Table of contents:

Docusaurus was designed from the ground up to be easily installed and used to get your website up and running quickly.

> **Important Note:** we highly encourage you to use Docusaurus 2 instead.

Installing Docusaurus

We have created a helpful script that will get all of the infrastructure set up for you:

1. Ensure you have the latest version of Node installed. We also recommend you install Yarn as well.

   > You have to be on Node >= 10.9.0 and Yarn >= 1.5.

2. Create a project, if none exists, and change your directory to this project's root.

   You will be creating the docs in this directory. The root directory may contain other files. The Docusaurus installation script will create two new directories: `docs` and `website`.

   > Commonly, either an existing or newly created GitHub project will be the location for your Docusaurus site, but that is not mandatory to use Docusaurus.

3. Run the Docusaurus installation script: `npx docusaurus-init`.

   > If you don't have Node 8.2+ or if you prefer to install Docusaurus globally, run `yarn global add docusaurus-init` or `npm install --global docusaurus-init`. After that, run `docusaurus-init`.

Verifying Installation

Along with previously existing files and directories, your root directory will now contain a structure similar to:

```
root-directory
├── Dockerfile
├── README.md
├── docker-compose.yml
├── docs
|   ├── doc1.md
|   ├── doc2.md
|   ├── doc3.md
|   ├── exampledoc4.md
|   └── exampledoc5.md
└── website
    ├── blog
    |   ├── 2016-03-11-blog-post.md
    |   ├── 2017-04-10-blog-post-two.md
    |   ├── 2017-09-25-testing-rss.md
    |   ├── 2017-09-26-adding-rss.md
    |   └── 2017-10-24-new-version-1.0.0.md
    ├── core
    |   └── Footer.js
    ├── package.json
    ├── pages
    ├── sidebars.json
    ├── siteConfig.js
    └── static
```

> This installation creates some Docker files that are not necessary to run docusaurus. They may be deleted without issue in the interest of saving space. For more information on Docker, please see the Docker documentation.

Running the example website

After running the Docusaurus initialization script, `docusaurus-init` as described in the Installation section, you will have a runnable, example website to use as your site's base. To run:

1. `cd website`
2. From within the `website` directory, run the local web server using `yarn start` or `npm start`.
3. Load the example site at http://localhost:3000 if it did not already open automatically. If port 3000 has already been taken, another port will be used. Look at the console messages to see which.

   You should see the example site loaded in your web browser. There's also a LiveReload server running and any changes made to the docs and files in the `website` directory will cause the page to refresh. A randomly generated primary and secondary theme color will be picked for you.

Test Site

A website for testing

TRY IT OUT EXAMPLE LINK EXAMPLE LINK 2

Feature One

This is the content of my feature

Feature Two

The content of my second feature

Feature Callout

These are features of this project

Launching the server behind a proxy

If you are behind a corporate proxy, you need to disable it for the development server requests. It can be done using the `NO_PROXY` environment variable.

```
SET NO_PROXY=localhost
yarn start (or npm run start)
```

Updating Your Docusaurus Version

At any time after Docusaurus is installed, you can check your current version of Docusaurus by going into the `website` directory and typing `yarn outdated docusaurus` or `npm outdated docusaurus`.

You will see something like this:

```
$ yarn outdated
Using globally installed version of Yarn
yarn outdated v1.5.1
warning package.json: No license field
warning No license field
info Color legend :
 "<red>"    : Major Update backward-incompatible updates
 "<yellow>" : Minor Update backward-compatible features
 "<green>"  : Patch Update backward-compatible bug fixes
Package    Current Wanted Latest Package Type    URL
docusaurus 1.0.9   1.2.0  1.2.0  devDependencies https://github.com/facebook/docusaurus#r
✨  Done in 0.41s.
```

If there is no noticeable version output from the `outdated` commands, then you are up-to-date.

You can update to the latest version of Docusaurus by:

```
yarn upgrade docusaurus --latest
```

or

```
npm update docusaurus
```

If you are finding that you are getting errors after your upgrade, try to either clear your Babel cache (usually it's in a temporary directory or run the Docusaurus server (e.g., `yarn start`) with the `BABEL_DISABLE_CACHE=1` environment configuration.

After installing Docusaurus, you now have a skeleton to work from for your specific website. The following discusses the rest of the Docusaurus structure in order for you to prepare your site.

Directory Structure

As shown after you installed Docusaurus, the initialization script created a directory structure similar to:

```
root-directory
├── .gitignore
├── docs
|   ├── doc1.md
|   ├── doc2.md
|   ├── doc3.md
|   ├── exampledoc4.md
|   └── exampledoc5.md
└── website
    ├── blog
    |   ├── 2016-03-11-blog-post.md
    |   ├── 2017-04-10-blog-post-two.md
    |   ├── 2017-09-25-testing-rss.md
    |   ├── 2017-09-26-adding-rss.md
    |   └── 2017-10-24-new-version-1.0.0.md
    ├── core
    |   └── Footer.js
    ├── package.json
    ├── pages
    ├── sidebars.json
    ├── siteConfig.js
    └── static
```

Directory Descriptions

- **Documentation Source Files**: The `docs` directory contains example documentation files written in Markdown.
- **Blog**: The `website/blog` directory contains examples of blog posts written in markdown.
- **Pages**: The `website/pages` directory contains example top-level pages for the site.
- **Static files and images**: The `website/static` directory contains static assets used by the example site.

Key Files

- **Footer**: The `website/core/Footer.js` file is a React component that acts as the footer for the site generated by Docusaurus and should be customized by the user.
- **Configuration file**: The `website/siteConfig.js` file is the main configuration file used by Docusaurus.
- **Sidebars**: The `sidebars.json` file contains the structure and order of the documentation files.
- **.gitignore**: The `.gitignore` file lists the necessary ignore files for the generated site so that they do not get added to the git repo.

Preparation Notes

You will need to keep the `website/siteConfig.js` and `website/core/Footer.js` files but may edit them as you wish. The value of the `customDocsPath` key in `website/siteConfig.js` can be modified if you wish to use a different directory name or path. The `website` directory can also be renamed to anything you want it to be.

However, you should keep the `website/pages` and `website/static` directories. You may change the content inside them as you wish. At the bare minimum, you should have an `en/index.js` or `en/index.html` file inside `website/pages` and an image to use as your header icon inside `website/static`.

If your directory does not yet have a `.gitignore`, we generate it with the necessary ignored files listed. As a general rule, you should ignore all `node_modules`, build files, system files (`.DS_Store`), logs, etc. Here is a more comprehensive list of what is normally ignored for Node.js projects.

Docusaurus was created to hopefully make it super simple for you to create a site and documentation for your open source project.

After installation and preparation, much of the work to create a basic site for your docs is already complete.

Site Structure

Your site structure looks like the following:

```
root-directory
├── docs
└── website
    ├── blog
    ├── core
    │   └── Footer.js
    ├── package.json
    ├── pages
    ├── sidebars.json
    ├── siteConfig.js
    └── static
```

> This assumes that you removed the example `.md` files that were installed with the initialization script.

All of your documentation files should be placed inside the `docs` directory as markdown `.md` files. Any blog posts should be inside the `blog` directory.

> The blog posts must be formatted as `YYYY-MM-DD-your-file-name.md`

Create Your Basic Site

To create a fully functional site, you only need to do a few steps:

1. Add your documentation to the `/docs` directory as `.md` files, ensuring you have the proper header in each file. One example header would be the following, where `id` is the link name (e.g., `docs/intro.html`) and the `title` is the webpage's title.

```
---
id: intro
title: Getting Started
---
My new content here..
```

2. Add zero or more docs to the `sidebars.json` file so that your documentation is rendered in a sidebar if you choose them to be.

> If you do not add your documentation to the `sidebars.json` file, the docs will be rendered, but they can only be linked to from other documentation and visited with the known URL.

3. Modify the `website/siteConfig.js` file to configure your site, following the comments included in the docs and the `website/siteConfig.js` to guide you.
4. Create any custom pages and/or customize the `website/core/Footer.js` file that provides the footer for your site.
5. Place assets, such as images, in the `website/static/` directory.
6. Run the site to see the results of your changes.

```
cd website
yarn run start # or `npm run start`
# Navigate to http://localhost:3000
```

Special Customization

Docs Landing Page

If you prefer to have your landing page be straight to your documentation, you can do this through a redirect.

1. Remove the `index.js` file from the `website/pages` directory, if it exists.
2. Add a custom static `index.html` page in the `website/static` directory with the following contents:

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="UTF-8" />
    <meta
      http-equiv="refresh"
      content="0; url=docs/id-of-doc-to-land-on.html"
    />
    <script type="text/javascript">
      window.location.href = 'docs/id-of-doc-to-land-on.html';
    </script>
    <title>Your Site Title Here</title>
  </head>
  <body>
    If you are not redirected automatically, follow this
    <a href="docs/id-of-doc-to-land-on.html">link</a>.
  </body>
</html>
```

> You will get the `id` of the document to land on the `.md` metadata of that doc page.

Blog Only

You can also use Docusaurus to host your blog only.

You should now have a site up and running locally. Once you have customized it to your liking, it's time to publish it. Docusaurus generates a static HTML website that is ready to be served by your favorite web server or online hosting solution.

Building Static HTML Pages

To create a static build of your website, run the following script from the `website` directory:

```
yarn run build # or `npm run build`
```

This will generate a `build` directory inside the `website` directory containing the `.html` files from all of your docs and other pages included in `pages`.

Hosting Static HTML Pages

At this point, you can grab all of the files inside the `website/build` directory and copy them over to your favorite web server's `html` directory.

> For example, both Apache and Nginx serve content from `/var/www/html` by default. That said, choosing a web server or provider is outside the scope of Docusaurus.

> When serving the site from your own web server, ensure the web server is serving the asset files with the proper HTTP headers. CSS files should be served with the `content-type` header of `text/css`. In the case of Nginx, this would mean setting `include /etc/nginx/mime.types;` in your `nginx.conf` file. See this issue for more info.

Hosting on a Service:

- Vercel
- GitHub Pages
- Netlify
- Render

Using Vercel

Deploying your Docusaurus project to Vercel will provide you with various benefits in the areas of performance and ease of use.

Most importantly, however, deploying a Docusaurus project only takes a couple of seconds:

1. First, install their command-line interface:

```
npm i -g vercel
```

2. Run a single command inside the root directory of your project:

```
vercel
```

That's all. Your docs will automatically be deployed.

> Note that the directory structure Now supports is slightly different from the default directory structure of a Docusaurus project - The `docs` directory has to be within the `website` directory, ideally following the directory structure in this example. You will also have to specify a `customDocsPath` value in `siteConfig.js`. Take a look at the now-examples repository for a Docusaurus project.

Using GitHub Pages

Docusaurus was designed to work well with one of the most popular hosting solutions for open source projects: GitHub Pages.

Deploying to GitHub Pages

1. Docusaurus supports deploying as project pages or user/organization pages, your code repository does not even need to be public.

> Even if your repository is private, anything published to a `gh-pages` branch will be public.

Note: When you deploy as user/organization page, the publish script will deploy these sites to the root of the **`master`** branch of the *username*.github.io repo. In this case, note that you will want to have the Docusaurus infra, your docs, etc. either in **another branch of the *username*.github.io repo** (e.g., maybe call it `source`), or in another, separate repo (e.g. in the same as the documented source code).

2. You will need to modify the file `website/siteConfig.js` and add the required parameters.

Name	Description
`organizationName`	The GitHub user or organization that owns the repository. If you are the owner, then it is your GitHub username. In the case of Docusaurus, that would be the "*facebook*" GitHub organization.
`projectName`	The name of the GitHub repository for your project. For example, the source code for Docusaurus is hosted at https://github.com/facebook/docusaurus, so our project name, in this case, would be "docusaurus".
`url`	Your website's URL. For projects hosted on GitHub pages, this will be "https://*username*.github.io"
`baseUrl`	Base URL for your project. For projects hosted on GitHub pages, it follows the format "*/projectName/*". For https://github.com/facebook/docusaurus, `baseUrl` is `/docusaurus/`.

```
const siteConfig = {
  ...
  url: 'https://__userName__.github.io', // Your website URL
  baseUrl: '/testProject/',
  projectName: 'testProject',
  organizationName: 'userName'
  ...
}
```

In case you want to deploy as a user or organization site, specify the project name as `<username>.github.io` or `<orgname>.github.io`. E.g. If your GitHub username is "user42" then *user42.github.io*, or in the case of an organization name of "org123", it will be *org123.github.io*.

Note: Not setting the `url` and `baseUrl` of your project might result in incorrect file paths generated which can cause broken links to assets paths like stylesheets and images.

> While we recommend setting the `projectName` and `organizationName` in `siteConfig.js`, you can also use environment variables `ORGANIZATION_NAME` and `PROJECT_NAME`.

3. Now you have to specify the git user as an environment variable, and run the script `publish-gh-pages`

Name	Description
`GIT_USER`	The username for a GitHub account that has to commit access to this repo. For your repositories, this will usually be your own GitHub username. The specified `GIT_USER`

Name	Description
	must have push access to the repository specified in the combination of `organizationName` and `projectName`.

To run the script directly from the command-line, you can use the following, filling in the parameter values as appropriate.

Bash

```
GIT_USER=<GIT_USER> \
  CURRENT_BRANCH=master \
  USE_SSH=true \
  yarn run publish-gh-pages # or `npm run publish-gh-pages`
```

Windows

```
cmd /C "set "GIT_USER=<GIT_USER>"&& set CURRENT_BRANCH=master && set USE_SSH=true && yarn
```

There are also two optional parameters that are set as environment variables:

Name	Description
`USE_SSH`	If this is set to `true`, then SSH is used instead of HTTPS for the connection to the GitHub repo. HTTPS is the default if this variable is not set.
`CURRENT_BRANCH`	The branch that contains the latest docs changes that will be deployed. Usually, the branch will be `master`, but it could be any branch (default or otherwise) except for `gh-pages`. If nothing is set for this variable, then the current branch will be used.

If you run into issues related to SSH keys, visit GitHub's authentication documentation.

You should now be able to load your website by visiting its GitHub Pages URL, which could be something along the lines of https://*username*.github.io/*projectName*, or a custom domain if you have set that up. For example, Docusaurus' own GitHub Pages URL is https://facebook.github.io/Docusaurus because it is served from the `gh-pages` branch of the https://github.com/facebook/docusaurus GitHub repository. However, it can also be accessed via https://docusaurus.io/, via a generated `CNAME` file which can be configured via the `cname` siteConfig option.

We highly encourage reading through the GitHub Pages documentation to learn more about how this hosting solution works.

You can run the command above any time you update the docs and wish to deploy the changes to your site. Running the script manually may be fine for sites where the documentation rarely changes and it is not too much of an inconvenience to remember to manually deploy changes.

However, you can automate the publishing process with continuous integration (CI).

Automating Deployments Using Continuous Integration

Continuous integration (CI) services are typically used to perform routine tasks whenever new commits are checked in to source control. These tasks can be any combination of running unit tests and integration tests, automating builds, publishing packages to NPM, and yes, deploying changes to your website. All you need to do to automate the deployment of your website is to invoke the `publish-gh-pages` script whenever your docs get updated. In the following section, we'll be covering how to do just that using CircleCI, a popular continuous integration service provider.

Using CircleCI 2.0

If you haven't done so already, you can setup CircleCI for your open source project. Afterwards, in order to enable automatic deployment of your site and documentation via CircleCI, just configure Circle to run the `publish-gh-pages` script as part of the deployment step. You can follow the steps below to get that setup.

1. Ensure the GitHub account that will be set as the `GIT_USER` has `write` access to the repository that contains the documentation, by checking `Settings | Collaborators & teams` in the repository.
2. Log into GitHub as the `GIT_USER`.
3. Go to https://github.com/settings/tokens for the `GIT_USER` and generate a new personal access token, granting it full control of private repositories through the `repository` access scope. Store this token in a safe place, making sure to not share it with anyone. This token can be used to authenticate GitHub actions on your behalf in place of your GitHub password.
4. Open your CircleCI dashboard, and navigate to the Settings page for your repository, then select "Environment variables". The URL looks like https://circleci.com/gh/ORG/REPO/edit#env-vars, where "ORG/REPO" should be replaced with your own GitHub organization/repository.
5. Create a new environment variable named `GITHUB_TOKEN`, using your newly generated access token as the value.
6. Create a `.circleci` directory and create a `config.yml` under that directory.
7. Copy the text below into `.circleci/config.yml`.

```
# If you only want the circle to run on direct commits to master, you can uncomment this
# and uncomment the filters: *filter-only-master down below too
```

```
#
# aliases:
#   - &filter-only-master
#     branches:
#       only:
#         - master

version: 2
jobs:
  deploy-website:
    docker:
      # specify the version you desire here
      - image: circleci/node:8.11.1

    steps:
      - checkout
      - run:
          name: Deploying to GitHub Pages
          command: |
            git config --global user.email "<GITHUB_USERNAME>@users.noreply.github.com"
            git config --global user.name "<YOUR_NAME>"
            echo "machine github.com login <GITHUB_USERNAME> password $GITHUB_TOKEN" > ~/
            cd website && yarn install && GIT_USER=<GIT_USER> yarn run publish-gh-pages

workflows:
  version: 2
  build_and_deploy:
    jobs:
      - deploy-website:
#         filters: *filter-only-master
```

Make sure to replace all `<....>` in the `command:` sequence with appropriate values. For `<GIT_USER>`, it should be a GitHub account that has access to push documentation to your GitHub repository. Many times `<GIT_USER>` and `<GITHUB_USERNAME>` will be the same.

DO NOT place the actual value of `$GITHUB_TOKEN` in `circle.yml`. We already configured that as an environment variable back in Step 5.

> If you want to use SSH for your GitHub repository connection, you can set `USE_SSH=true`. So the above command would look something like: `cd website && npm install && GIT_USER=<GIT_USER> USE_SSH=true npm run publish-gh-pages`.

> Unlike when you run the `publish-gh-pages` script manually when the script runs within the Circle environment, the value of `CURRENT_BRANCH` is already defined as an environment variable

> within CircleCI and will be picked up by the script automatically.

Now, whenever a new commit lands in `master`, CircleCI will run your suite of tests and, if everything passes, your website will be deployed via the `publish-gh-pages` script.

> If you would rather use a deploy key instead of a personal access token, you can by starting with the CircleCI instructions for adding a read/write deploy key.

Tips & Tricks

When initially deploying to a `gh-pages` branch using CircleCI, you may notice that some jobs triggered by commits to the `gh-pages` branch fail to run successfully due to a lack of tests (This can also result in chat/slack build failure notifications).

You can work around this by:

- Setting the environment variable `CUSTOM_COMMIT_MESSAGE` flag to the `publish-gh-pages` command with the contents of `[skip ci]`. e.g.

```
CUSTOM_COMMIT_MESSAGE="[skip ci]" \
  yarn run publish-gh-pages # or `npm run publish-gh-pages`
```

- Alternatively, you can work around this by creating a basic CircleCI config with the following contents:

```
# CircleCI 2.0 Config File
# This config file will prevent tests from being run on the gh-pages branch.
version: 2
jobs:
  build:
    machine: true
    branches:
      ignore: gh-pages
    steps:
      - run: echo "Skipping tests on gh-pages branch"
```

Save this file as `config.yml` and place it in a `.circleci` directory inside your `website/static` directory.

Using Travis CI

1. Go to https://github.com/settings/tokens and generate a new personal access token
2. Using your GitHub account, add the Travis CI app to the repository you want to activate.
3. Open your Travis CI dashboard. The URL looks like https://travis-ci.com/USERNAME/REPO, and navigate to the `More options` > `Setting` > `Environment Variables` section of your repository.
4. Create a new environment variable named `GH_TOKEN` with your newly generated token as its value, then `GH_EMAIL` (your email address) and `GH_NAME` (your GitHub username).
5. Create a `.travis.yml` on the root of your repository with below text.

```
# .travis.yml
language: node_js
node_js:
  - '8'
branches:
  only:
    - master
cache:
  yarn: true
script:
  - git config --global user.name "${GH_NAME}"
  - git config --global user.email "${GH_EMAIL}"
  - echo "machine github.com login ${GH_NAME} password ${GH_TOKEN}" > ~/.netrc
  - cd website && yarn install && GIT_USER="${GH_NAME}" yarn run publish-gh-pages
```

Now, whenever a new commit lands in `master`, Travis CI will run your suite of tests and, if everything passes, your website will be deployed via the `publish-gh-pages` script.

Using Azure Pipelines

1. Sign Up at Azure Pipelines if you haven't already.
2. Create an organization and within the organization create a project and connect your repository from GitHub.
3. Go to https://github.com/settings/tokens and generate a new personal access token with repository scope.
4. In the project page (which looks like https://dev.azure.com/ORG_NAME/REPO_NAME/_build) create a new pipeline with the following text. Also, click on edit and add a new environment variable named `GH_TOKEN` with your newly generated token as its value, then `GH_EMAIL` (your email address) and `GH_NAME` (your GitHub username). Make sure to mark them as secret. Alternatively, you can also add a file named `azure-pipelines.yml` at your repository root.

```
# azure-pipelines.yml
trigger:
  - master

pool:
  vmImage: 'ubuntu-latest'

steps:
  - checkout: self
    persistCredentials: true

  - task: NodeTool@0
    inputs:
      versionSpec: '10.x'
    displayName: 'Install Node.js'

  - script: |
      git config --global user.name "${GH_NAME}"
      git config --global user.email "${GH_EMAIL}"
      git checkout -b master
      echo "machine github.com login ${GH_NAME} password ${GH_TOKEN}" > ~/.netrc
      cd website
      yarn install
      GIT_USER="${GH_NAME}" CURRENT_BRANCH=master yarn run publish-gh-pages
    env:
      GH_NAME: $(GH_NAME)
      GH_EMAIL: $(GH_EMAIL)
      GH_TOKEN: $(GH_TOKEN)
    displayName: 'yarn install and build'
```

Using Drone

1. Create a new ssh key that will be the deploy key for your project.
2. Name your private and public keys to be specific and so that it does not overwrite your other ssh keys.
3. Go to https://github.com/USERNAME/REPO/settings/keys and add a new deploy key by pasting in our public key you just generated.
4. Open your Drone.io dashboard and login. The URL looks like https://cloud.drone.io/USERNAME/REPO.
5. Click on the repository, click on activate repository, and add a secret called `git_deploy_private_key` with your private key value that you just generated.
6. Create a `.drone.yml` on the root of your repository with below text.

```
# .drone.yml
kind: pipeline
```

```
  type: docker
  trigger:
    event:
      - tag
  - name: Website
    image: node
    commands:
      - mkdir -p $HOME/.ssh
      - ssh-keyscan -t rsa github.com >> $HOME/.ssh/known_hosts
      - echo "$GITHUB_PRIVATE_KEY > $HOME/.ssh/id_rsa"
      - chmod 0600 $HOME/.ssh/id_rsa
      - cd website
      - npm i
      - npm run publish-gh-pages
    environment:
      USE_SSH: true
      GIT_USER: $DRONE_COMMIT_AUTHOR
      GITHUB_PRIVATE_KEY: git_deploy_private_key
```

Now, whenever you push a new tag to github, this trigger will start the drone ci job to publish your website.

Hosting on Vercel

Deploying your Docusaurus project to Vercel will provide you with various benefits in the areas of performance and ease of use.

To deploy your Docusaurus project with a Vercel for Git Integration, make sure it has been pushed to a Git repository.

Import the project into Vercel using the Import Flow. During the import, you will find all relevant options preconfigured for you; however, you can choose to change any of these options, a list of which can be found here.

After your project has been imported, all subsequent pushes to branches will generate Preview Deployments, and all changes made to the Production Branch (commonly "main") will result in a Production Deployment.

Hosting on Netlify

Steps to configure your Docusaurus-powered site on Netlify.

1. Select **New site from Git**

2. Connect to your preferred Git provider.

3. Select the branch to deploy. Default is `master`

4. Configure your build steps:

   - For your build command enter: `cd website; npm install; npm run build;`
   - For publish directory: `website/build/<projectName>` (use the `projectName` from your `siteConfig`)

5. Click **Deploy site**

You can also configure Netlify to rebuild on every commit to your repository, or only `master` branch commits.

Hosting on Render

Render offers free static site hosting with fully managed SSL, custom domains, a global CDN and continuous auto deploy from your Git repo. Deploy your app in just a few minutes by following these steps.

1. Create a new **Web Service** on Render, and give Render's GitHub app permission to access your Docusaurus repo.

2. Select the branch to deploy. The default is `master`.

3. Enter the following values during creation.

Field	Value
Environment	`Static Site`
Build Command	`cd website; yarn install; yarn build`
Publish Directory	`website/build/<projectName>`

   `projectName` is the value you defined in your `siteConfig.js`.

   ```
   const siteConfig = {
     // ...
     projectName: 'your-project-name',
     // ...
   ```

That's it! Your app will be live on your Render URL as soon as the build finishes.

Publishing to GitHub Enterprise

GitHub enterprise installations should work in the same manner as github.com; you only need to identify the organization's GitHub Enterprise host.

Name	Description
`GITHUB_HOST`	The hostname for the GitHub enterprise server.

Alter your `siteConfig.js` to add a property `'githubHost'` which represents the GitHub Enterprise hostname. Alternatively, set an environment variable `GITHUB_HOST` when executing the publish command.

Docker is a tool that enables you to create, deploy, and manage lightweight, stand-alone packages that contain everything needed to run an application. It can help us to avoid conflicting dependencies & unwanted behavior when running Docusaurus.

Run the local web server in docker

Ensure you have previously installed docker.

To run the local web server:

1. **Build the docker image** -- Enter the folder where you have Docusaurus installed. Run `docker build -t docusaurus-doc .`

   Once the build phase finishes, you can verify the image exists by running `docker images`.

   > We now include a `Dockerfile` when you install Docusaurus.

2. **Run the Docusaurus container** -- To start docker run `docker run --rm -p 3000:3000 docusaurus-doc`

   This will start a docker container with the image `docusaurus-doc`. To see more detailed container info run `docker ps` .

To access Docusaurus from outside the docker container you must add the `--host` flag to the `docusaurus-start` command as described in: API Commands

Use docker-compose

We can also use `docker-compose` to configure our application. This feature of docker allows you to run the web server and any additional services with a single command.

> Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Using Compose is a three-step process:

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.

2. Define the services that make up your app in `docker-compose.yml` so they can be run together in an isolated environment.

3. Run `docker-compose up` and Compose starts and runs your entire app.

We include a basic `docker-compose.yml` in your project:

```
version: '3'

services:
  docusaurus:
    build: .
    ports:
      - 3000:3000
      - 35729:35729
    volumes:
      - ./docs:/app/docs
      - ./website/blog:/app/website/blog
      - ./website/core:/app/website/core
      - ./website/i18n:/app/website/i18n
      - ./website/pages:/app/website/pages
      - ./website/static:/app/website/static
      - ./website/sidebars.json:/app/website/sidebars.json
      - ./website/siteConfig.js:/app/website/siteConfig.js
    working_dir: /app/website
```

To run a local web server with `docker-compose` run `docker-compose up`.

To build static HTML pages for publishing run `docker-compose run docusaurus bash -c 'yarn publish-gh-pages'`

Initial Setup

To setup your site's blog, start by creating a `blog` directory within your repo's `website` directory.

Then, add a header link to your blog within `siteConfig.js` :

```
headerLinks: [
    ...
    { blog: true, label: 'Blog' },
    ...
]
```

Adding Posts

To publish in the blog, create a file within the blog directory with a formatted name of `YYYY-MM-DD-my-blog-post-title.md` . The post date is extracted from the file name.

For example, at `website/blog/2017-12-14-introducing-docusaurus.md` :

```
---
title: Introducing Docusaurus
author: Joel Marcey
authorURL: http://twitter.com/JoelMarcey
authorFBID: 611217057
authorTwitter: JoelMarcey
---
Lorem Ipsum...
```

Adding slug will override the url of the blog post.

For example:

```
---
slug: introducing-docusaurus
title: Introducing Docusaurus
author: Joel Marcey
authorURL: http://twitter.com/JoelMarcey
authorFBID: 611217057
authorTwitter: JoelMarcey
```

```
---
Lorem Ipsum...
```

Will be available at `https://website/blog/introducing-docusaurus`

Header Options

The only required field is `title`; however, we provide options to add author information to your blog post as well along with other options.

- `author` - The text label of the author byline.
- `authorURL` - The URL associated with the author. This could be a Twitter, GitHub, Facebook account, etc.
- `authorFBID` - The Facebook profile ID that is used to fetch the profile picture.
- `authorImageURL` - The URL to the author's image. (Note: If you use both `authorFBID` and `authorImageURL`, `authorFBID` will take precedence. Don't include `authorFBID` if you want `authorImageURL` to appear.)
- `title` - The blog post title.
- `slug` - The blog post url slug. Example: `/blog/my-test-slug`. When not specified, the blog url slug will be extracted from the file name.
- `unlisted` - The post will be accessible by directly visiting the URL but will not show up in the sidebar in the final build; during local development, the post will still be listed. Useful in situations where you want to share a WIP post with others for feedback.
- `draft` - The post will not appear if set to `true`. Useful in situations where WIP but don't want to share the post.

Summary Truncation

Use the `<!--truncate-->` marker in your blog post to represent what will be shown as the summary when viewing all published blog posts. Anything above `<!--truncate-->` will be part of the summary. For example:

```
---
title: Truncation Example
---
All this will be part of the blog post summary.

Even this.

<!--truncate-->
```

```
But anything from here on down will not be.

Not this.

Or this.
```

Changing How Many Blog Posts Show on Sidebar

By default, 5 recent blog posts are shown on the sidebar.

You can configure a specific amount of blog posts to show by adding a `blogSidebarCount` setting to your `siteConfig.js`.

The available options are an integer representing the number of posts you wish to show or a string with the value `'ALL'`.

Example:

```
blogSidebarCount: 'ALL',
```

Changing The Sidebar Title

You can configure a specific sidebar title by adding a `blogSidebarTitle` setting to your `siteConfig.js`.

The option is an object which can have the keys `default` and `all`. Specifying a value for `default` allows you to change the default sidebar title. Specifying a value for `all` allows you to change the sidebar title when the `blogSidebarCount` option is set to `'ALL'`.

Example:

```
blogSidebarTitle: { default: 'Recent posts', all: 'All blog posts' },
```

RSS Feed

Docusaurus provides an RSS feed for your blog posts. Both RSS and Atom feed formats are supported. This data is automatically added to your website page's HTML `<HEAD>` tag.

A summary of the post's text is provided in the RSS feed up to the `<!--truncate-->`. If no `<!--truncate-->` tag is found, then all text up to 250 characters are used.

Social Buttons

If you want Facebook and/or Twitter social buttons at the bottom of your blog posts, set the `facebookAppId` and/or `twitter` site configuration options in `siteConfig.js`.

Advanced Topics

I want to run in "Blog Only" mode.

You can run your Docusaurus site without a landing page and instead have your blog load first.

To do this:

1. Create a file `index.html` in `website/static/`.
2. Place the contents of the template below into `website/static/index.html`
3. Customize the `<title>` of `website/static/index.html`
4. Delete the dynamic landing page `website/pages/en/index.js`

> Now, when Docusaurus generates or builds your site, it will copy the file from `static/index.html` and place it in the site's main directory. The static file is served when a visitor arrives on your page. When the page loads, it will redirect the visitor to `/blog`.

You can use this template:

Copy

```
<!DOCTYPE html>
<html lang="en-US">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="refresh" content="0; url=blog/" />
    <script type="text/javascript">
      window.location.href = 'blog/';
    </script>
    <title>Title of Your Blog</title>
  </head>
  <body>
    If you are not redirected automatically, follow this
    <a href="blog/">link</a>.
  </body>
</html>
```

You can add pages to your site that are not part of the standard docs or blog markdown files. You can do this by adding `.js` files to the `website/pages` directory. These files are React components and the `render()` is called to create them, backed by CSS classes, etc.

Customizing Your Home Page

The easiest way to get started customizing your home page is to use the example site that was created when you ran the Docusaurus initialization script.

You can start your local server and go to `http://localhost:3000` to see what the example home page looks like. From there, edit the `website/pages/en/index.js` file and its various components to use the images and text you want for your project.

Adding Other Custom Pages

Docusaurus provides some helpful example pages in the `website/pages/en` directory, including `index.js`, `users.js`, and `help.js`. These are good examples to showcase how to create a custom page for Docusaurus.

```
root-directory
├── docs
└── website
    ├── blog
    ├── core
    |   └── Footer.js
    ├── package.json
    ├── pages
    |   └── en
    |       ├── help.js
    |       ├── index.js
    |       └── users.js
    ├── sidebars.json
    ├── siteConfig.js
    └── static
```

You are also free to write your own pages. It is strongly suggested that you at least have an index page, but none of the pages provided are mandatory to include in your site. More information on how to use the provided components or include your own can be found here. Information on how to link to your different pages in the header navigation bar can be found here.

> If you want your page to show up in your navigation header, you will need to update `siteConfig.js` to add to the `headerLinks` element. e.g., `{ page: 'about-slash', label: 'About/' }`,

Adding Static Pages

Static `.html` files can also be used, but they will not include Docusaurus' header, footer, or styles by default. These can be added to the `static` directory in the same way as other static assets. Alternatively, they can be placed in the `pages` directory and would be served as-is instead of being rendered from React.

If you wish to use Docusaurus' stylesheet, you can access it at `${baseUrl}css/main.css`. If you wish to use separate CSS for these static pages, you can exclude them from being concatenated to Docusaurus' styles by adding them into the `siteConfig.separateCss` field in `siteConfig.js`.

> You can set the `$wrapPagesHTML` site config option in order to wrap raw HTML fragments with the Docusaurus site styling, header and footer.

Customizing Your Site Footer

Starting from the example `core/Footer.js` file that was created when you ran the Docusaurus initialization script, edit the footer to include any links to pages on your site or other sites that you wish to have.

The example provided has three columns with a footer image on the left and Facebook's open source logo and copyright at the bottom. If your project is not a Facebook open source project, remove the logo and copyright. Otherwise, feel free to get creative with your footer and make it look however you'd like!

Some suggestions for links you may want to provide: documentation, API, Twitter, Discord, Facebook groups, Stack Overflow, GitHub, etc.

Your footer will automatically get applied to all pages on your site, including docs and blog posts. The sole exception to this is any static HTML pages you include.

If you do not want a footer for your site, change the `render` function of `core/Footer.js` to return `null`. e.g.,

Copy

```
const React = require('react');

class Footer extends React.Component {
  render() {
    return null;
  }
}

module.exports = Footer;
```

Docusaurus supports search using Algolia DocSearch. Once your website is online, you can submit it to DocSearch. Algolia will then send you credentials you can add to your `siteConfig.js`.

DocSearch works by crawling the content of your website every 24 hours and putting all the content in an Algolia index. This content is then queried directly from your front-end using the Algolia API. Note that your website needs to be publicly available for this to work (ie. not behind a firewall). This service is free.

Enabling the Search Bar

Enter your API key and index name (sent by Algolia) into `siteConfig.js` in the `algolia` section to enable search for your site.

```
const siteConfig = {
  ...
  algolia: {
    apiKey: 'my-api-key',
    indexName: 'my-index-name',
    appId: 'app-id', // Optional, if you run the DocSearch crawler on your own
    algoliaOptions: {} // Optional, if provided by Algolia
  },
  ...
};
```

Extra Search Options

You can also specify extra search options used by Algolia by using an `algoliaOptions` field in `algolia`. This may be useful if you want to provide different search results for the different versions or languages of your docs. Any occurrences of "VERSION" or "LANGUAGE" will be replaced by the version or language of the current page, respectively. More details about search options can be found here.

```
const siteConfig = {
  ...
  algolia: {
    ...
    algoliaOptions: {
      facetFilters: [ "language:LANGUAGE", "version:VERSION" ]
    }
  },
};
```

Algolia might provide you with extra search options. If so, you should add them to the `algoliaOptions` object.

Controlling the Location of the Search Bar

By default, the search bar will be the rightmost element in the top navigation bar.

If you want to change the default location, add the `searchBar` flag in the `headerLinks` field of `siteConfig.js` in your desired location. For example, you may want the search bar between your internal and external links.

```
const siteConfig = {
  ...
  headerLinks: [
    {...}
    {...}
    { search: true }
    {...}
    {...}
  ],
  ...
};
```

Customizing the placeholder

If you want to change the placeholder (which defaults to *Search*), add the `placeholder` field in your config. For example, you may want the search bar to display *Ask me something*:

```
const siteConfig = {
  ...
  algolia: {
    ...
    placeholder: 'Ask me something'
  },
};
```

Disabling the Search Bar

To disable the search bar, comment out (recommended) or delete the `algolia` section in the `siteConfig.js` file.

Also, if you have customized the location of the search bar in `headerLinks`, set `search: false`.

Referencing Site Documents

If you want to reference another document in your `docs` directory (or the location you set via the optional `customDocsPath` path site configuration option), then you just use the name of the document you want to reference.

For example, if you are in `doc2.md` and you want to reference `doc1.md`:

```
I am referencing a [document](doc1.md).
```

How Documents are Linked

New markdown files within `docs` will show up as pages on the website. Links to those documents are created first by using the `id` in the header of each document. If there is no `id` field, then the name of the file will serve as the link name.

For example, creating an empty file such as `docs/getting-started.md` will enable the new page URL as `/docs/getting-started.html`.

Suppose you add this to your document:

```
id: intro
title: Getting Started
---
My new content here..
```

If you set the `id` field in the markdown header of the file, the doc will then be accessed from a URL of the form `/docs/intro.html`.

> You need an `id` field to be able to add the document to the sidebar.

Adding Documents to a Sidebar

Many times, you will want to add a document to a sidebar that will be associated with one of the headers in the top navigation bar of the website. The most common sidebar, and the one that comes installed when Docusaurus is initialized, is the `docs` sidebar.

"docs" is just a name. It has no inherent meaning. You can change it as you wish.

You configure the contents of the sidebar, and the order of its documents, in the `website/sidebars.json` file.

Until you add your document to `website/sidebars.json`, they will only be accessible via a direct URL. The doc will not show up in any sidebar.

Within `sidebars.json`, add the `id` you used in the document header to the existing sidebar/category. In the below case, `docs` is the name of the sidebar and `Getting Started` is a category within the sidebar.

```
{
  "docs": {
    "Getting Started": [
      "getting-started"
    ],
    ...
  },
  ...
}
```

Or you can create a new category within the sidebar:

```
{
  "docs": {
    "My New Sidebar Category": [
      "getting-started"
    ],
    ...
  },
  ...
}
```

However, for a document located in a docs subdirectory like below:

```
docs
└── dir1
    └── getting-started.md
```

You should provide `directory/id` instead of `id` in `sidebars.json`.

```
{
  "docs": {
    "My New Sidebar Category": [
      "dir1/getting-started"
    ],
    ...
  },
  ...
}
```

Adding Subcategories

It is possible to add subcategories to a sidebar. Instead of using IDs as the contents of the category array like the previous examples, you can pass an object where the keys will be the subcategory name and the value an array of IDs for that subcategory.

```
{
  "docs": {
    "My Example Category": [
      "examples",
      {
        "type": "subcategory",
        "label": "My Example Subcategory",
        "ids": [
          "my-examples",
          ...
        ]
      },
      {
        "type": "subcategory",
        "label": "My Next Subcategory",
        "ids": [
          "some-other-examples"
        ]
      },
      "even-more-examples",
      ...
    ],
    ...
  }
}
```

```
/*
The above will generate:

- My Example Category
  - examples
  - My Example Subcategory
    - my-examples
    ...
  - My Next Subcategory
    - some-other-examples
  - even-more-examples
  ...
*/
```

Adding New Sidebars

You can also put a document in a new sidebar. In the following example, we are creating an `examples-sidebar` sidebar within `sidebars.json` that has a category called `My Example Category` containing a document with an `id` of `my-examples`.

```
{
  "examples-sidebar": {
    "My Example Category": [
      "my-examples"
    ],
    ...
  },
  ...
}
```

It is important to note that until you add a document from the `"examples-sidebar"` sidebar to the nav bar, it will be hidden.

Additions to the Site Navigation Bar

To expose sidebars, you will add click-able labels to the site navigation bar at the top of the website. You can add documents, pages and external links.

Adding Documents

After creating a new sidebar for the site by adding it to `sidebars.json`, you can expose the new sidebar from the top navigation bar by editing the `headerLinks` field of `siteConfig.js`.

```
{
  headerLinks: [
    ...
    { doc: 'my-examples', label: 'Examples' },
    ...
  ],
  ...
}
```

A label called `Examples` will be added to the site navigation bar and when you click on it at the top of your site, the `examples-sidebar` will be shown and the default document will be `my-examples`.

Adding Custom Pages

To add custom pages to the site navigation bar, entries can be added to the `headerLinks` of `siteConfig.js`. For example, if we have a page within `website/pages/help.js`, we can link to it by adding the following:

```
{
  headerLinks: [
    ...
    { page: 'help', label: 'Help' },
    ...
  ],
  ...
}
```

A label called `Help` will be added to the site navigation bar and when you click on it at the top of your site, the content from the `help.js` page will be shown.

Adding External Links

Custom links can be added to the site navigation bar with the following entry in `siteConfig.js`:

```
{
  headerLinks: [
    ...
    { href: 'https://github.com/facebook/docusaurus', label: 'GitHub' },
    ...
  ],
  ...
}
```

A label called `GitHub` will be added to the site navigation bar and when you click on it at the top of your site, the content of the external link will be shown.

> To open external links in a new tab, provide an `external: true` flag within the header link config.

Site Navigation Bar Positioning

You have limited control where the search and languages dropdown elements are shown in the site navigation bar at the top of your website.

Search

If search is enabled on your site, your search bar will appear to the right of your links. If you want to put the search bar between links in the header, add a search entry in the `headerLinks` config array:

```
{
  headerLinks: [
    { doc: 'foo', label: 'Foo' },
    { search: true },
    { doc: 'bar', label: 'Bar' },
  ],
  ...
}
```

Languages Dropdown

If translations are enabled on your site, the language dropdown will appear to the right of your links (and to the left of the search bar, if search is enabled). If you want to put the language selection dropdown between links in the header, add a languages entry in the `headerLinks` config array:

```
{
  headerLinks: [
    { doc: 'foo', label: 'Foo' },
    { languages: true },
    { doc: 'bar', label: 'Bar' },
  ],
  ...
}
```

Active Links In Site Navigation Bar

The links in the top navigation bar get `siteNavItemActive` and `siteNavGroupActive` class names to allow you to style the currently active link different from the others. `siteNavItemActive` is applied when there's an exact match between the navigation link and the currently displayed web page.

> This does not include links of type `href` which are meant for external links only. If you manually set an `href` in your `headerLinks` to an internal page, document, or blog post, it will not get the `siteNavItemActive` class even if that page is being displayed.

The `siteNavGroupActive` class will be added to these links:

- `doc` links that belong to the same sidebar as the currently displayed document
- The blog link when a blog post or the blog listing page is being displayed

These are two separate class names so you can have the active styles applied to either exact matches only or a bit more broadly for docs that belong together. If you don't want to make this distinction you can add both classes to the same CSS rule.

Secondary On-Page Navigation

We support secondary on-page navigation so you can quickly see the topics associated with a given document. To enable this feature, you need to add the `onPageNav` site configuration option to your `siteConfig.js`.

```
{
  onPageNav: 'separate',
  ...
}
```

Currently, `'separate'` is the only option available for this field. This provides a separate navigation on the right side of the page.

Collapsible Categories

For sites with a sizable amount of content, we support the option to expand/collapse the links and subcategories under categories. To enable this feature, set the `docsSideNavCollapsible` site configuration option in `siteConfig.js` to true.

```
{
  docsSideNavCollapsible: true,
  ...
}
```

Docusaurus allows for useful translation functionality using Crowdin. Documentation files written in English are uploaded to Crowdin for translation by users within a community. Top-level pages written with English strings can be translated by wrapping any strings you want to translate in a `<translate>` tag. Other titles and labels will also be found and properly translated.

Docusaurus Translation Configurations

To generate example files for translations with Docusaurus, run the `examples` script with the command line argument `translations`:

```
npm run examples translations
```

or

```
yarn examples translations
```

This will create the following files:

```
pages/en/help-with-translations.js
languages.js
../crowdin.yaml
```

- The `pages/en/help-with-translations.js` file includes the same starter help page generated by the `examples` script but now includes translation tags.

> Generally, you will use `help-with-translations.js` as a guide to enable translations in your other pages, but not actually commit the file to your repo (i.e., you can delete it). However, if you want a Help page, and you currently do not have one, you can rename this file to `help.js` and use it as a starting point.

- The `languages.js` file tells Docusaurus what languages you want to enable for your site. By default, we expect English to be enabled.

- The `crowdin.yaml` file is used to configure Crowdin integration and is copied up one level into your Docusaurus project repo. If your Docusaurus project resides in `/project/website`, then `crowdin.yaml` will be copied to `/project/crowdin.yaml`.

Translating Your Existing Docs

Your documentation files (e.g., the `.md` files that live in your `docs` directory) do not need to be changed or moved to support translations. They will be uploaded to Crowdin to be translated directly.

Enabling Translations on Pages

Pages allow you to customize the layout and specific content of pages like a custom index page or help page.

Pages with text that you want translated should be placed in `website/pages/en` directory.

Wrap strings that you want translated in a `<translate>` tag, and add the following `require` statement to the top of the file:

Copy

```
...
const translate = require('../../server/translate.js').translate;
...
<h2>
  <translate>This header will be translated</translate>
</h2>
...
```

You can also include an optional description attribute to give more context to a translator about how to translate the string:

Copy

```
<p>
  <translate desc="flower, not verb">Rose</translate>
<p>
```

> The `<translate>` tag generally works well on pure strings. If you have a string like "Docusaurus currently provides support to help your website use translations", wrapping the `<translation>` tag around that entire string will cause issues because of the markdown linking, etc. Your options are to not translate those strings, or spread a bunch of `<translate>` tags amongst the pure substrings of that string.

Gathering Strings to Translate

The strings within localized Pages must be extracted and provided to Crowdin.

Add the following script to your `website/package.json` file, if it does not exist already:

```
{
  ...
  "scripts": {
    "write-translations": "docusaurus-write-translations"
  },
  ...
}
```

Running the script will generate a `website/i18n/en.json` file containing all the strings that will be translated from English into other languages.

The script will include text from the following places:

- `title` and `sidebar_label` strings in document markdown headers
- category names in `sidebars.json`
- tagline in `siteConfig.js`
- header link `label` strings in `siteConfig.js`
- strings wrapped in the `<translate>` tag in any `.js` files inside `pages`

Custom Translation Strings

If you want to add additional custom translation strings or override any of the strings that get produced by the script that creates the `website/i18n/en.json` file, you can add a `website/data/custom-translation-strings.json` file. The file should have a form of:

```
{
  "localized-strings": {
    "docs": {
      "id": {
        "title": "string1",
        "sidebar_label": "string2"
      },
      "version-0.0.1-id": {
        "title": "string3",
        "sidebar_label": "string4"
      }
    }
  },
  "pages-strings": {
    "id3": "string3",
    "id4": "string4"
  }
}
```

where `localized-strings` represent strings in your documentation content and `pages-strings` represents metadata in your documentation (e.g., title, links, etc).

Here is an example:

```
{
  "_comment": "This file is used to provide custom strings for translations, including ov
  "localized-strings": {
    "translation": "Translations and Localization"
  },
  "pages-strings": {
    "Help Translate|recruit community translators for your project": "Help Us Translate"
  }
}
```

See the generated `website/i18n/en.json` for an example.

How Strings Get Translated

Docusaurus itself does not do any translation from one language to another. Instead, it integrates Crowdin to upload translations and then download the appropriately translated files from Crowdin.

How Docusaurus Uses String Translations

This section provides context about how translations in Docusaurus works.

Strings

A Docusaurus site has many strings used throughout it that require localization. However, maintaining a list of strings used throughout a site can be laborious. Docusaurus simplifies this by centralizing strings.

The header navigation, for example, can have links to 'Home' or your 'Blog'. This and other strings found in the headers and sidebars of pages are extracted and placed into `i18n/en.json`. When your files are translated, say into Spanish, an `i18n/es-ES.json` file will be downloaded from Crowdin. Then, when the Spanish pages are generated, Docusaurus will replace the English version of corresponding strings with translated strings from the corresponding localized strings file (e.g. In a Spanish enabled site 'Help' will become 'Ayuda').

Markdown Files

For documentation files themselves, translated versions of these files are downloaded and then rendered through the proper layout template.

Other Pages

For other pages, Docusaurus will automatically transform all `<translate>` tags it finds into function calls that return the translated strings from the corresponding localized file *`locale.json`*.

Crowdin

Crowdin is a company that provides translation services. For Open Source projects, Crowdin provides free string translations.

Create your translation project on Crowdin. You can use Crowdin's guides to learn more about the translations workflow. *We suggest that you deselect and do not include "English" as a translatable language to prevent the creation of `en-US` localization files as this can lead to confusion.*

> Ensure in your Crowdin settings, in the Translations section, that "Duplicate Strings" are set to "Hide - all duplicates will share the same translation". This setting will ensure that identical strings between versions share a single translation.

Your project will need a `crowdin.yaml` file generated. If you ran `yarn examples translations` or `npm run examples translations`, this file was created for you on the same level as your `website` directory.

> You will need to install the `crowdin` command line interface. Please follow the installation directions.

The example below can be automatically generated by the Docusaurus cli with the `examples` script. It should be placed in the top level of your project directory to configure how and what files are uploaded/downloaded.

Below is an example Crowdin configuration for the respective languages: German, Spanish, French, Japanese, Korean, Bahasa Indonesia, Portuguese Brazilian, Chinese Simplified, and Chinese Traditional.

```
project_identifier_env: CROWDIN_DOCUSAURUS_PROJECT_ID
api_key_env: CROWDIN_DOCUSAURUS_API_KEY
base_path: './'
```

```
preserve_hierarchy: true

files:
  - source: '/docs/**/*.md'
    translation: '/website/translated_docs/%locale%/**/%original_file_name%'
    languages_mapping: &anchor
      locale:
        'de': 'de'
        'es-ES': 'es-ES'
        'fr': 'fr'
        'ja': 'ja'
        'ko': 'ko'
        'mr': 'mr-IN'
        'pt-BR': 'pt-BR'
        'zh-CN': 'zh-CN'
        'zh-TW': 'zh-TW'
```

You can go here to learn more about customizing your `crowdin.yaml` file.

Setup the Crowdin Scripts

You will want to manually sync your files to and from Crowdin. The sync process will upload any markdown files in `/docs` as well as translatable strings in `website/i18n/en.json`. (These strings can be generated by running `yarn write-translations`.)

You can add the following to your `package.json` to manually trigger Crowdin.

```
"scripts": {
  "crowdin-upload": "crowdin --config ../crowdin.yaml upload sources --auto-update -b mas
  "crowdin-download": "crowdin --config ../crowdin.yaml download -b master"
},
```

Manual File Sync

You will always want to upload your markdown files and translatable strings first and download the translations section. So run the commands in this order:

```
CROWDIN_DOCUSAURUS_PROJECT_ID=YOUR_CROWDIN_PROJECT_ID CROWDIN_DOCUSAURUS_API_KEY=YOUR_CRO
CROWDIN_DOCUSAURUS_PROJECT_ID=YOUR_CROWDIN_PROJECT_ID CROWDIN_DOCUSAURUS_API_KEY=YOUR_CRO
```

`YOUR_CROWDIN_PROJECT_ID` is the name of your Crowdin project. e.g., for https://crowdin.com/project/docusaurus/, that variable would be set to `docusaurus`. `YOUR_CROWDIN_API_KEY` is a unique key that is like a password. You can find it in the `API` tab of your Crowdin project's `Settings`.

These commands require having an environment variable set with your Crowdin project id and api key (`CROWDIN_PROJECT_ID`, `CROWDIN_API_KEY`). You can preface them inline as done above or add them permanently to your `.bashrc` or `.bash_profile`.

If you run more than one localized Docusaurus project on your computer, you should change the name of the environment variables to something unique (`CROWDIN_PROJECTNAME_PROJECT_ID`, `CROWDIN_PROJECTNAME_API_KEY`).

Since the files are generated, you do not need to have any files in your `website/i18n` or `website/translated_docs` directory as part of your repo. So you can can add `website/i18n/*` and `website/translated_docs` to your `.gitignore` file.

Automated File Sync Using CircleCI

You can automate pulling down and uploading translations for your files using the CircleCI web continuous integration service.

First, update the `.circleci/config.yml` file in your project directory to include steps to upload English files to be translated and download translated files using the Crowdin CLI. Here is an example `.circleci/config.yml` file:

```
Copy
# If you only want circle to run on direct commits to master, you can uncomment this out
# and uncomment the filters: *filter-only-master down below too
#
# aliases:
#  - &filter-only-master
#    branches:
#      only:
#        - master

version: 2
jobs:
  deploy-website:
    docker:
      # specify the version you desire here
      - image: circleci/node:8.11.1

    steps:
      - checkout
      - run:
          name: Deploying to GitHub Pages
          command: |
            git config --global user.email "<GITHUB_USERNAME>@users.noreply.github.com"
            git config --global user.name "<YOUR_NAME>"
            echo "machine github.com login <GITHUB_USERNAME> password $GITHUB_TOKEN" > ~/
            # install Docusaurus and generate file of English strings
            - cd website && yarn install && yarn run write-translations && cd ..
            # crowdin install
            - sudo apt-get install default-jre
            - wget https://artifacts.crowdin.com/repo/deb/crowdin.deb -O crowdin.deb
            - sudo dpkg -i crowdin.deb
            # translations upload/download
            - crowdin --config crowdin.yaml upload sources --auto-update -b master
            - crowdin --config crowdin.yaml download -b master
            # build and publish website
            cd website && GIT_USER=<GIT_USER> yarn run publish-gh-pages

workflows:
  version: 2
  build_and_deploy:
    jobs:
      - deploy-website:
#         filters: *filter-only-master
```

The `crowdin` command uses the `crowdin.yaml` file generated with the `examples` script. It should be placed in your project directory to configure how and what files are uploaded/downloaded.

Note that in the `crowdin.yaml` file, `CROWDIN_PROJECT_ID` and `CROWDIN_API_KEY` are environment variables set-up in Circle for your Crowdin project. They can be found in your Crowdin project settings.

Now, Circle will help you automatically get translations prior to building your website. The provided `crowdin.yaml` file will copy translated documents into `website/translated_docs/`, and translated versions of the `i18n/en.json` strings file will into `i18n/${language}.json`.

If you wish to use Crowdin on your machine locally, you can install the Crowdin CLI tool and run the same commands found in the `circle.yaml` file. The only difference is that you must set `project_identifier` and `api_key` values in the `crowdin.yaml` file since you will not have Circle environment variables set up.

Versioned Translations

If you wish to have translation and versioning for your documentation, add the following section to the end of your `crowdin.yaml` file:

```yaml
  -
    source: '/website/versioned_docs/**/*.md'
    translation: '/website/translated_docs/%locale%/**/%original_file_name%'
    languages_mapping: *anchor
```

Translated, versioned documents will be copied into `website/translated_docs/${language}/${version}/`.

You can use the `version` script to cut a new documentation version based on the latest content in the `docs` directory. That specific set of documentation will then be preserved and accessible even as the documentation in the `docs` directory changes moving forward.

How to Create New Versions

Run the following script to generate a starter versions page listing all the site versions:

```
yarn examples versions
```

This creates the `pages/en/versions.js` file.

You can edit this file, later on, to customize how you display the versions.

Add the following script to your `package.json` file if it doesn't already exist:

```
...
"scripts": {
  "version": "docusaurus-version"
},
...
```

Run the script with a command line argument of the version you wish to create. e.g.,

```
yarn run version 1.0.0
```

This will preserve all documents currently in the `docs` directory and make them available as documentation for version `1.0.0`.

If, for example, you ran the version script with `1.0.0` as the version number, version `1.0.0` is considered the latest release version for your project. The site will display the version number next to the title in the header. This version number links to a versions page that you created earlier.

Documents in the `docs` directory will be considered part of version `next` and they are available, for example, at the URL `docs/next/doc1.html`. Documents from the latest version use the URL `docs/doc1.html`.

Running the script again with `yarn run version 2.0.0` will create a version `2.0.0`, making version `2.0.0` the most recent set of documentation. Documents from version `1.0.0` will use the URL `docs/1.0.0/doc1.html` while `2.0.0` will use `docs/doc1.html`.

This table below summarizes Docusaurus versioning at a glance:

Version	Tag	URL
1.0.0	1.0.0	docs/1.0.0/doc1.html
1.0.1	1.0.1	docs/1.0.1/doc1.html
2.0.0	current	docs/doc1.html
`master` branch	next	docs/next/doc1.html

Versioning Patterns

You can create version numbers in whatever format you wish, and a new version can be created with any version number as long as it does not match an existing version. Version ordering is determined by the order in which versions are created, independently of how they are numbered.

Storing Files for Each Version

Versioned documents are placed into `website/versioned_docs/version-${version}`, where `${version}` is the version number you supplied the `version` script.

The markdown header for each versioned doc is altered by renaming the id front matter field to `original_id`, then using `"version-${version}-${original_id}"` as the value for the actual `id` field.

Versioned sidebars are copied into `website/versioned_sidebars` and are named as `version-${version}-sidebars.json`.

A `website/versions.json` file is created the first time you cut a version and is used by Docusaurus to detect what versions exist. Each time a new version is added, it gets added to the `versions.json` file.

If you wish to change the documentation for a past version, you can access the files for that respective version.

Fallback Functionality

Only files in the `docs` directory and sidebar files that differ from those of the latest version will get copied each time a new version is specified. If there is no change across versions, Docusaurus will use the file from the latest version with that file.

For example, a document with the original id `doc1` exists for the latest version, `1.0.0`, and has the same content as the document with the id `doc1` in the `docs` directory. When a new version `2.0.0` is created, the file for `doc1` will not be copied into `versioned_docs/version-2.0.0/`. There will still be a page for `docs/2.0.0/doc1.html`, but it will use the file from version `1.0.0`.

Because of the way this fallback works, pages that you delete are not really deleted from the website unless you tell Docusaurus to skip fallback after a certain version. To do this, use the `deletedDocs` option in `siteConfig.js`.

Renaming Existing Versions

To rename an existing version number to something else, first make sure the following script is in your `package.json` file:

```
...
"scripts": {
  "rename-version": "docusaurus-rename-version"
},
...
```

Run the script with command line arguments of first, the current version name, then second, the new version name. e.g.,

```
yarn run rename-version 1.0.0 1.0.1
```

Versioning and Translations

If you wish to use versioning and translations features, the `crowdin.yaml` file should be set up to upload and download versioned documents to and from Crowdin for translation. Translated, versioned files will go into the directory `translated_docs/${language}/version-${version}/`. For more information, check out the translations guide.

Docusaurus provides a set of scripts to help you generate, serve, and deploy your website. These scripts can be invoked with the `run` command when using Yarn or npm. Some common commands are:

- `yarn run start`: build and serve the website from a local server
- `yarn run examples`: create example configuration files

Running from the command line

The scripts can be run using either Yarn or npm. If you've already gone through our Getting Started guide, you may already be familiar with the `start` command. It's the command that tells Docusaurus to run the `docusaurus-start` script which generates the site and starts up a server, and it's usually invoked like so:

```
yarn run start
```

The same script can be invoked using npm:

```
npm run start
```

To run a particular script, just replace the `start` command in the examples above with the command associated with your script.

Using arguments

Some commands support optional arguments. For example, to start a server on port 8080, you can specify the `--port` argument when running `start`:

```
yarn run start --port 8080
```

If you run Docusaurus using npm, you can still use the command line arguments by inserting a `--` between `npm run <command>` and the command arguments:

```
npm run start -- --port 8080
```

Configuration

These scripts are set up under the `"scripts"` key in your `website/package.json` file as part of the installation process. If you need help setting them up again, please refer to the Installation guide.

Docusaurus provides some default mappings to allow you to run commands following Node conventions. Instead of typing `docusaurus-start` every time, you can type `yarn run start` or `npm start` to achieve the same.

Commands

- `docusaurus-build`
- `docusaurus-examples`
- `docusaurus-publish`
- `docusaurus-rename-version`
- `docusaurus-start`
- `docusaurus-version <version>`
- `docusaurus-write-translations`

Reference

`docusaurus-build`

Alias: `build`.

Options	Default	Description
`--skip-image-compression`	`false`	Skip compression of image assets. You usually won't want to skip this unless your images have already been optimized.
`--skip-next-release`	`false`	Skip the next release documents when versioning is enabled. This will not build HTML files for documents in `/docs` directory.

Generates the static website, applying translations if necessary. Useful for building the website prior to deployment.

See also `docusaurus-start`.

`docusaurus-examples`

Alias: `examples`

Arguments	Default	Description
`<feature>`	-	Specify a feature `translations` or `versions` to generate the extra example files for that feature.

Example

```
docusaurus-examples <feature>
```

When no feature is specified, sets up a minimally configured example website in your project. This command is covered in depth in the Site Preparation guide.

`docusaurus-publish`

Alias: `publish-gh-pages`

Builds, then deploys the static website to GitHub Pages. This command is meant to be run during the deployment step in CircleCI, and therefore expects a few environment variables to be defined:

The following environment variables are generally set manually by the user in the CircleCI `config.yml` file.

- `GIT_USER` : The git user to be associated with the deploy commit.
- `USE_SSH` : Whether to use SSH instead of HTTPS for your connection to the GitHub repo.

Example

```
GIT_USER=docusaurus-bot USE_SSH=true yarn run publish-gh-pages
```

The following environment variables are set by CircleCI during the build process.

- `CIRCLE_BRANCH` : The git branch associated with the commit that triggered the CI run.
- `CI_PULL_REQUEST` : Expected to be truthy if the current CI run was triggered by a commit in a pull request.

The following should be set by you in `siteConfig.js` as `organizationName` and `projectName`, respectively. If they are not set in your site configuration, they fall back to the CircleCI environment.

- `CIRCLE_PROJECT_USERNAME` : The GitHub username or organization name that hosts the Git repo, e.g. "facebook".
- `CIRCLE_PROJECT_REPONAME` : The name of the Git repo, e.g. "Docusaurus".

You can learn more about configuring automatic deployments with CircleCI in the Publishing guide.

`docusaurus-rename-version`

Alias: `rename-version`

Renames an existing version of the docs to a new version name.

Arguments	Default	Description
`<currentVersion>`	-	Version to be renamed.
`<newVersion>`	-	Version to be renamed to.

Example

```
docusaurus-rename-version <currentVersion> <newVersion>
```

See the Versioning guide to learn more.

`docusaurus-start`

Alias: `start`.

This command will build the static website, apply translations if necessary, and then start a local server.

Options	Default	Description
`--port <number>`	`3000`	The website will be served from port 3000 by default, but if the port is taken up, Docusaurus will attempt to find an available one.
`--host <host>`	`localhost`	Specify a host to use. E.g., if you want your server to be accessible externally, you can use --host 0.0.0.0.
`--watch`	-	Whether to watch the files and live reload the page when files are changed. Defaults to true. Disable this by using `--no-`

Options	Default	Description
		`watch` .

You can specify the browser application to be opened by setting the `BROWSER` environment variable before the command, e.g.:

```
$ BROWSER=firefox yarn start
```

`docusaurus-version <version>`

Alias: `version`

Generates a new version of the docs. This will result in a new copy of your site being generated and stored in its own versioned directory. Useful for capturing snapshots of API docs that map to specific versions of your software. Accepts any string as a version number.

See the Versioning guide to learn more.

`docusaurus-write-translations`

Alias: `write-translations`

Writes the English for any strings that need to be translated into a `website/i18n/en.json` file. The script will go through every file in `website/pages/en` and through the `siteConfig.js` file and other config files to fetch English strings that will then be translated on Crowdin. See the Translation guide to learn more.

Docusaurus uses GitHub Flavored Markdown (GFM). Find out more about Docusaurus-specific fields when writing Markdown.

Markdown Headers

Documents

Documents use the following markdown header fields that are enclosed by a line `---` on either side:

- `id`: A unique document id. If this field is not present, the document's `id` will default to its file name (without the extension).
- `title`: The title of your document. If this field is not present, the document's `title` will default to its `id`.
- `hide_title`: Whether to hide the title at the top of the doc.
- `description`: The description of your document which will become the `<meta name="description" content="..."/>` and `<meta property="og:description" content="..."/>` in `<head>`, used by search engines. If this field is not present, it will default to the first line of the contents.
- `sidebar_label`: The text shown in the document sidebar and in the next/previous button for this document. If this field is not present, the document's `sidebar_label` will default to its `title`.

For example:

```
---
id: doc1
title: My Document
sidebar_label: Document
---
```

Versioned documents have their ids altered to include the version number when they get copied. The new `id` is `version-${version}-${id}` where `${version}` is the version number of that document and `${id}` is the original `id`. Additionally, versioned documents get an added `original_id` field with the original document id.

For example:

```
---
id: version-1.0.0-doc1
title: My Document
```

```
sidebar_label: Document
original_id: doc1
---
```

`custom_edit_url`: The URL for editing this document. If this field is not present, the document's edit URL will fall back to `editUrl` from optional fields of `siteConfig.js`. See siteConfig.js docs for more information.

For example:

```
---
id: doc-markdown
title: Markdown Features
custom_edit_url: https://github.com/facebook/docusaurus/edit/master/docs/api-doc-markdown
---
```

Blog Posts

Blog posts use the following markdown header fields that are enclosed by a line `---` on either side:

`title`: The title of this blog post.

`author`: The author of this blog post. If this field is omitted, no author name will be shown.

`authorURL`: A page to link to when a site user clicks the author's name. If this field is omitted, the author's name will not link to anything.

`authorFBID`: The author's Facebook id, used only to get the author's profile picture to display with the blog post. If this field is omitted, no author picture will be shown for the blog post.

For example:

```
---
title: My First Blog Post
author: Frank Li
authorURL: http://twitter.com/franchementli
authorFBID: 100002976521003
---
```

Extra Features

Docusaurus supports some extra features when writing documentation in markdown.

Linking other Documents

You can use relative URLs to other documentation files which will automatically get converted to the corresponding HTML links when they get rendered.

Example:

```
[This links to another document](other-document.md)
```

This markdown will automatically get converted into a link to `/docs/other-document.html` (or the appropriately translated/versioned link) once it gets rendered.

This can help when you want to navigate through docs on GitHub since the links there will be functional links to other documents (still on GitHub), but the documents will have the correct HTML links when they get rendered.

Linking to Images and Other Assets

Static assets can be linked to in the same way that documents are, using relative URLs. Static assets used in documents and blogs should go into `docs/assets` and `website/blog/assets`, respectively. The markdown will get converted into correct link paths so that these paths will work for documents of all languages and versions.

Example:

```
![alt-text](assets/doc-image.png)
```

Generating Table of Contents

You can make an auto-generated list of links, which can be useful as a table of contents for API docs.

In your markdown file, insert a line with the text `<AUTOGENERATED_TABLE_OF_CONTENTS>`. Write your documentation using `h3` headers for each function inside a code block. These will be found by Docusaurus and a list of links to these sections will be inserted at the text ``.

Example:

```
### `docusaurus.function(a, b)`

Text describing my function

### `docdoc(file)`

Text describing my function
```

will lead to a table of contents of the functions:

```
- `docusaurus.function(a, b)`
- `docdoc(file)`
```

and each function will link to their corresponding sections in the page.

Language-specific Code Tabs

Display code in multiple programming languages using code tabs. First, mark the start and end of a code tabs group, by using `<!-- DOCUSAURUS_CODE_TABS -->` and `<!-- END_DOCUSAURUS_CODE_TABS -->` respectively in your markdown. Then start each tab with `<!--[TAB_TITLE]-->`.

Adding the following code to your Markdown file:

~~~
<!--DOCUSAURUS_CODE_TABS-->
<!--JavaScript-->

```js
console.log('Hello, world!');
```
<!--Python-->

```py
print('Hello, world!')
```

<!--C-->

```C
#include <stdio.h>

int main() {
   printf("Hello World!");
   return 0;
~~~

```
}
```
~~~

<!--Pascal-->

```Pascal
program HelloWorld;
begin
  WriteLn('Hello, world!');
end.
~~~

<!--END_DOCUSAURUS_CODE_TABS-->
```

docusaurus-code-tabs.md hosted with ❤ by GitHub view raw

produces this:

JavaScript Python C Pascal

```
console.log('Hello, world!');
```

Copy

Syntax Highlighting

Syntax highlighting is enabled by default on fenced code blocks. The language should be detected automatically, but you can sometimes get better results by specifying the language. You can do so using an info string, following the three opening backticks. The following JavaScript example...

````
```js
ReactDOM.render(<h1>Hello, world!</h1>, document.getElementById('root'));
```
~~~

Copy

...would be rendered with syntax highlighting like so:

```
ReactDOM.render(<h1>Hello, world!</h1>, document.getElementById('root'));
```

Copy

Highlighting is provided by Highlight.js using the theme specified in your `siteConfig.js` file as part of the `highlight` key:
~~~

```
{
  ...
  highlight: {
    theme: 'default'
  }
  ...
}
```

You can find the full list of supported themes in the Highlight.js `styles` directory.

## Registering additional languages

While Highlight.js provides support for many popular languages out of the box, you may find the need to register additional language support. For these cases, we provide an escape valve by exposing the `hljs` constant as part of the `highlight` config key. This, in turn, allows you to call `registerLanguage`:

```
{
  ...
  highlight: {
    theme: 'default',
    hljs: function(hljs) {
      hljs.registerLanguage('galacticbasic', function(hljs) {
        // ...
      });
    }
  }
}
```

## Using Prism as additional syntax highlighter

You can also opt to use Prism to syntax highlight certain languages available in the list here. Include those languages in `usePrism` field in your siteConfig.js

Example:

```
// siteConfig.js
usePrism: ['jsx']
```

Notice that the code block below uses JSX syntax highlighting from Prism.

Copy

```
class Example extends React.Component {
  render() {
    return (
      <View style={{flex: 1, alignItems: 'center', justifyContent: 'center'}}>
        <Text>Docusaurus</Text>
        <Button
          title="Click me"
          onPress={() => this.props.navigation.push('Docusaurus')}
        />
      </View>
    );
  }
}
```

## Adding Copy Code Buttons

Docusaurus allows for adding buttons to copy the code within fenced code blocks. Please follow the instructions here to add "Copy" buttons to your code blocks.

Docusaurus provides support for writing pages as React components inside the `website/pages` directory which will share the same header, footer, and styles as the rest of the site.

## Provided Props

Docusaurus provides your siteConfig.js as a `config` props. Hence, you can access `baseUrl` or `title` through this props.

Example

```
const React = require('react');

class MyPage extends React.Component {
  render() {
    const siteConfig = this.props.config;
    return <div>{siteConfig.title}</div>;
  }
}

module.exports = MyPage;
```

## URLs for Pages

Any `.js` files in `website/pages` will be rendered to static HTML using the path of the file after `pages`. Files in `website/pages/en` will also get copied out into `pages` and will OVERRIDE any files of the same name in `pages`. For example, the page for the `website/pages/en/help.js` file will be found at the URL `${baseUrl}en/help.js` as well as the URL `${baseUrl}help.js`, where `${baseUrl}` is the `baseUrl` fieldset in your siteConfig.js file.

## Titles for Pages

By default, the title of your page is `<title> • <tagline>` where `title` and `tagline` fields are set in `siteConfig.js`. You can exclude the tagline in the title by setting `disableTitleTagline` to `true`. If you want to set a specific title for your custom pages, add a `title` class property on your exported React component.

Example:

```
const React = require('react');
```

```
class MyPage extends React.Component {
  render() {
    // ... your rendering code
  }
}

MyPage.title = 'My Custom Title';

module.exports = MyPage;
```

## Description for Pages

By default, the description your page is `tagline` set in `siteConfig.js`. If you want to set a specific description for your custom pages, add a `description` class property on your exported React component.

Example:

```
const React = require('react');

class MyPage extends React.Component {
  render() {
    // ... your rendering code
  }
}

MyPage.description = 'My Custom Description';

module.exports = MyPage;
```

This will be translated to a description metadata tag on the generated HTML.

```
<meta property="og:description" content="My Custom Description" />
<meta name="description" content="My Custom Description" />
```

## Page Require Paths

Docusaurus provides a few useful React components for users to write their own pages, found in the `CompLibrary` module. This module is provided as part of Docusaurus in `node_modules/docusaurus`, so to access it, pages in the `pages` directory are temporarily copied into `node_modules/docusaurus` when rendering to static HTML. As seen in the example files, this means that a user page at `pages/en/index.js` uses a require path to `'../../core/CompLibrary.js'` to import the provided components.

What this means to the user is that if you wish to use the `CompLibrary` module, make sure the require path is set correctly. For example, a page at `page/mypage.js` would use a path `'../core/CompLibrary.js'`.

If you wish to use your own components inside the website directory, use `process.cwd()` which will refer to the `website` directory to construct require paths. For example, if you add a component to `website/core/mycomponent.js`, you can use the require path, `'process.cwd() + /core/mycomponent.js'`.

## Provided Components

Docusaurus provides the following components in `CompLibrary`:

### `CompLibrary.MarkdownBlock`

A React component that parses markdown and renders to HTML.

Example:

```
const MarkdownBlock = CompLibrary.MarkdownBlock;

<MarkdownBlock>
  [Markdown syntax for a link](http://www.example.com)
</MarkdownBlock>;
```

### `CompLibrary.Container`

A React container component using Docusaurus styles. Has optional padding and background color props that you can configure.

**Props**

| Prop | Type | Default | Description |
|---|---|---|---|
| `padding` | Array of `'all'`, `'bottom'`, `'left'`, `'right'`, `'top'` | `[]` | Positions of the padding. |

| Prop | Type | Default | Description |
|---|---|---|---|
| `background` | One of `'dark'`, `'highlight'`, `'light'` | `null` | Background styling of the element. |
| `className` | String | - | Custom class to add to the element. |

### Example

Copy

```
<Container
  padding={['bottom', 'top']}
  background="light"
  className="myCustomClass">
  ...
</Container>
```

## `CompLibrary.GridBlock`

A React component to organize text and images.

### Props

| Prop | Type | Default | Description |
|---|---|---|---|
| `align` | One of `'left'`, `'center'`, `'right'` | `'left'` | Text alignment of content. |
| `layout` | One of `'twoColumn'`, `'threeColumn'`, `'fourColumn'` | `'twoColumn'` | Number of column sections in the `GridBlock`. |
| `className` | String | - | Custom class to add to the element. |
| `contents` | Array of content objects | `[]` | Contents of each section of the GridBlock. Refer to the next table for the fields available on a content object. |

### Content Object

| Key | Type | Default | Description |
|---|---|---|---|
| `title` | String | - | The display title of this section, which is parsed using Markdown |
| `content` | String | - | The text of this section, which is parsed using Markdown |
| `image` | String | - | The path of the display image |
| `imageAlt` | String | - | The text that will be shown in case the image is not available |
| `imageAlign` | One of `'top'`, `'left'`, `'bottom'`, `'right'` | `'left'` | Image alignment relative to the text |
| `imageLink` | String | - | Link destination from clicking the image |

**Example**

```
<GridBlock
  align="center"
  layout="threeColumn"
  className="myCustomClass"
  contents={[
    {
      title: `[Learn](${siteConfig.baseUrl}${siteConfig.docsUrl}/tutorial.html)`,
      content: 'Learn how to use this project',
      image: siteConfig.baseUrl + 'img/learn.png',
      imageAlt: 'Learn how to use this project',
      imageLink: siteConfig.baseUrl + 'docs/tutorial.html',
    },
    {
      title: 'Frequently Asked Questions',
      content: 'Questions gathered from the community',
      image: siteConfig.baseUrl + 'img/faq.png',
      imageAlign: 'top',
    },
    {
      title: 'More',
      content: 'Lots of documentation is on this site',
    },
  ]}
/>
```

More examples of how these components are used can be found in the generated example files as well as in Docusaurus' own repository for its website set-up.

## Translating Strings

When translations are enabled, any pages inside `website/pages/en` will be translated for all enabled languages. URLs for non-English pages will use their language tags as specified in the `languages.js` file. E.g. The URL for a French page of `website/pages/en/help.js` would be found at `${baseUrl}fr/help.html`.

When writing pages that you wish to translate, wrap any strings to be translated inside a `<translate>` tag. e.g.,

```
<p>
  <translate>I like translations</translate>
</p>
```

You can also provide an optional description attribute to provide context for translators. e.g,

```
<a href="/community">
  <translate desc="Footer link to page referring to community GitHub and Slack">
    Community
  </translate>
</a>
```

Add the following require statement as well:

```
const translate = require('../../server/translate.js').translate;
```

Note that this path is valid for files inside `pages/en` and should be adjusted accordingly if files are in different locations, as discussed above.

## Using Static Assets

Static assets should be placed into the `website/static` directory. They can be accessed by their paths, excluding `static`. For example, if the site's `baseUrl` is `/docusaurus/`, an image in `website/static/img/logo.png` is available at `/docusaurus/img/logo.png`.

## Styles

You should configure your site's primary, secondary, and code block colors using the `colors` field in `siteConfig` as specified here. You can also configure other colors in the same way as described in the `siteConfig` doc.

There are several ways to access the default styles provided for your site. If you have started developing your website and executed the `docusaurus-init` or `yarn install` command, your default styles can be found at `website/node_modules/docusaurus/lib/static/css/main.css`. Alternatively, the `main.css` file may be inspected directly at the Docusaurus GitHub repository.

You can provide your own custom styles by adding them anywhere in the `website/static` directory. Any `.css` files you provide in the `static` directory will get concatenated to the end of Docusaurus' provided styles, allowing you to add to or override Docusaurus default styles as you wish.

One way to figure out what classes you wish to override or add to is to start your server locally and use your browser's inspect element tool.

A large part of the site configuration is done by editing the `siteConfig.js` file.

## User Showcase

The `users` array is used to store objects for each project/user that you want to show on your site. Currently, this field is used by the example `pages/en/index.js` and `pages/en/users.js` files provided. Each user object should have `caption`, `image`, `infoLink`, and `pinned` fields. The `caption` is the text showed when someone hovers over the `image` of that user, and the `infoLink` is where clicking the image will bring someone. The `pinned` field determines whether or not it shows up on the `index` page.

Currently, this `users` array is used only by the `index.js` and `users.js` example files. If you do not wish to have a users page or show users on the `index` page, you may remove this section.

## siteConfig Fields

The `siteConfig` object contains the bulk of the configuration settings for your website.

### Mandatory Fields

#### `baseUrl` [string]

baseUrl for your site. This can also be considered the path after the host. For example, `/metro/` is the baseUrl of https://facebook.github.io/metro/. For URLs that have no path, the baseUrl should be set to `/`. This field is related to the `url` field.

#### `colors` [object]

Color configurations for the site.

- `primaryColor` is the color used the header navigation bar and sidebars.
- `secondaryColor` is the color seen in the second row of the header navigation bar when the site window is narrow (including on mobile).
- Custom color configurations can also be added. For example, if user styles are added with colors specified as `$myColor`, then adding a `myColor` field to `colors` will allow you to configure this color.

#### `copyright` [string]

The copyright string at the footer of the site and within the feed

#### `favicon` [string]

URL for site favicon.

`headerIcon` [string]

URL for icon used in the header navigation bar.

`headerLinks` [array]

Links that will be used in the header navigation bar. The `label` field of each object will be the link text and will also be translated for each language.

Example Usage:

```
headerLinks: [
  // Links to document with id doc1 for current language/version
  { doc: "doc1", label: "Getting Started" },
  // Link to page found at pages/en/help.js or if that does not exist, pages/help.js, for
  { page: "help", label: "Help" },
  // Links to href destination, using target=_blank (external)
  { href: "https://github.com/", label: "GitHub", external: true },
  // Links to blog generated by Docusaurus (${baseUrl}blog)
  { blog: true, label: "Blog" },
  // Determines search bar position among links
  { search: true },
  // Determines language drop down position among links
  { languages: true }
],
```

`organizationName` [string]

GitHub username of the organization or user hosting this project. This is used by the publishing script to determine where your GitHub pages website will be hosted.

`projectName` [string]

Project name. This must match your GitHub repository project name (case-sensitive).

`tagline` [string]

The tagline for your website.

`title` [string]

Title for your website.

### `url` [string]

URL for your website. This can also be considered the top-level hostname. For example, `https://facebook.github.io` is the URL of https://facebook.github.io/metro/, and `https://docusaurus.io` is the URL for https://docusaurus.io. This field is related to the `baseUrl` field.

## Optional Fields

### `algolia` [object]

Information for Algolia search integration. If this field is excluded, the search bar will not appear in the header. You must specify two values for this field, and one ( `appId` ) is optional.

- `apiKey` - the Algolia provided an API key for your search.
- `indexName` - the Algolia provided index name for your search (usually this is the project name)
- `appId` - Algolia provides a default scraper for your docs. If you provide your own, you will probably get this id from them.

### `blogSidebarCount` [number]

Control the number of blog posts that show up in the sidebar. See the adding a blog docs for more information.

### `blogSidebarTitle` [string]

Control the title of the blog sidebar. See the adding a blog docs for more information.

### `cleanUrl` [boolean]

If `true`, allow URLs with no `HTML` extension. For example, a request to URL https://docusaurus.io/docs/installation will return the same result as https://docusaurus.io/docs/installation.html.

> If users intend for this website to be used exclusively offline, this value must be set to `false`. Otherwise, it will cause the site to route to the parent folder of the linked page.

### `cname` [string]

The CNAME for your website. It will go into a `CNAME` file when your site is built.

### `customDocsPath` [string]

By default, Docusaurus expects your documentation to be in a directory called `docs`. This directory is at the same level as the `website` directory (i.e., not inside the `website` directory). You can specify a custom path to your documentation with this field.

```
customDocsPath: 'docs/site';
```

```
customDocsPath: 'website-docs';
```

`defaultVersionShown` [string]

The default version for the site to be shown. If this is not set, the latest version will be shown.

`deletedDocs` [object]

Even if you delete the main file for a documentation page and delete it from your sidebar, the page will still be created for every version and for the current version due to fallback functionality. This can lead to confusion if people find the documentation by searching and it appears to be something relevant to a particular version but actually is not.

To force removal of content beginning with a certain version (including for current/next), add a `deletedDocs` object to your config, where each key is a version and the value is an array of document IDs that should not be generated for that version and all later versions.

Example:

```
{
  deletedDocs: {
    "2.0.0": [
      "tagging"
    ]
  }
}
```

The version keys must match those in `versions.json`. Assuming the versions list in `versions.json` is `["3.0.0", "2.0.0", "1.1.0", "1.0.0"]`, the `docs/1.0.0/tagging` and `docs/1.1.0/tagging` URLs will work but `docs/2.0.0/tagging`, `docs/3.0.0/tagging`, and `docs/tagging` will not. The files and folders for those versions will not be generated during the build.

`docsUrl` [string]

The base URL for all docs file. Set this field to `''` to remove the `docs` prefix of the documentation URL. If unset, it is defaulted to `docs`.

#### `disableHeaderTitle` [boolean]

An option to disable showing the title in the header next to the header icon. Exclude this field to keep the header as normal, otherwise set to `true`.

#### `disableTitleTagline` [boolean]

An option to disable showing the tagline in the title of main pages. Exclude this field to keep page titles as `Title • Tagline`. Set to `true` to make page titles just `Title`.

#### `docsSideNavCollapsible` [boolean]

Set this to `true` if you want to be able to expand/collapse the links and subcategories in the sidebar.

#### `editUrl` [string]

URL for editing docs, usage example: `editUrl + 'en/doc1.md'`. If this field is omitted, there will be no "Edit this Doc" button for each document.

#### `enableUpdateBy` [boolean]

An option to enable the docs showing the author who last updated the doc. Set to `true` to show a line at the bottom right corner of each doc page as `Last updated by <Author Name>`.

#### `enableUpdateTime` [boolean]

An option to enable the docs showing last update time. Set to `true` to show a line at the bottom right corner of each doc page as `Last updated on <date>`.

#### `facebookAppId` [string]

If you want Facebook Like/Share buttons in the footer and at the bottom of your blog posts, provide a Facebook application id.

#### `facebookComments` [boolean]

Set this to `true` if you want to enable Facebook comments at the bottom of your blog post. `facebookAppId` has to be also set.

#### `facebookPixelId` [string]

Facebook Pixel ID to track page views.

`fonts` [object]

Font-family CSS configuration for the site. If a font family is specified in `siteConfig.js` as `$myFont`, then adding a `myFont` key to an array in `fonts` will allow you to configure the font. Items appearing earlier in the array will take priority of later elements, so ordering of the fonts matter.

In the below example, we have two sets of font configurations, `myFont` and `myOtherFont`. `Times New Roman` is the preferred font in `myFont`. `-apple-system` is the preferred in `myOtherFont`.

```
fonts: {
  myFont: [
    'Times New Roman',
    'Serif'
  ],
  myOtherFont: [
    '-apple-system',
    'system-ui'
  ]
},
```

The above fonts would be represented in your CSS file(s) as variables `$myFont` and `$myOtherFont`.

```
h1 {
  font-family: $myFont;
}
```

`footerIcon` [string]

URL for a footer icon. Currently used in the `core/Footer.js` file provided as an example, but it can be removed from that file.

`gaTrackingId` [string]

Google Analytics tracking ID to track page views.

`gaGtag` [boolean]

Set this to `true` if you want to use global site tags (gtag.js) for Google analytics instead of `analytics.js`.

`githubHost` [string]

The hostname of your server. Useful if you are using GitHub Enterprise.

`highlight`

Syntax highlighting options:

```
{
  // ...
  highlight: {
    // The name of the theme used by Highlight.js when highlighting code.
    // You can find the list of supported themes here:
    // https://github.com/isagalaev/highlight.js/tree/master/src/styles
    theme: 'default',

    // The particular version of Highlight.js to be used.
    version: '9.12.0',

    // Escape valve by passing an instance of Highlight.js to the function specified here
    hljs: function(highlightJsInstance) {
      // do something here
    },

    // Default language.
    // It will be used if one is not specified at the top of the code block. You can find
    // https://github.com/isagalaev/highlight.js/tree/master/src/languages

    defaultLang: 'javascript',

    // custom URL of CSS theme file that you want to use with Highlight.js. If this is pr
    themeUrl: 'http://foo.bar/custom.css'
  },
}
```

`manifest` [string]

Path to your web app manifest (e.g., `manifest.json`). This will add a `<link>` tag to `<head>` with `rel` as `"manifest"` and `href` as the provided path.

`markdownOptions` [object]

Override default Remarkable options that will be used to render markdown.

> To manage syntax extensions, use the `markdownPlugins` field.

`markdownPlugins` [array]

An array of plugins to be loaded by Remarkable, the markdown parser and renderer used by Docusaurus. The plugin will receive a reference to the Remarkable instance, allowing custom parsing and rendering rules to be defined.

For example, if you want to enable superscript and subscript in your markdown that is rendered by Remarkable to HTML, you would do the following:

```
markdownPlugins: [
  function foo(md) {
    md.inline.ruler.enable(['sub', 'sup']);
  },
],
```

`noIndex` [boolean]

Boolean. If true, Docusaurus will politely ask crawlers and search engines to avoid indexing your site. This is done with a header tag and so only applies to docs and pages. Will not attempt to hide static resources. This is a best effort request. Malicious crawlers can and will still index your site.

`ogImage` [string]

Local path to an Open Graph image (e.g., `img/myImage.png`). This image will show up when your site is shared on Facebook and other websites/apps where the Open Graph protocol is supported.

`onPageNav` [string]

If you want a visible navigation option for representing topics on the current page. Currently, there is one accepted value for this option:

- `separate` - The secondary navigation is a separate pane defaulting on the right side of a document. See http://docusaurus.io/docs/en/translation.html for an example.

`scripts` [array]

An array of JavaScript sources to load. The values can be either strings or plain objects of attribute-value maps. Refer to the example below. The script tag will be inserted in the HTML head.

`separateCss` [array]

Directories inside which any `css` files will not be processed and concatenated to Docusaurus' styles. This is to support static `HTML` pages that may be separate from Docusaurus with completely separate styles.

`scrollToTop` [boolean]

Set this to `true` if you want to enable the scroll to top button at the bottom of your site.

`scrollToTopOptions` [object]

Optional options configuration for the scroll to top button. You do not need to use this, even if you set `scrollToTop` to `true`; it just provides you more configuration control of the button. You can find more options here. By default, we set the zIndex option to 100.

`slugPreprocessor` [function]

Define the slug preprocessor function if you want to customize the text used for generating the hash links. Function provides the base string as the first argument and must always return a string.

`stylesheets` [array]

An array of CSS sources to load. The values can be either strings or plain objects of attribute-value maps. The link tag will be inserted in the HTML head.

`translationRecruitingLink` [string]

URL for the `Help Translate` tab of language selection when languages besides English are enabled. This can be included you are using translations but does not have to be.

`twitter` [boolean]

Set this to `true` if you want a Twitter social button to appear at the bottom of your blog posts.

`twitterUsername` [string]

If you want a Twitter follow button at the bottom of your page, provide a Twitter username to follow. For example: `docusaurus`.

`twitterImage` [string]

Local path to your Twitter card image (e.g., `img/myImage.png`). This image will show up on the Twitter card when your site is shared on Twitter.

`useEnglishUrl` [string]

If you do not have translations enabled (e.g., by having a `languages.js` file), but still want a link of the form `/docs/en/doc.html` (with the `en`), set this to `true`.

`users` [array]

The `users` array mentioned earlier.

`usePrism` [array]

An array of languages to use Prism syntax highlighter. Refer to Using Prism as additional syntax highlighter. Set it to `true` to use Prism on all languages.

`wrapPagesHTML` [boolean]

Boolean flag to indicate whether `HTML` files in `/pages` should be wrapped with Docusaurus site styles, header and footer. This feature is experimental and relies on the files being `HTML` fragments instead of complete pages. It inserts the contents of your `HTML` file with no extra processing. Defaults to `false`.

Users can also add their own custom fields if they wish to provide some data across different files.

## Adding Google Fonts

Google Fonts offers faster load times by caching fonts without forcing users to sacrifice privacy. For more information on Google Fonts, see the Google Fonts documentation.

To add Google Fonts to your Docusaurus deployment, add the font path to the `siteConfig.js` under `stylesheets`:

```
stylesheets: [
  'https://fonts.googleapis.com/css?family=Source+Sans+Pro:400,400i,700',
],
```

## Example siteConfig.js with many available fields

```
const users = [
  {
    caption: 'User1',
    image: '/test-site/img/docusaurus.svg',
    infoLink: 'https://www.example.com',
    pinned: true,
  },
];

const siteConfig = {
  title: 'Docusaurus',
  tagline: 'Generate websites!',
  url: 'https://docusaurus.io',
  baseUrl: '/',
  // For github.io type URLS, you would combine the URL and baseUrl like:
```

```
  // url: 'https://reasonml.github.io',
  // baseUrl: '/reason-react/',
  defaultVersionShown: '1.0.0',
  organizationName: 'facebook',
  projectName: 'docusaurus',
  noIndex: false,
  // For no header links in the top nav bar -> headerLinks: [],
  headerLinks: [
    {doc: 'doc1', label: 'Docs'},
    {page: 'help', label: 'Help'},
    {search: true},
    {blog: true},
  ],
  headerIcon: 'img/docusaurus.svg',
  favicon: 'img/favicon.png',
  colors: {
    primaryColor: '#2E8555',
    secondaryColor: '#205C3B',
  },
  editUrl: 'https://github.com/facebook/docusaurus/edit/master/docs/',
  // Users variable set above
  users,
  disableHeaderTitle: true,
  disableTitleTagline: true,
  separateCss: ['static/css/non-docusaurus', 'static/assets/separate-css'],
  footerIcon: 'img/docusaurus.svg',
  translationRecruitingLink: 'https://crowdin.com/project/docusaurus',
  algolia: {
    apiKey: '0f9f28b9ab9efae89810921a351753b5',
    indexName: 'github',
  },
  gaTrackingId: 'UA-12345678-9',
  highlight: {
    theme: 'default',
  },
  markdownPlugins: [
    function foo(md) {
      md.renderer.rules.fence_custom.foo = function (
        tokens,
        idx,
        options,
        env,
        instance,
      ) {
        return '<div class="foo">bar</div>';
      };
    },
  ],
    scripts: [
      'https://docusaurus.io/slash.js',
      {
        src:
          'https://cdnjs.cloudflare.com/ajax/libs/clipboard.js/2.0.0/clipboard.min.js',
        async: true,
      },
    ],
    stylesheets: [
      'https://docusaurus.io/style.css',
      {
        href: 'http://css.link',
        type: 'text/css',
      },
    ],
    facebookAppId: '1615782811974223',
    facebookComments: true,
    facebookPixelId: '352490515235776',
    twitter: 'true',
    twitterUsername: 'docusaurus',
    twitterImage: 'img/docusaurus.png',
    ogImage: 'img/docusaurus.png',
    cleanUrl: true,
    scrollToTop: true,
    scrollToTopOptions: {
      zIndex: 100,
    },
    // Remove the HTML tags and HTML tags content before generating the slug
    slugPreprocessor: (slugBase) =>
      slugBase.replace(/<([^>]+?)([^>]*?)>(.*?)<\/\1>/gi, ''),
  };

  module.exports = siteConfig;
```