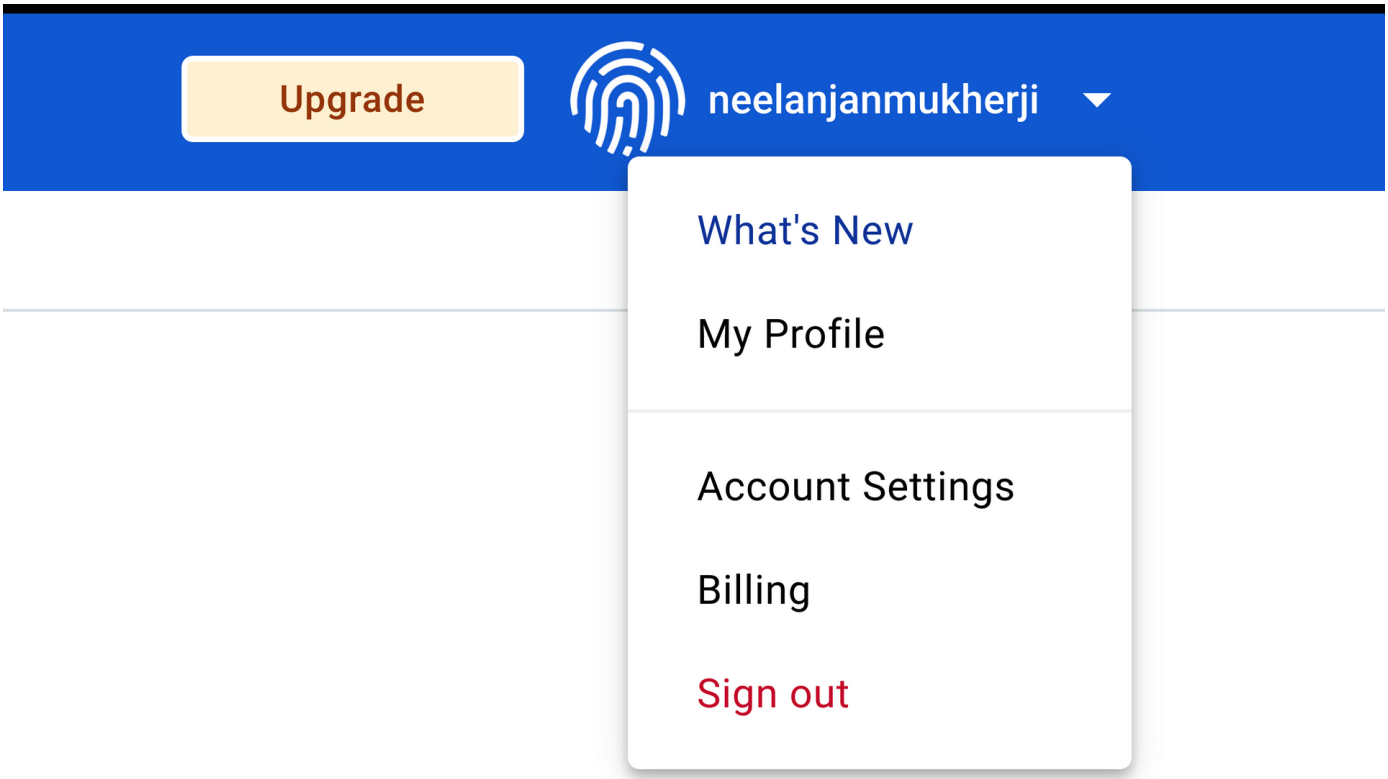
```
[main 0f794c3] Dockerized the React.js web application
17 files changed, 3770 insertions(+), 42 deletions(-)
create mode 100644 .eslintrc.cjs
rewrite .gitignore (96%)
create mode 100644 Dockerfile
create mode 100644 index.html
create mode 100644 package-lock.json
create mode 100644 package.json
create mode 100644 src/App.css
create mode 100644 src/App.jsx
create mode 100644 src/assets/docker.svg
create mode 100644 src/assets/git.svg
create mode 100644 src/assets/github-2.svg
```

```
create mode 100644 src/assets/myLogo.svg
create mode 100644 src/assets/react.svg
create mode 100644 src/assets/vite.svg
create mode 100644 src/index.css
create mode 100644 src/main.jsx
create mode 100644 vite.config.js
Enumerating objects: 23, done.
Counting objects: 100% (23/23), done.
Delta compression using up to 8 threads
Compressing objects: 100% (21/21), done.
Writing objects: 100% (21/21), 46.03 KiB | 11.51 MiB/s, done.
Total 21 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Maverick7274/NTCC-Docker.git
ea22cc6..0f794c3  main -> main
```

- Now we will create a GitHub Actions workflow for building and deploying the React.js web application.
- Here first we are supposed to create a Docker Hub account.

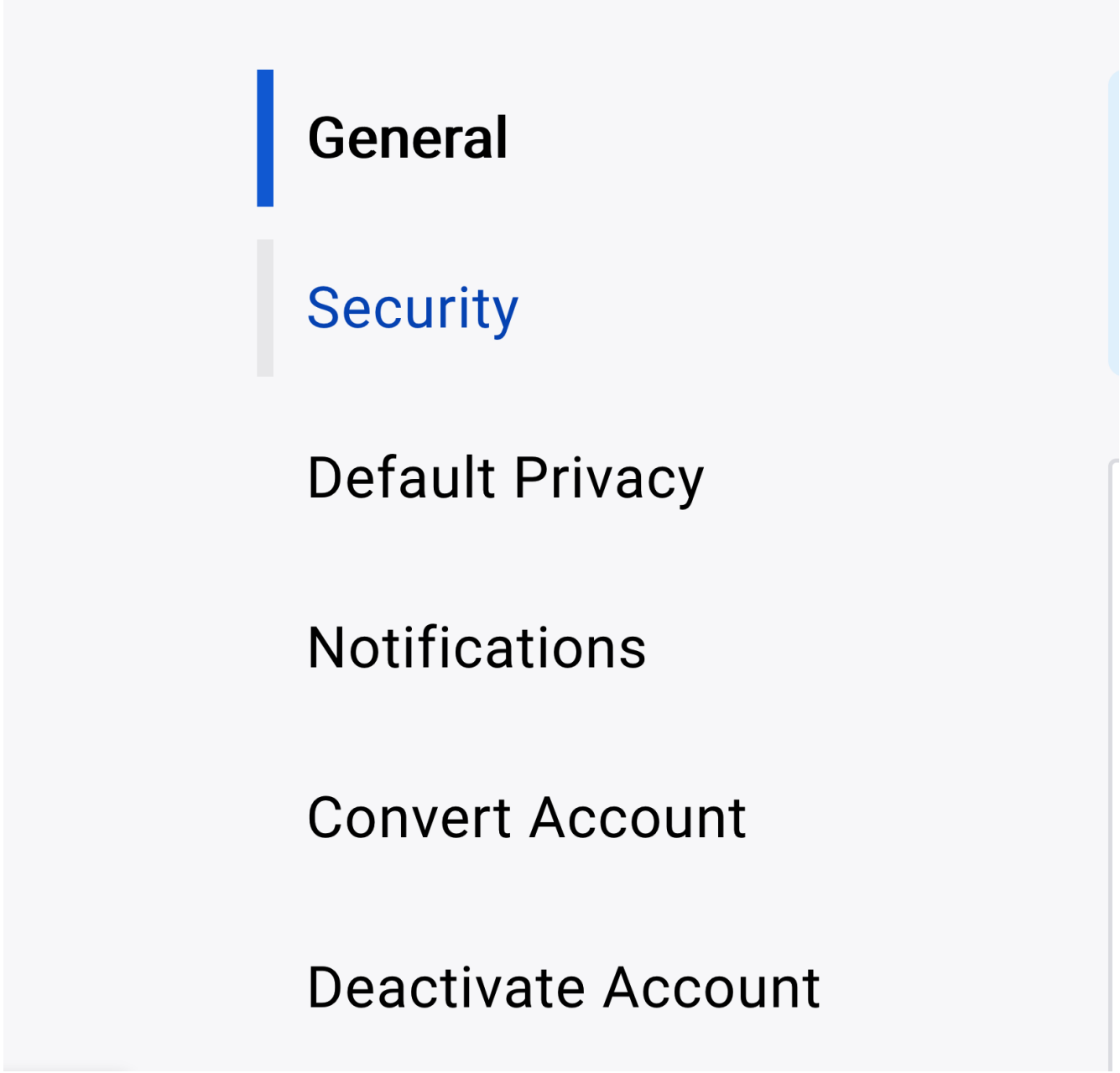


- Then goto `Account Settings` of Docker Hub.

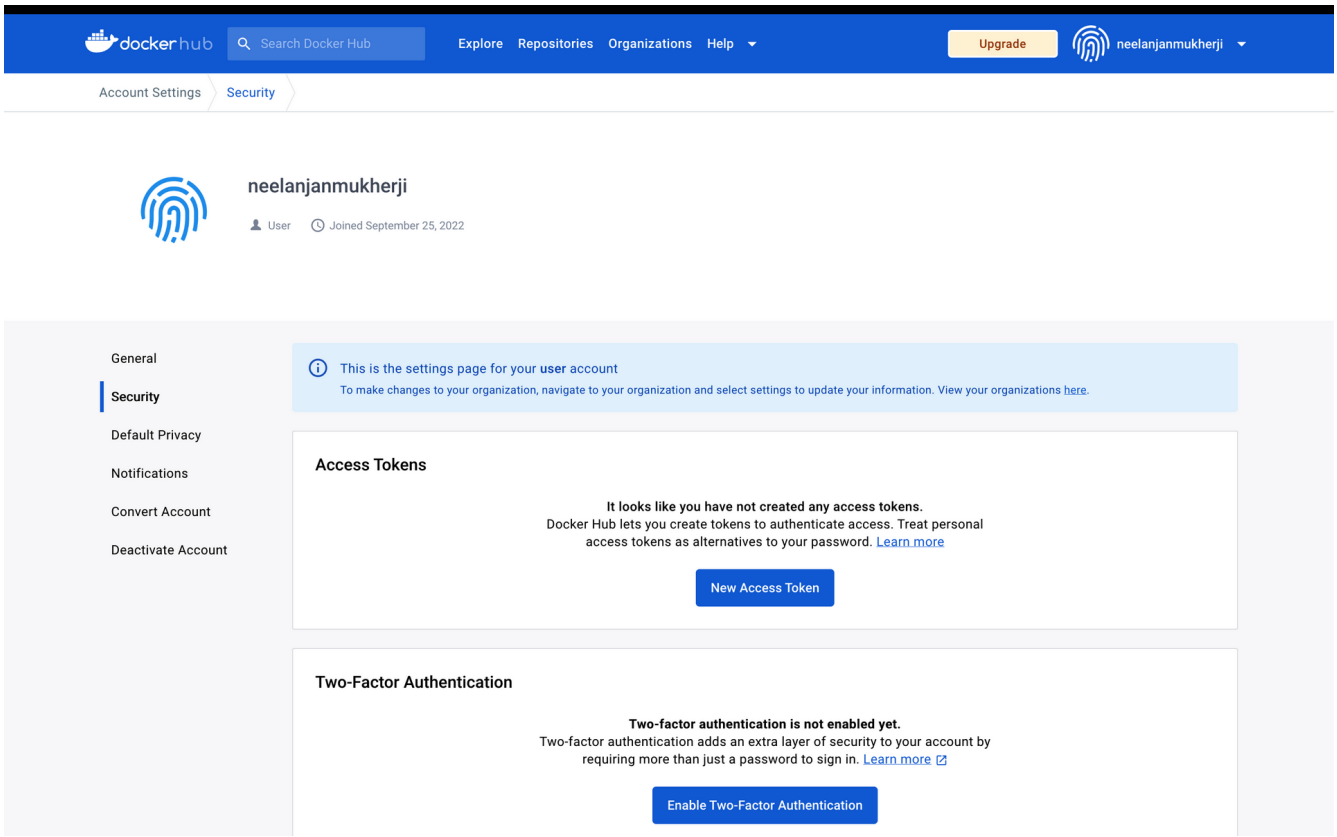


www.docker.com/blog/category/products

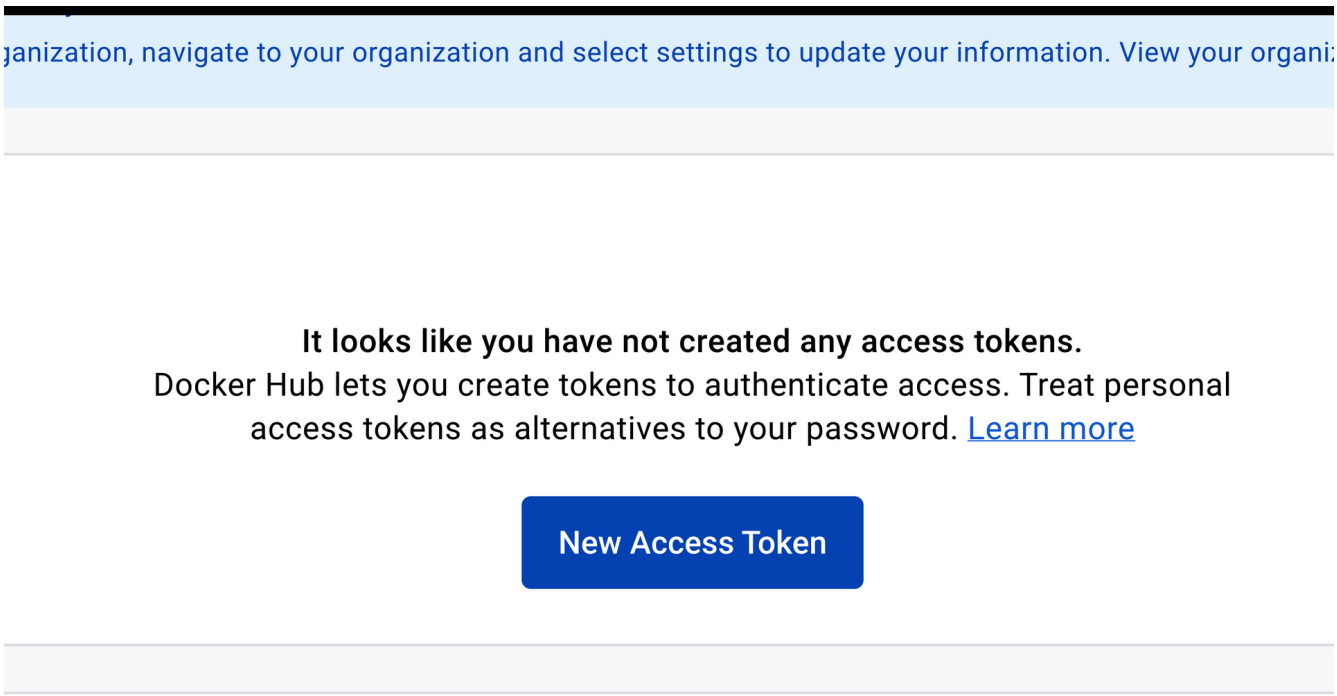
- Now goto `Security` tab.



The page should look something like this.

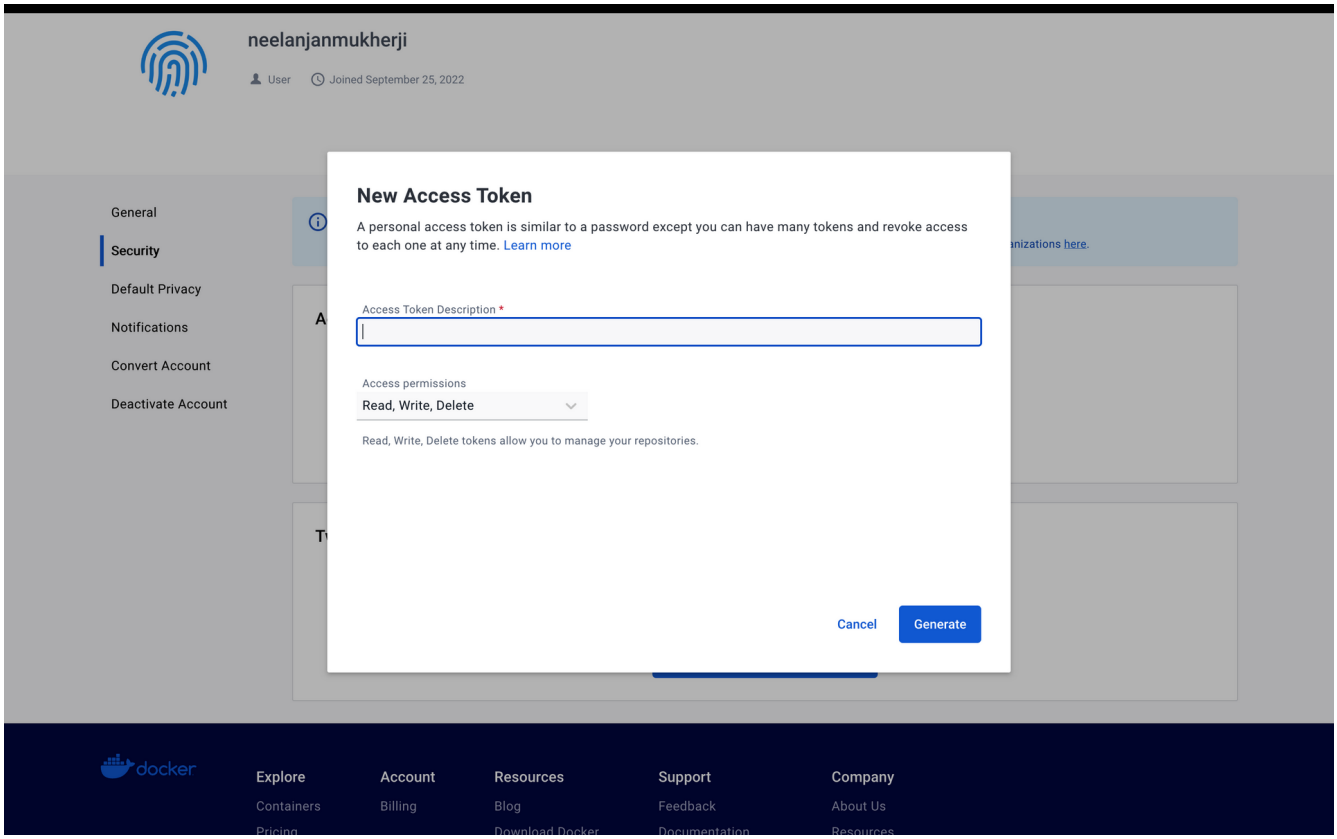


- Now click on the `New Access Token` button.

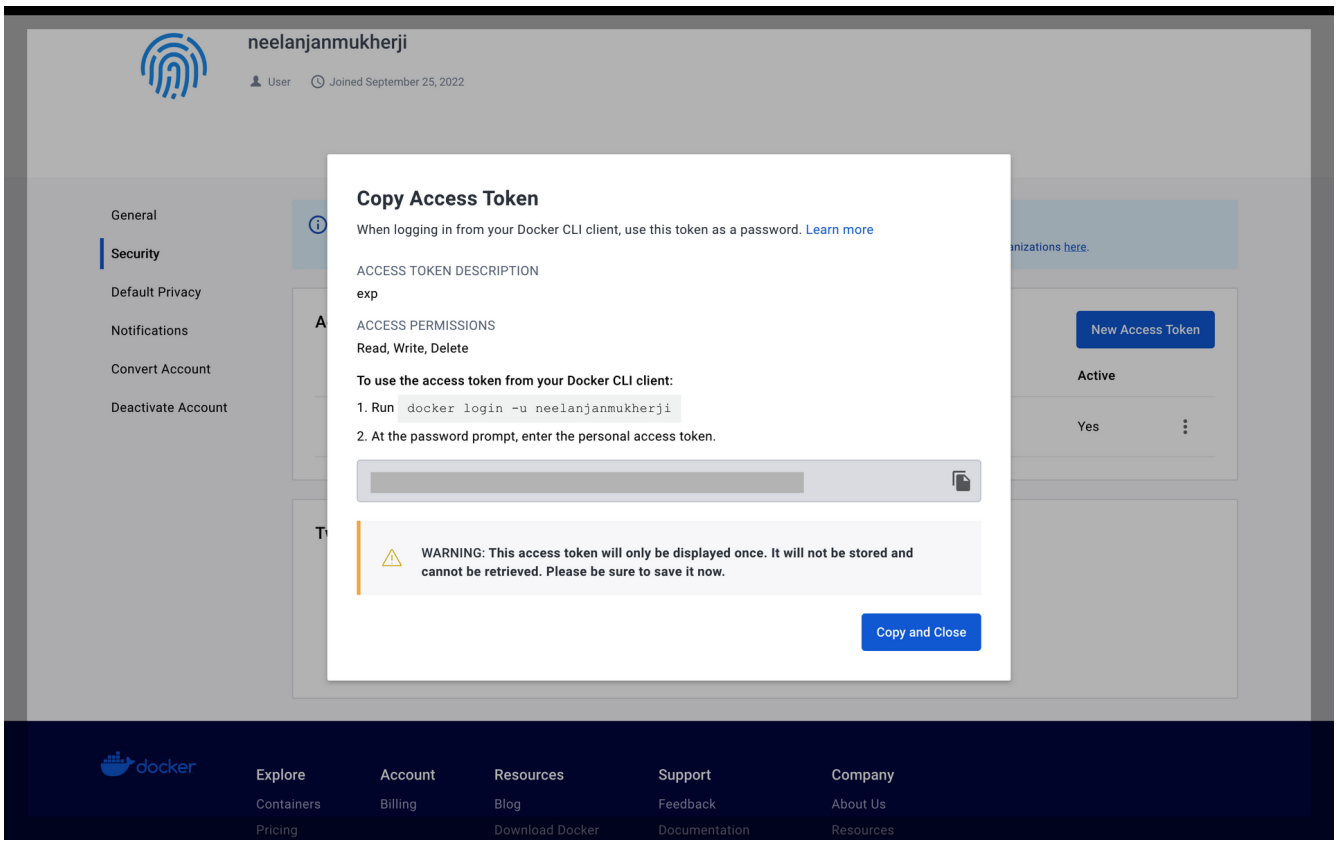


tion

The page should look something like this.



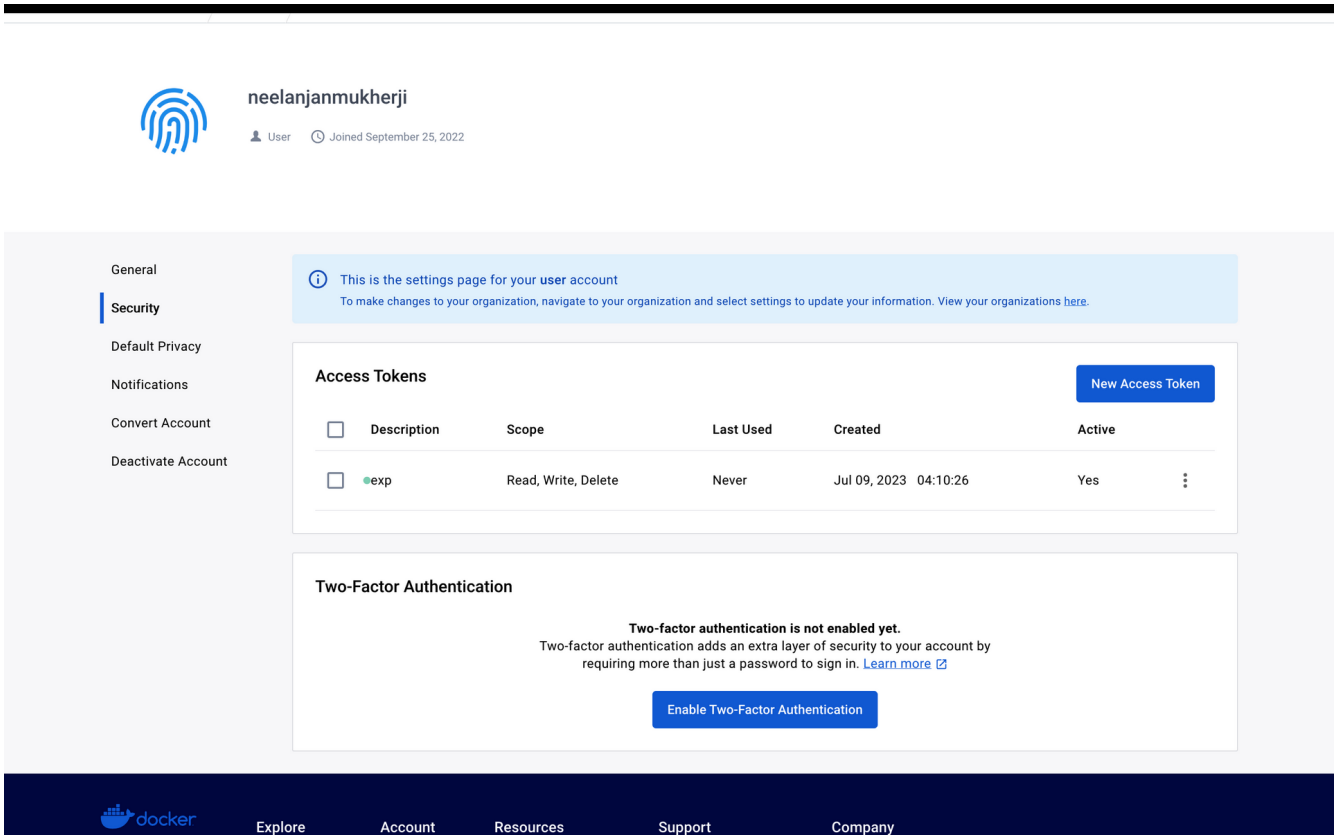
Add some description and click on the `Create a new workflow` button.



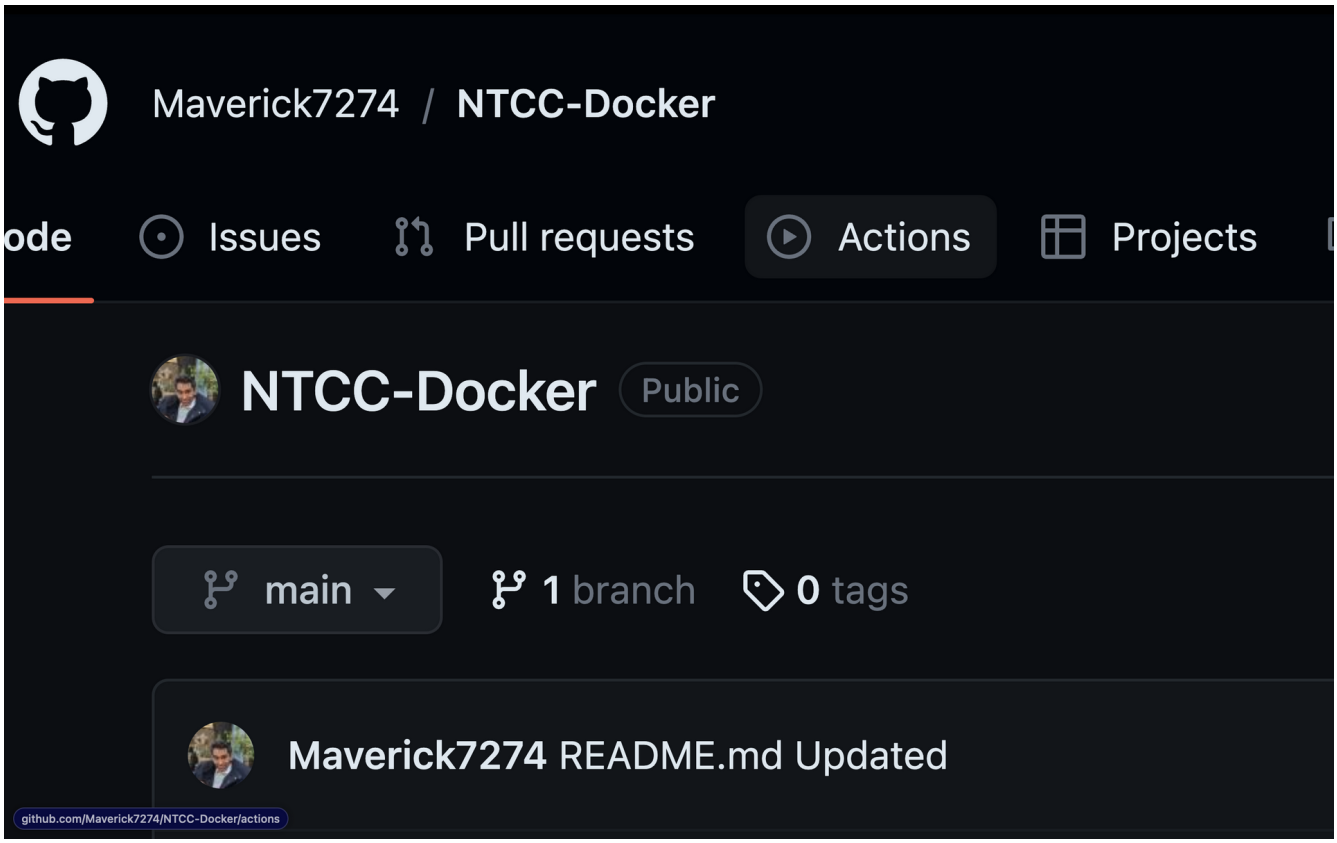
Copy the secret token and later we'll paste it in the `Secrets` tab of the GitHub repository.

Close the modal.

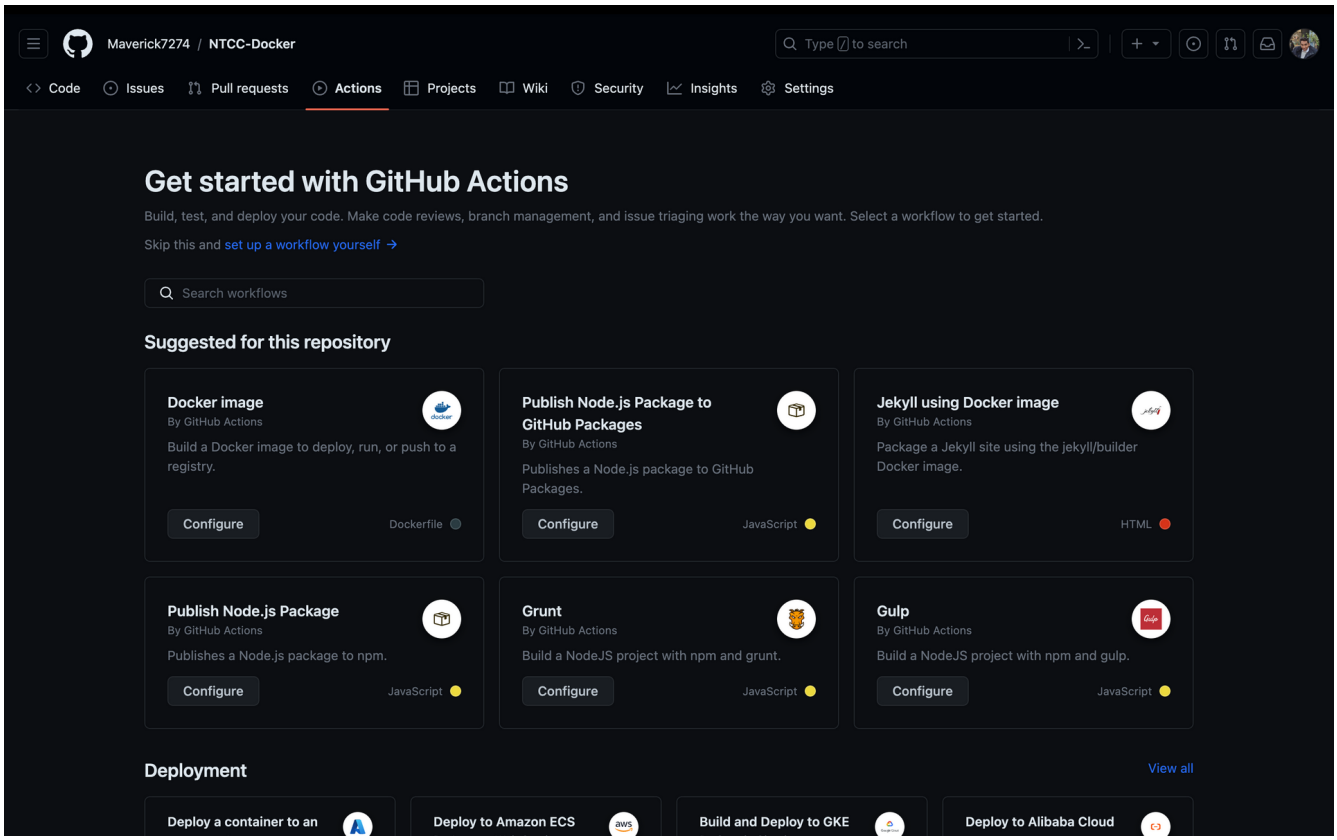
The page should look something like this.



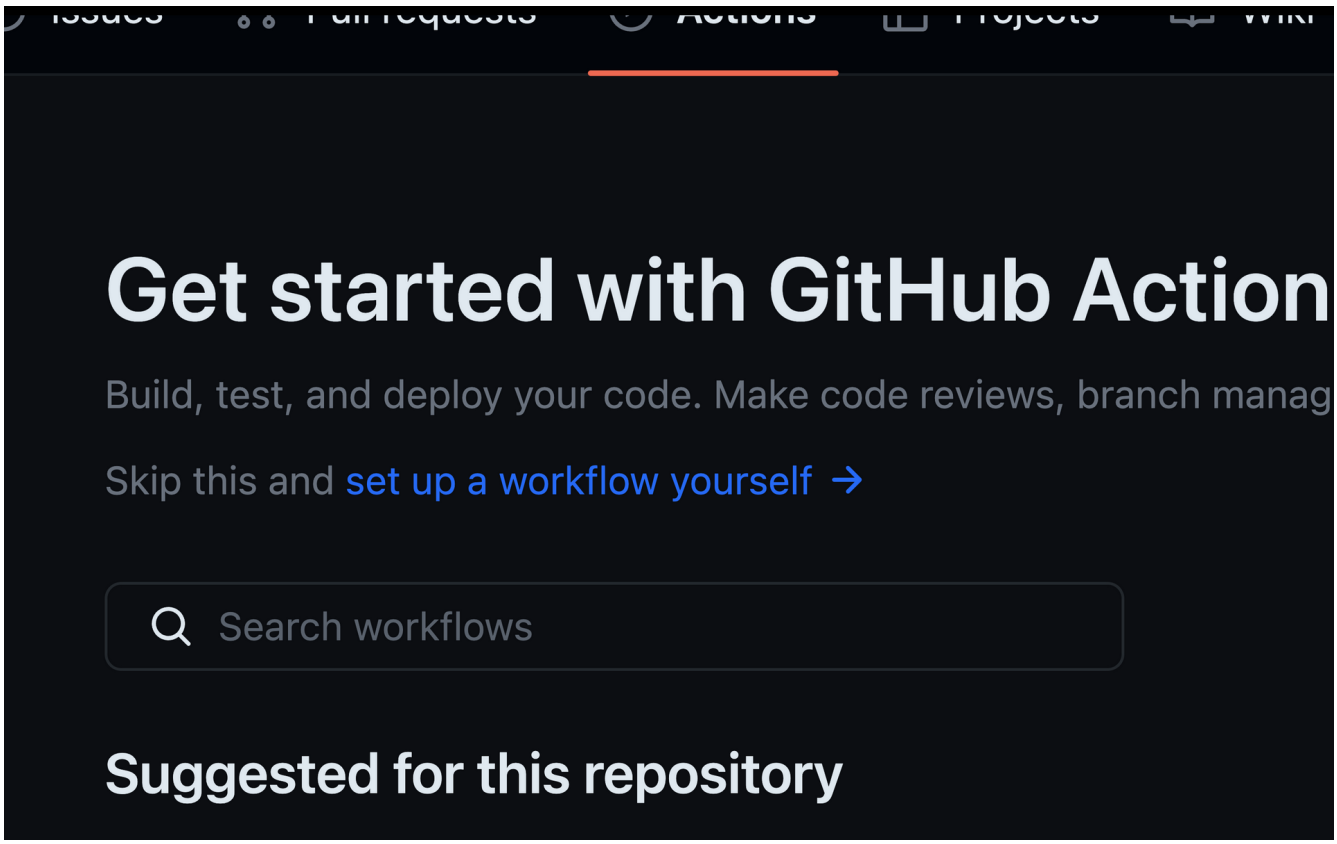
- Goto the GitHub repository.
- Goto the Actions tab in the GitHub repository.



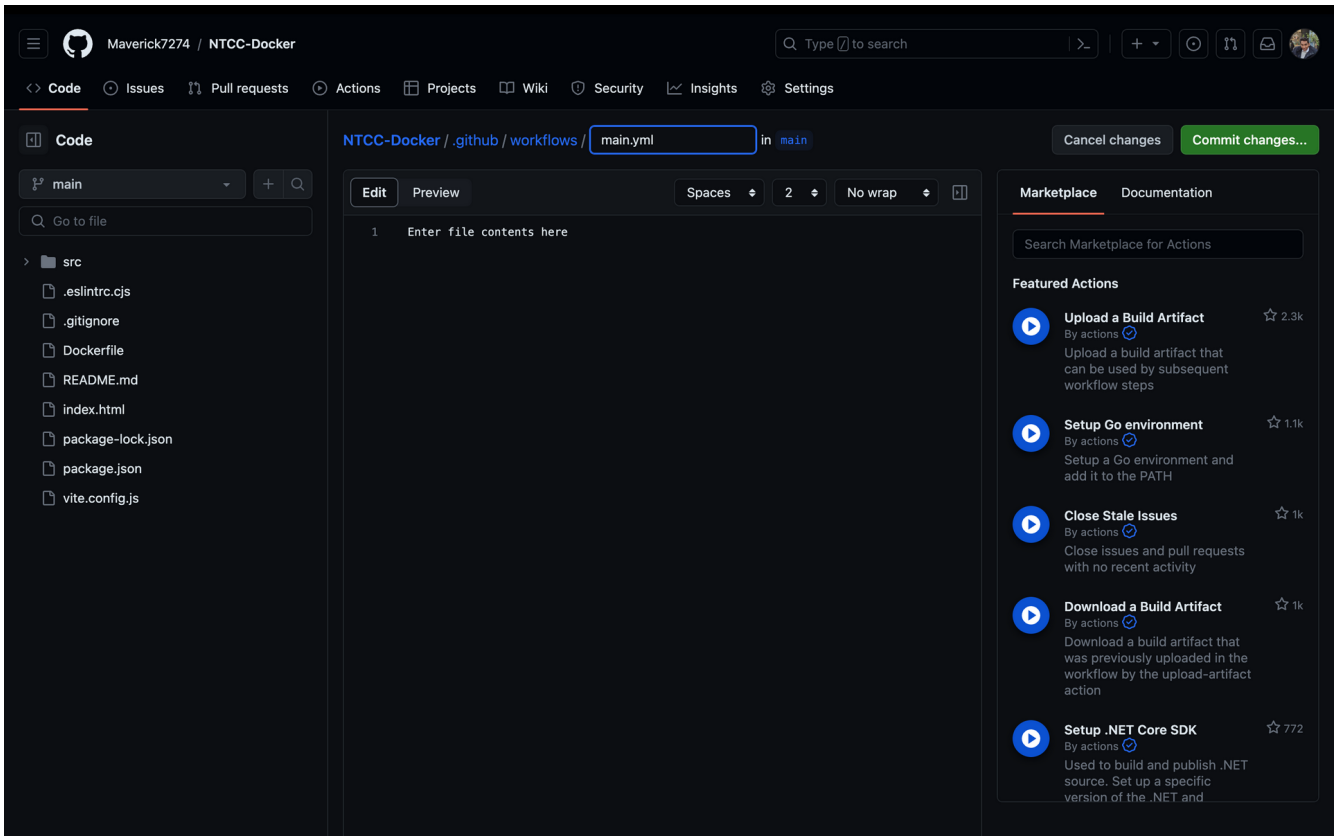
The page should look something like this.



- Now click on the `Set up a workflow yourself` button.



You'll be redirected to a page where you can write the workflow.



- Now we will write a bunch of Yaml Code which will build and deploy the React.js web application.

```yaml
name: CI

on:
push:
    branches: [ "main" ]
pull_request:
    branches: [ "main" ]
workflow_dispatch:

jobs:
Docker:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - uses: actions/setup-node@v3
        with:
        node-version: 14

    - name: setup git config
        run: |
        git config user.name "GitHub Actions Bot"
        git config user.email "<>"

    - name: Dependecies
```

```
        run: npm ci

    - name: Build
        run: npm run build

    - name: Save version
        id: version
        run: echo ::set-output name=tag::$(echo $(node -p -e
"require('./package.json').version"))

    - name: Increase version
        run: npm version patch

    - name: Push new version
        run: git push

    - name: Login to DockerHub Registry
        run: echo ${{ secrets.DOCKERHUB_PASSWORD }} | docker login -u ${{
secrets.DOCKERHUB_USERNAME }} --password-stdin

    - name: Build Docker image
        run: docker build . --file Dockerfile --tag
neelanjanmukherji/exp:${{steps.version.outputs.tag}}

    - name: Push to Docker Hub
        run: docker push
neelanjanmukherji/exp:${{steps.version.outputs.tag}}
```

Here we have used the `ubuntu-latest` image for running the GitHub Actions workflow.

We have also used the `actions/checkout@v3` action to checkout the code from the GitHub repository.

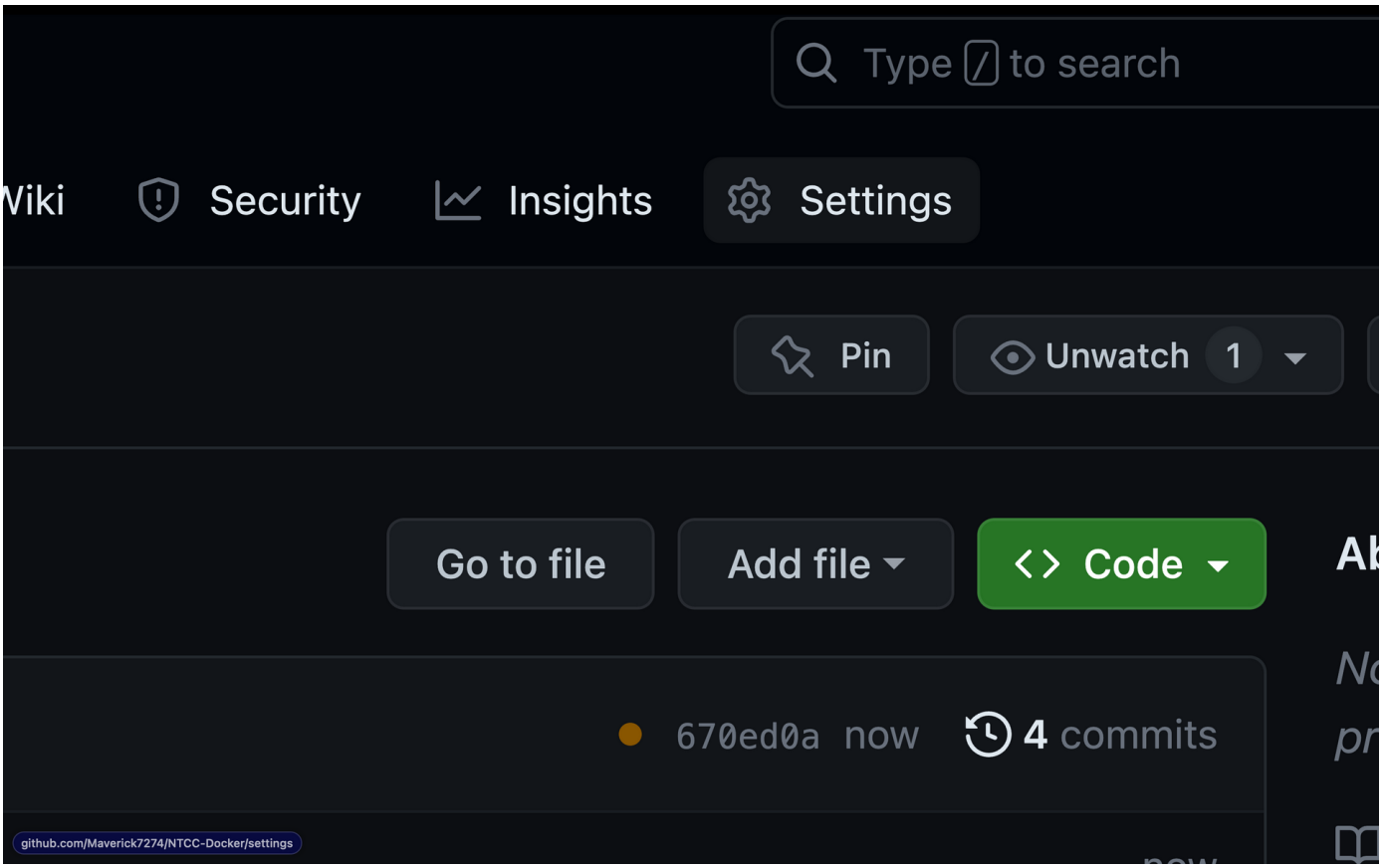
We have also used the `actions/setup-node@v3` action to setup the Node.js environment. We have also used the `npm ci` command to install the dependencies.

We have also used the `npm run build` command to build the React.js web application.

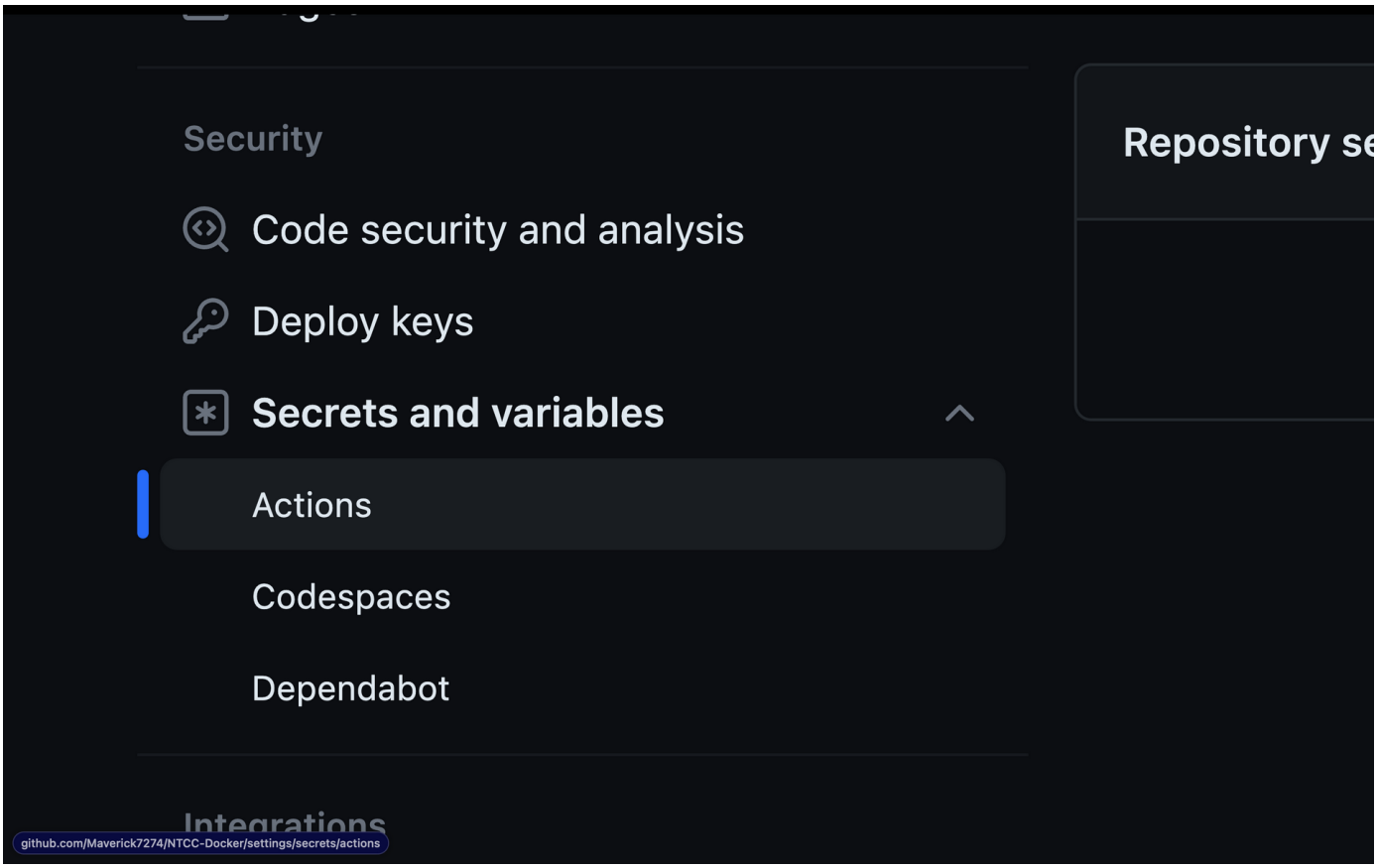
We have also used the `docker build` command to build the Docker image.

We have also used the `docker push` command to push the Docker image to the Docker Hub Registry.

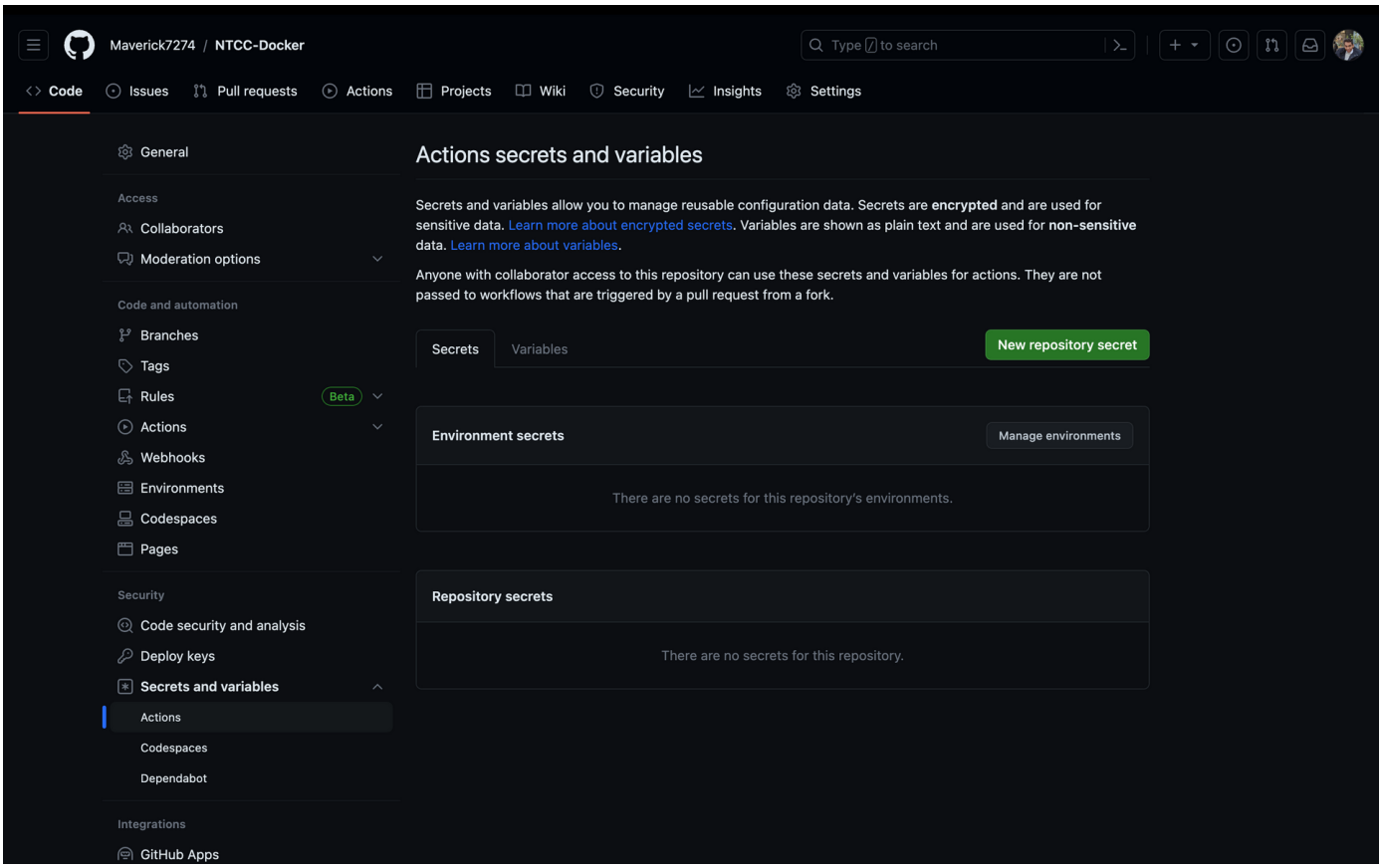
- After Completing with the Yaml Code click on the `Commit Changes` button.
- Note that we have used two dynamic variables `${{ secrets.DOCKERHUB_PASSWORD }}` & `${{ secrets.DOCKERHUB_USERNAME }}` in the Yaml Code. They are the secrets which we have created in the Docker Hub.
- To add a secret goto the Settings tab in the GitHub repository.



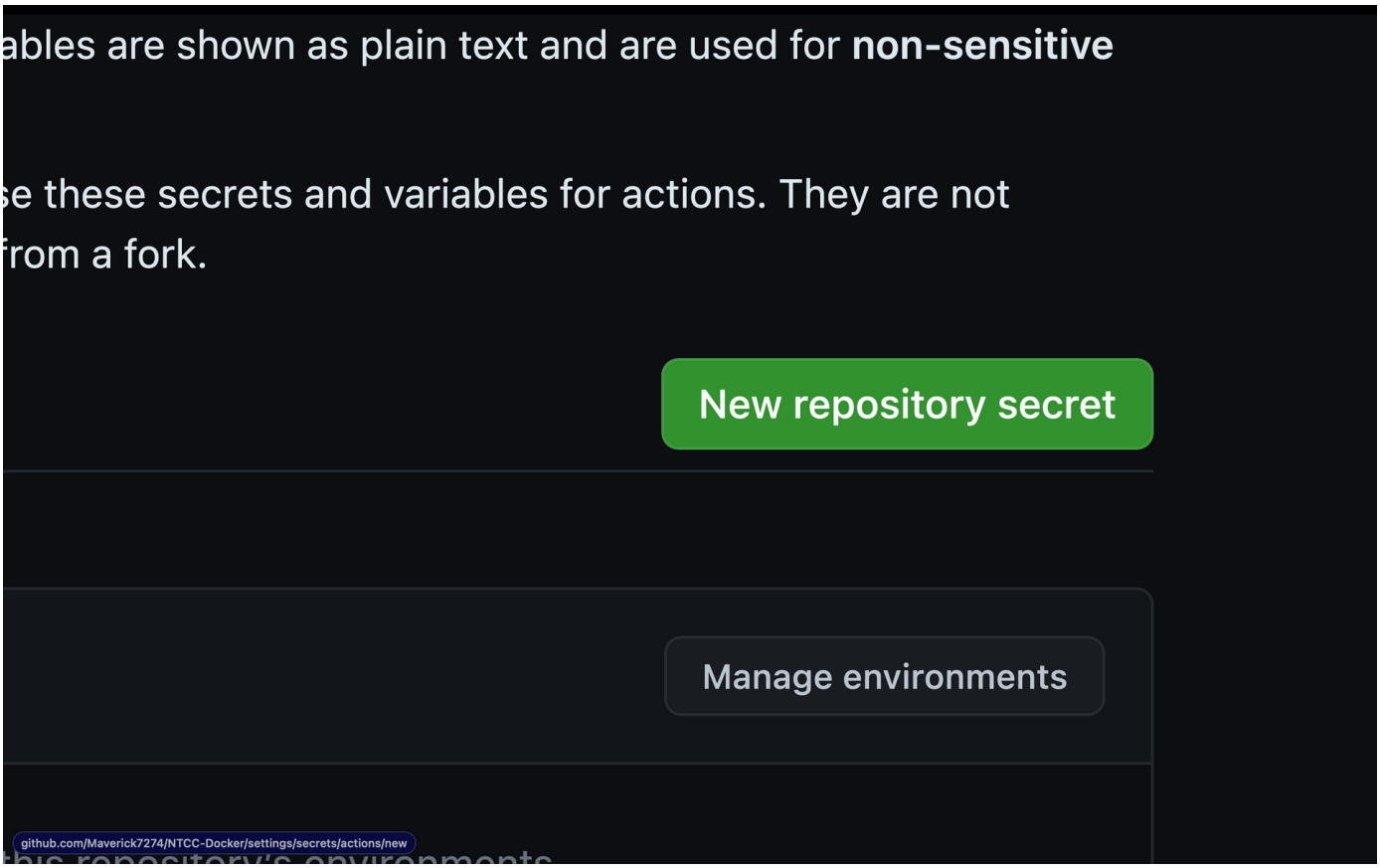
- The `Secret` button is a dropdown button.
- After that click on the `Action` button.



The page should look something like this.



Now click on the `New repository secret` button.

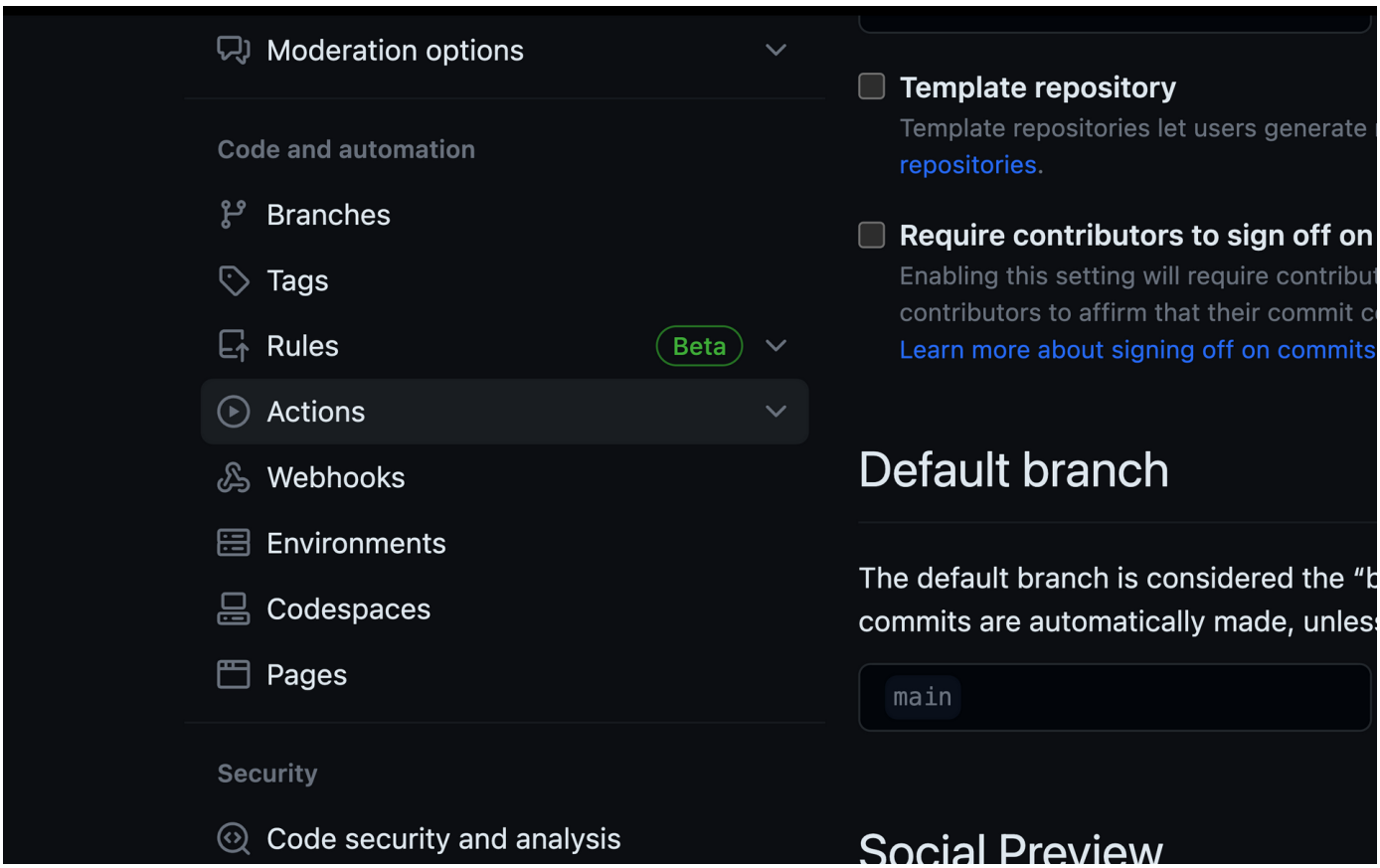


Now we will create two secrets `DOCKERHUB_USERNAME` and `DOCKERHUB_PASSWORD`.

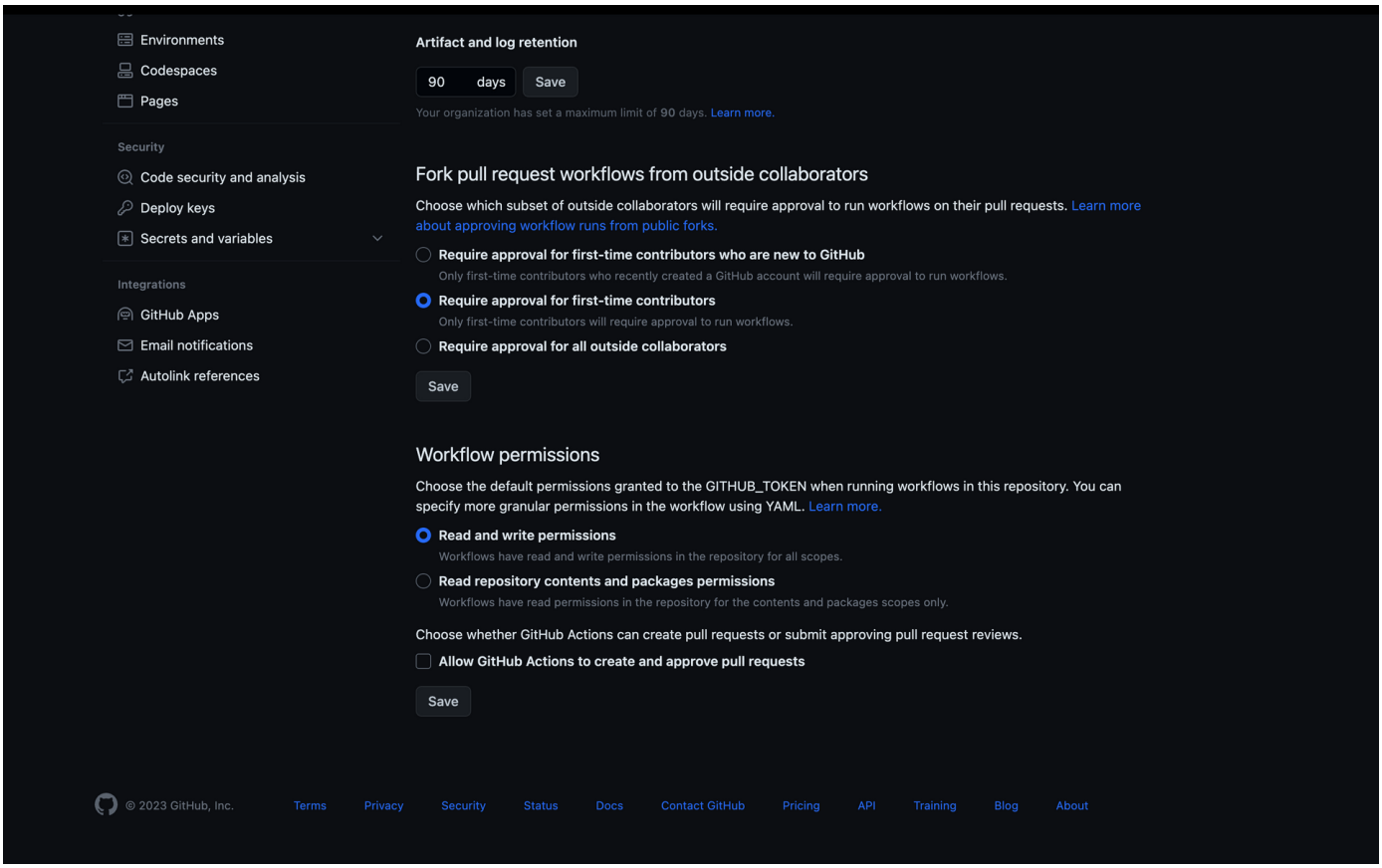
> Note: Here we have used the `neelanjanmukherji` as the Docker Hub username and `exp` as the Docker Hub repository name.

> Note: I have not shown the image due to security reasons.

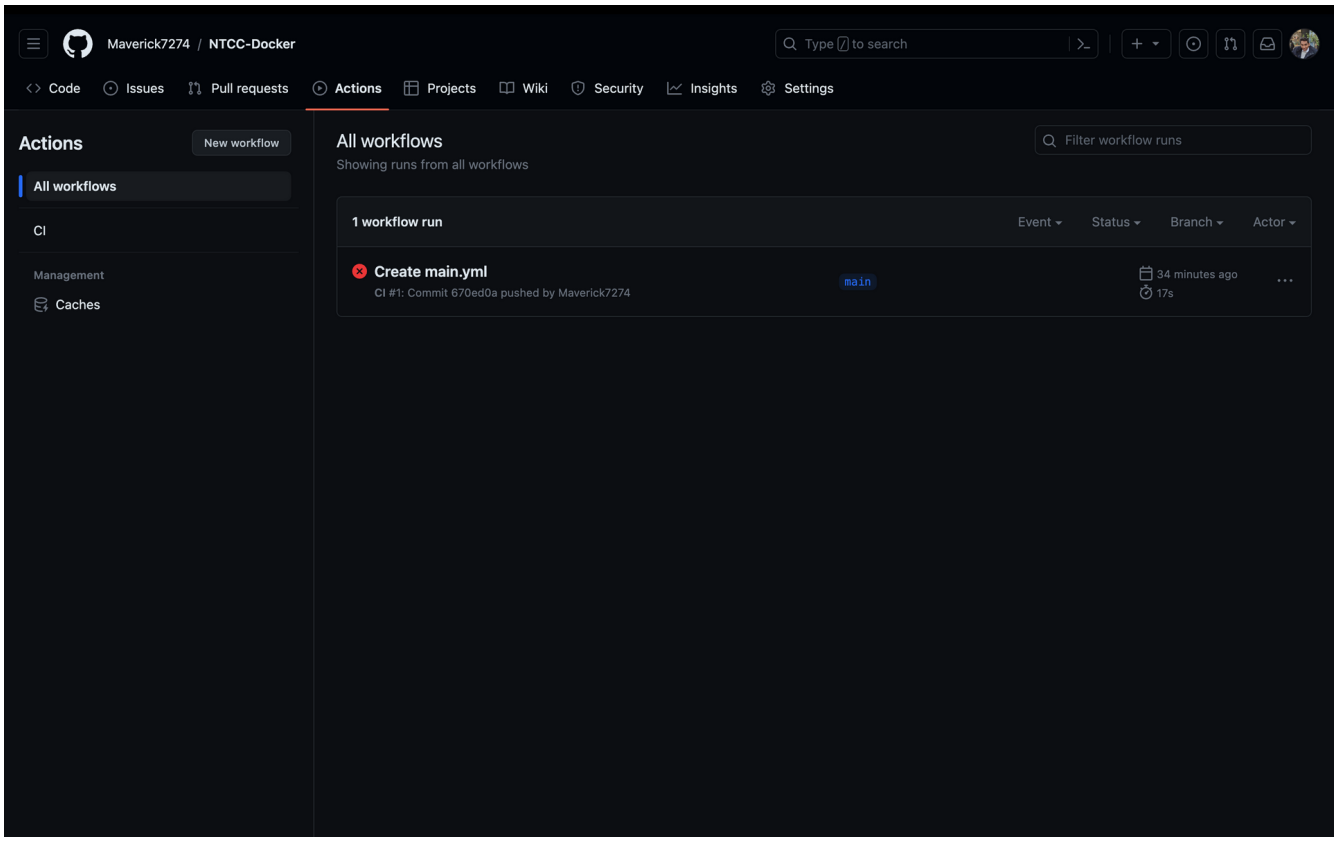
- Paste the `DOCKERHUB_USERNAME` in the `Name` field & then paste the Docker Hub username in the `Secret` field.
- Click on the `New Repository Secret` button.
- Paste the `DOCKERHUB_PASSWORD` in the `Name` field & then paste the Docker Hub password obtained before from the Docker Hub security settings in the `Secret` field.
- After adding the Secrets go back to settings and then goto the 'actions' tab and as it is a dropdown goto `general`.



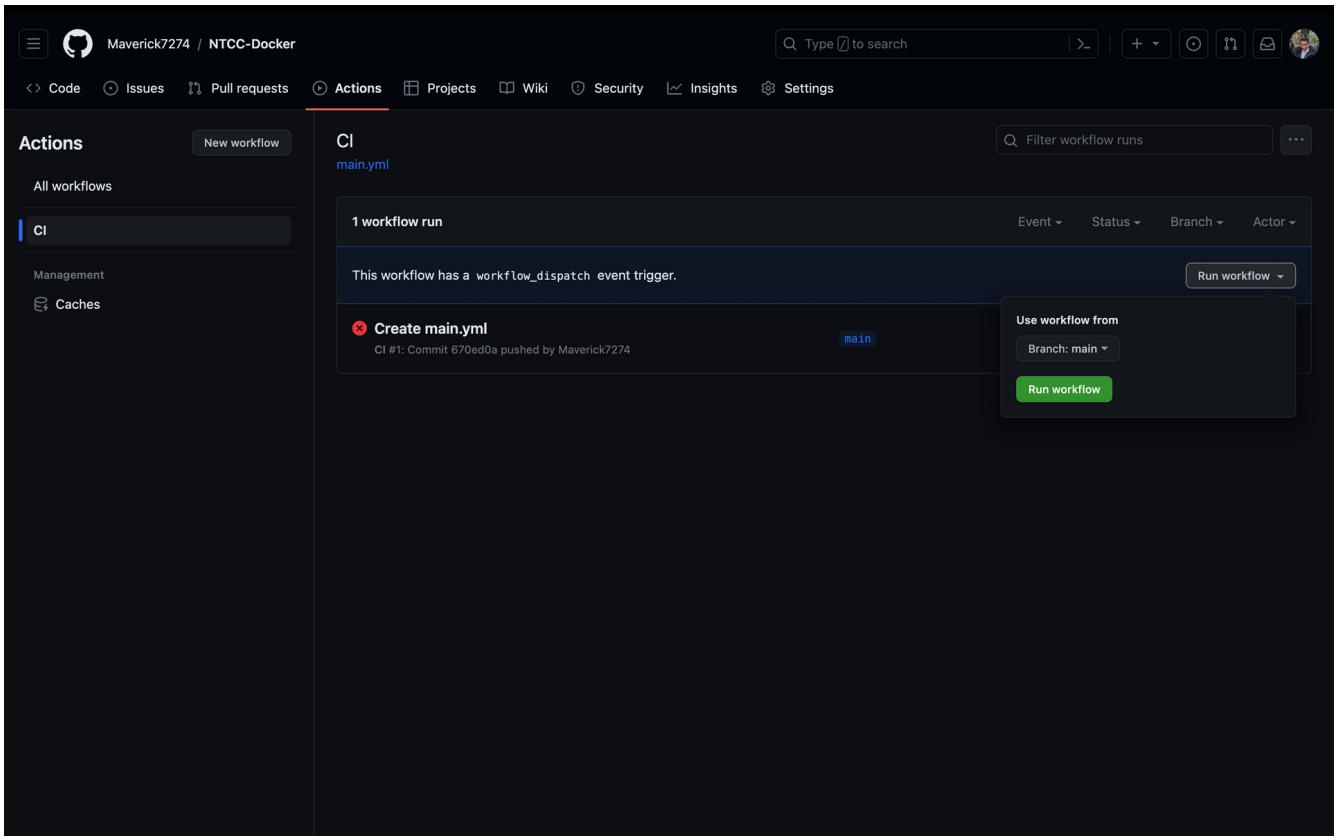
- Change the workflow permissions to `Read and write permissions` and save.



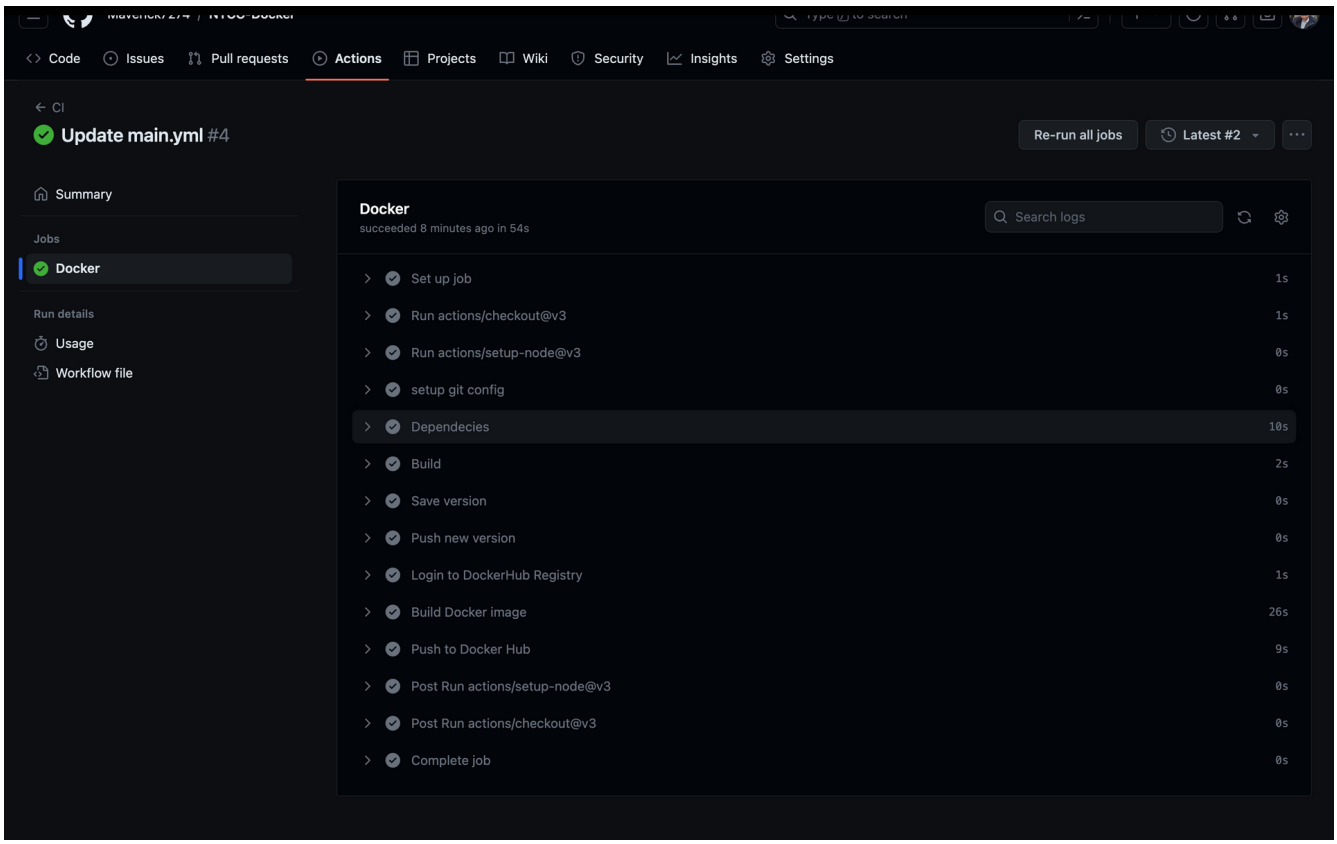
- Goto the Actions tab in the GitHub repository.



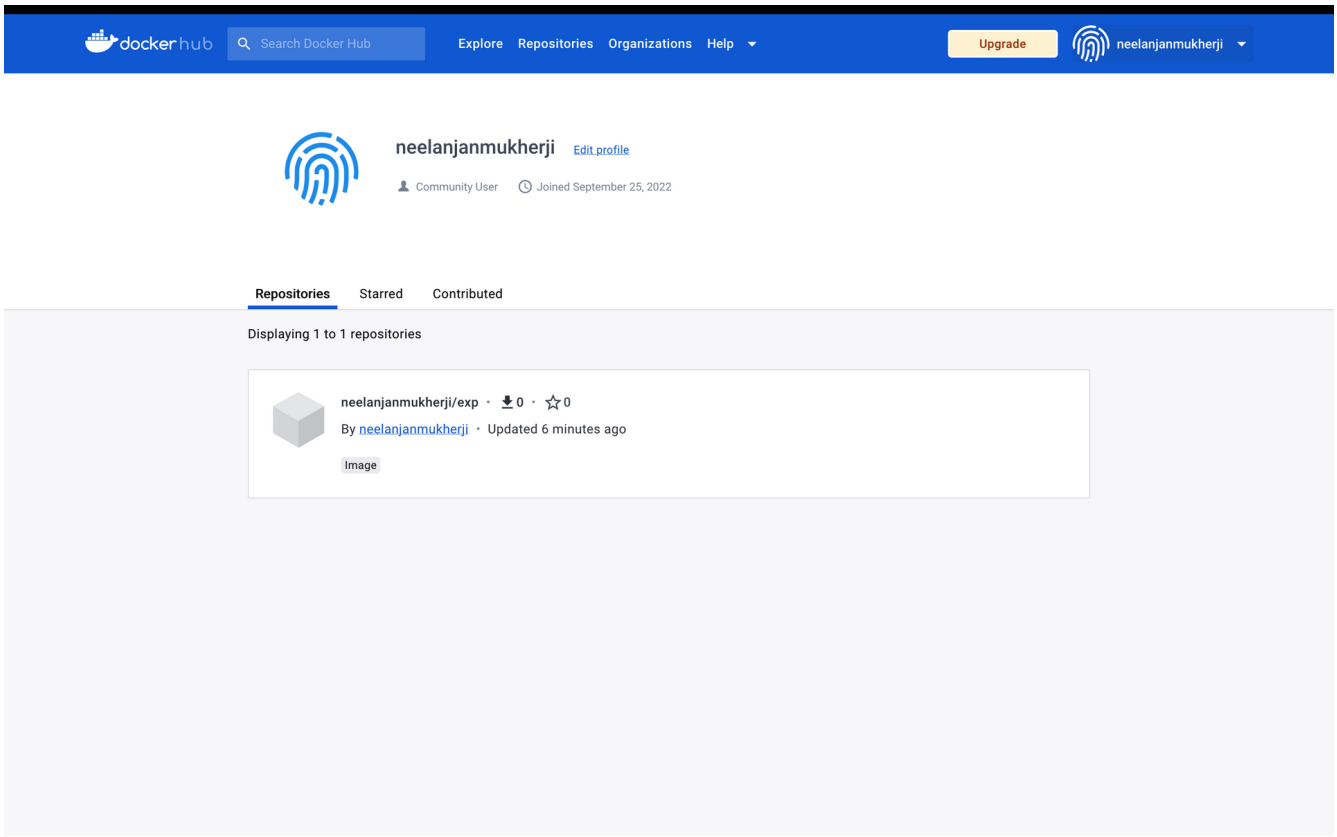
- Goto to CI workflow and click on the `Run workflow` button.



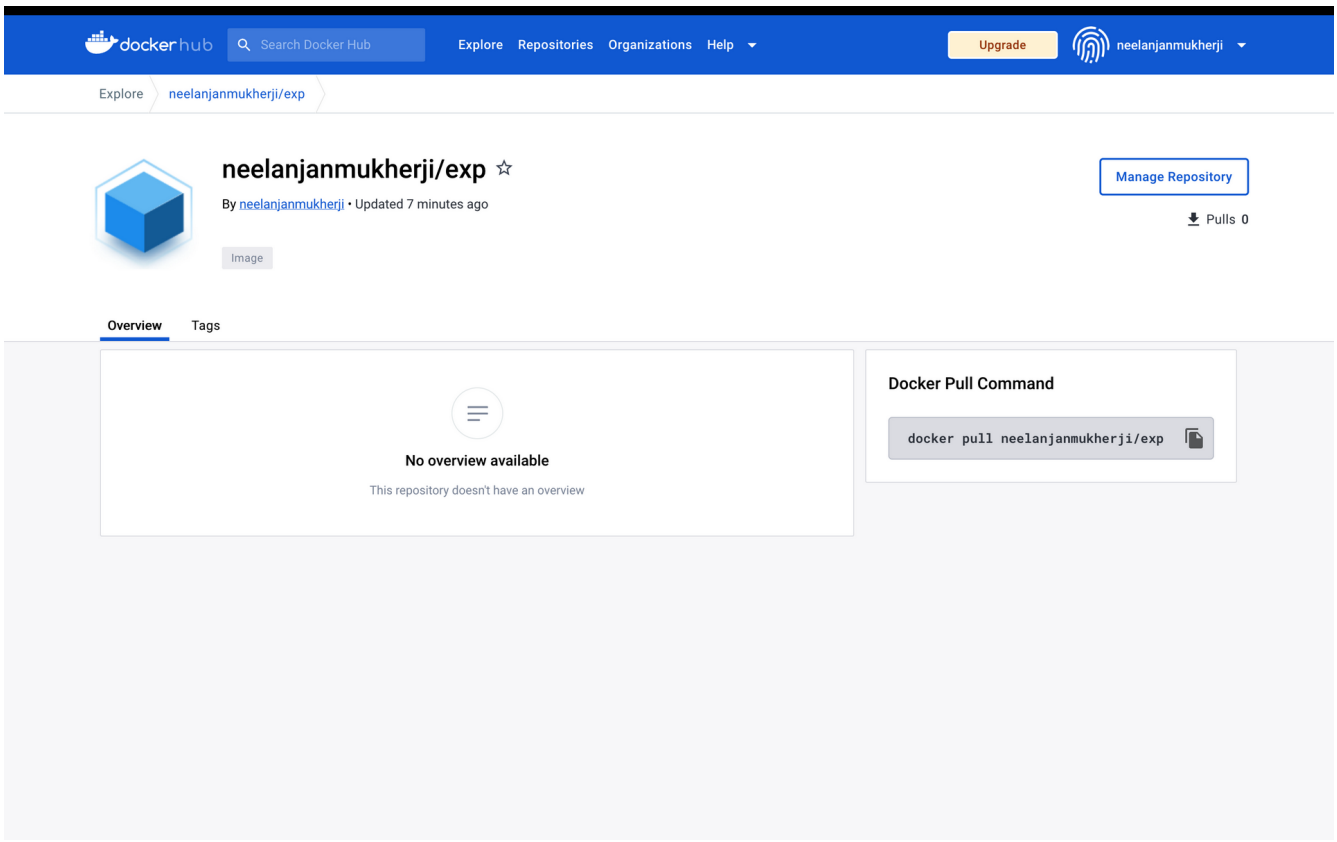
- All the steps should be successful.



- Goto the Docker Hub repository.
- You should see the Docker image.



- Here is the link to the Docker Hub repository.

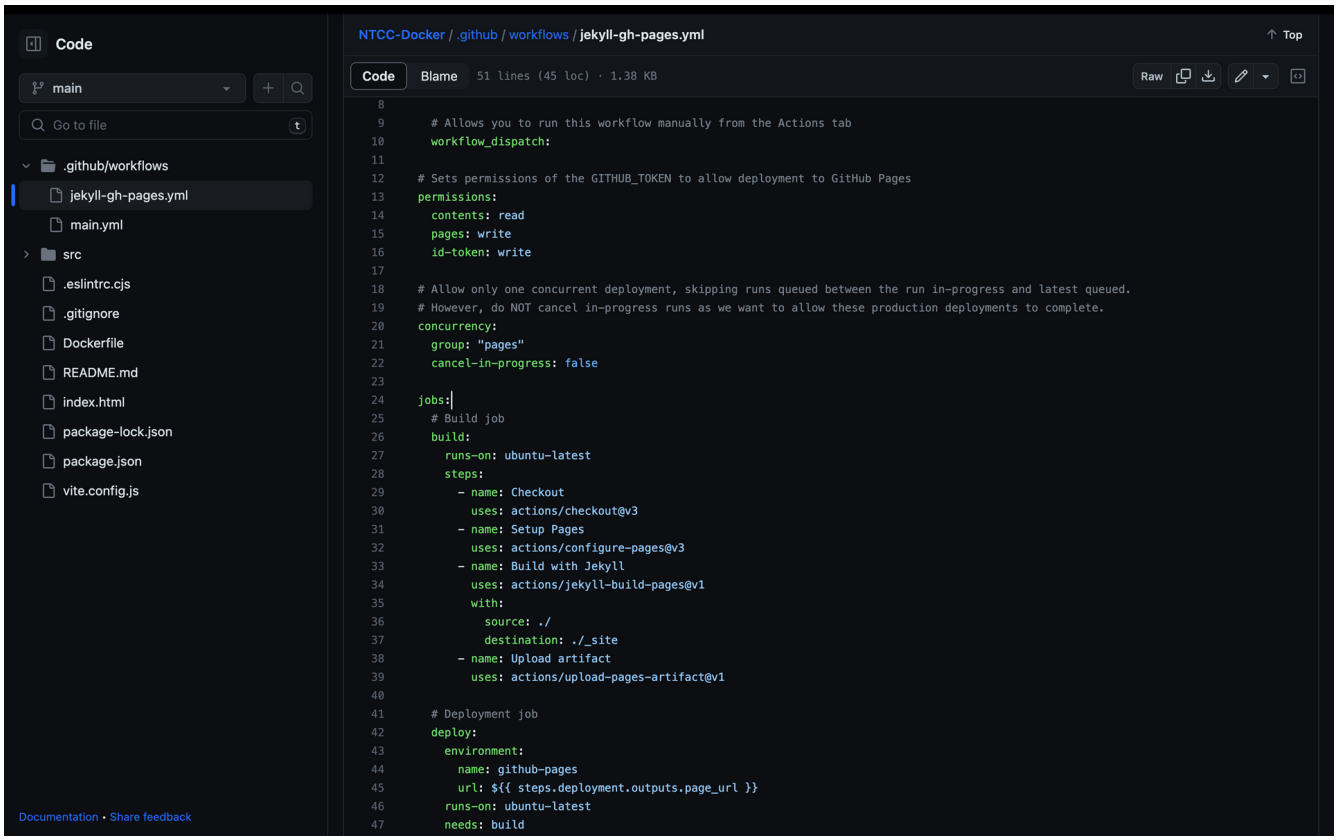


To deploy the React.js web application we will use the GitHub Pages.

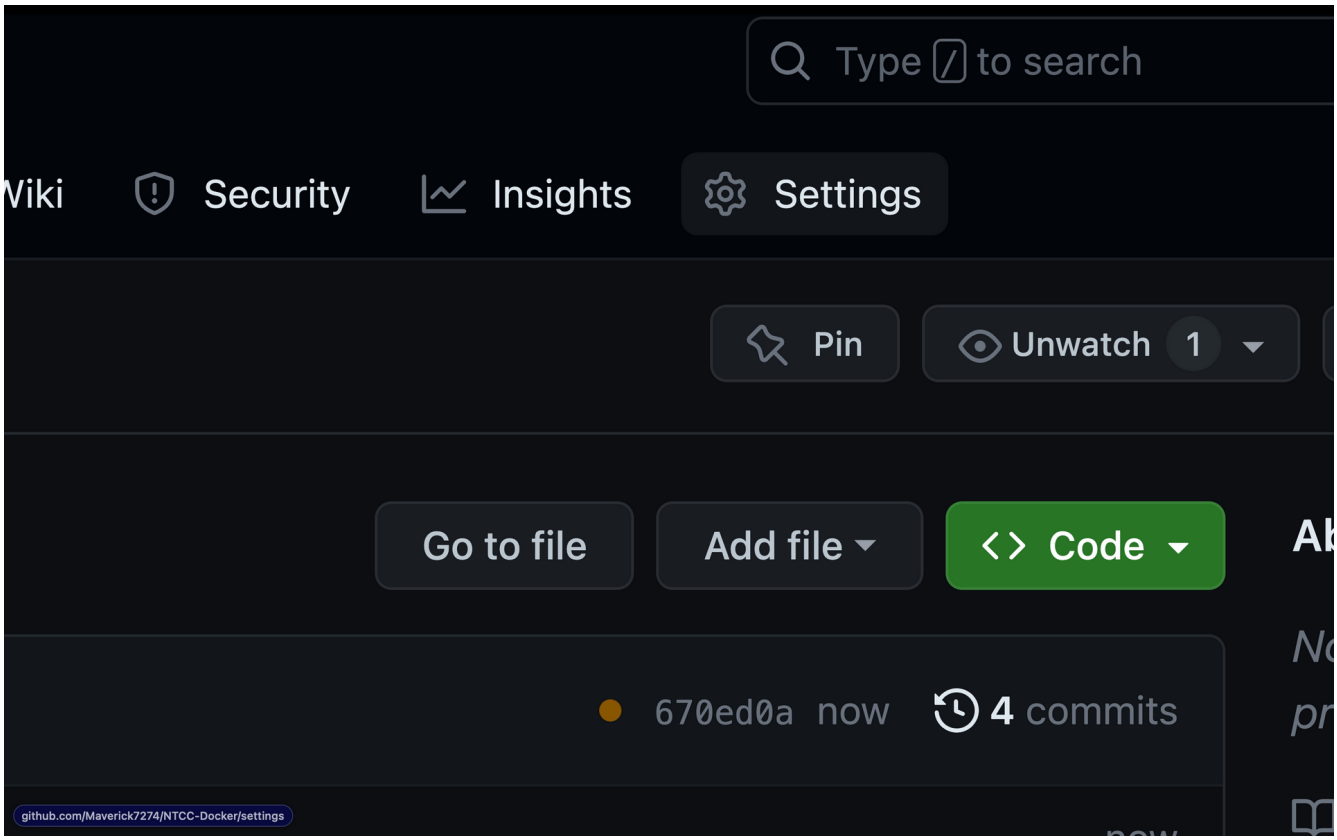
- Goto the GitHub repository.
- Goto the Actions tab in the GitHub repository.
- Add a Jekyll workflow.

```yaml
# Sample workflow for building and deploying a Jekyll site to GitHub Pages
name: Deploy Jekyll with GitHub Pages dependencies preinstalled

on:
# Runs on pushes targeting the default branch
push:
    branches: ["main"]

# Allows you to run this workflow manually from the Actions tab
workflow_dispatch:

# Sets permissions of the GITHUB_TOKEN to allow deployment to GitHub Pages
permissions:
contents: read
pages: write
id-token: write

# Allow only one concurrent deployment, skipping runs queued between the run in-progress and latest queued.
# However, do NOT cancel in-progress runs as we want to allow these production deployments to complete.
concurrency:
group: "pages"
cancel-in-progress: false

jobs:
# Build job
build:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout
        uses: actions/checkout@v3
    - name: Setup Pages
        uses: actions/configure-pages@v3
    - name: Build with Jekyll
        uses: actions/jekyll-build-pages@v1
        with:
        source: ./
        destination: ./_site
    - name: Upload artifact
        uses: actions/upload-pages-artifact@v1
```

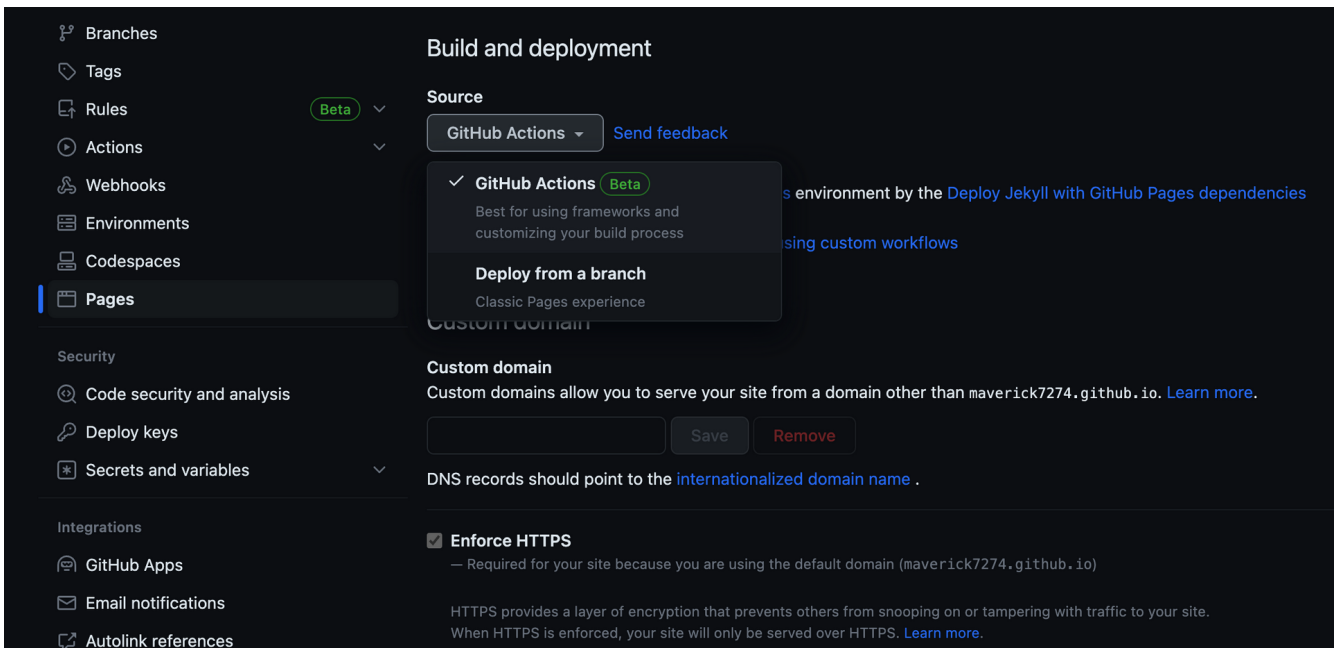
```yaml
# Deployment job
deploy:
    environment:
    name: github-pages
    url: ${{ steps.deployment.outputs.page_url }}
    runs-on: ubuntu-latest
    needs: build
    steps:
    - name: Deploy to GitHub Pages
        id: deployment
        uses: actions/deploy-pages@v2
```



- Goto the Settings tab in the GitHub repository.



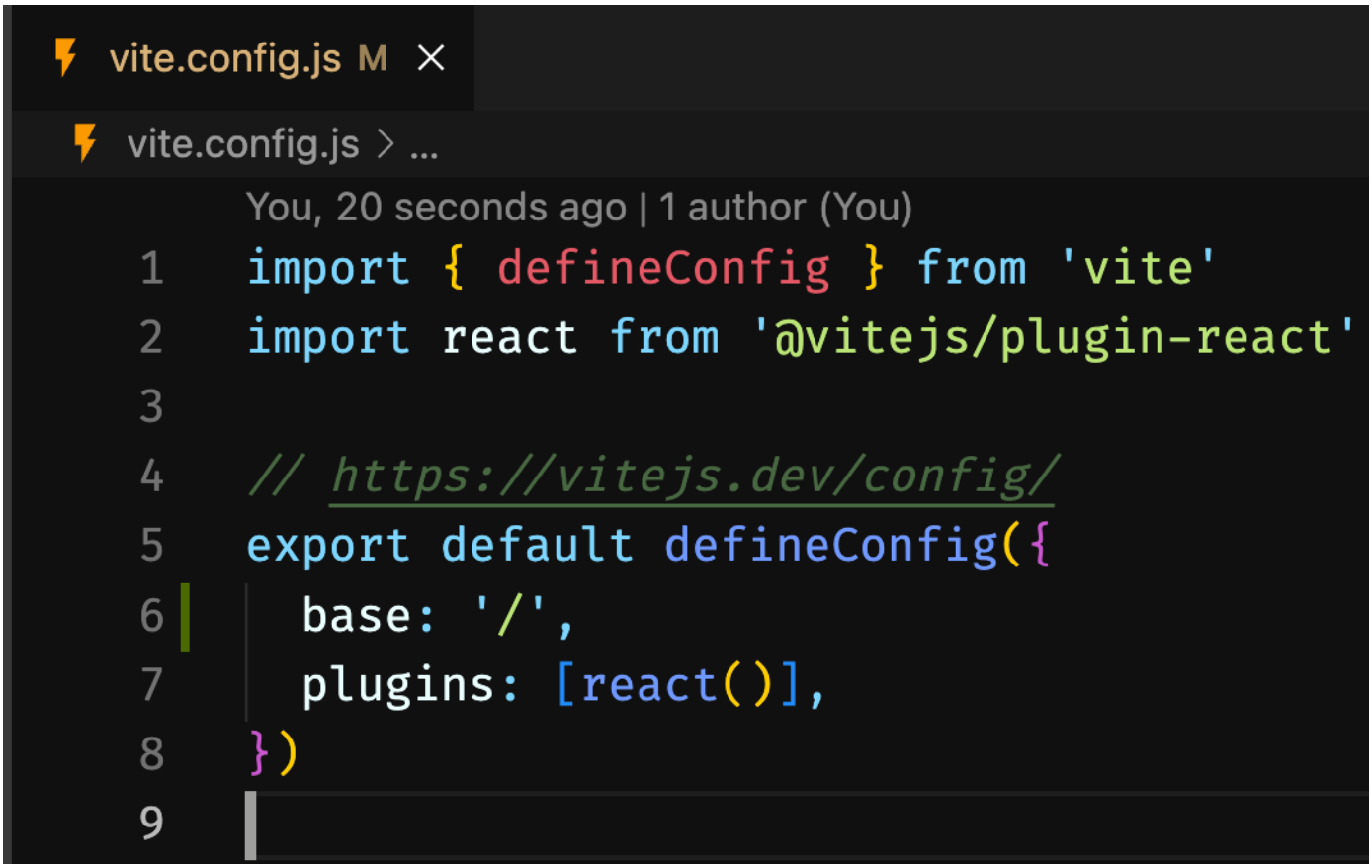
- Click on the Pages section.
- Select the Source as `GitHub Actions`.



> Note: If you used vite to create the React.js web application then you have to change the `base` in the `vite.config.js` file to `./`.

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'

// https://vitejs.dev/config/
export default defineConfig({
  base: '/',
```

```
  plugins: [react()],
})
```



- After that you'll see a link to the deployed React.js web application.
- You can find the app running via github pages here
- And there you go you have successfully deployed a React.js web application using Docker, GitHub Actions & GitHub Pages.