


Zowe Documentation



Table of contents:

- [Introduction](#)
 - [Fast Track](#) 
 - [Disclaimer](#)
 - [Features](#)
 - [Design principles](#)
 - [Comparison with other tools](#)
 - [Gatsby](#)
 - [Next.js](#)
 - [VuePress](#)
 - [MkDocs](#)
 - [Docsify](#)
 - [GitBook](#)
 - [Jekyll](#)
 - [Staying informed](#)
 - [Something missing?](#)
- [Installation](#)
 - [Requirements](#)
 - [Scaffold project website](#)
 - [Project structure](#)
 - [Project structure rundown](#)
 - [Running the development server](#)
 - [Build](#)
 - [Updating your Docusaurus version](#)
 - [Problems?](#)
- [Configuration](#)
 - [What goes into a docusaurus.config.js?](#)
 - [Site metadata](#)
 - [Deployment configurations](#)
 - [Theme, plugin, and preset configurations](#)
 - [Custom configurations](#)
 - [Accessing configuration from components](#)
 - [Customizing Babel Configuration](#)
- [TypeScript Support](#)
 - [Setup](#)
 - [Swizzling TypeScript theme components](#)
- [Creating Pages](#)
 - [Add a React page](#)
 - [Add a Markdown page](#)

- [Routing](#)
- [Using React](#)
- [Duplicate Routes](#)
- [Docs Introduction](#)
 - [Document ID](#)
 - [Home page docs](#)
 - [Docs-only mode](#)
- [Create a doc](#)
 - [Hello from Docusaurus](#)
 - [Headers](#)
 - [Only h2 and h3 will be in the toc](#)
- [Sidebar](#)
 - [Default sidebar](#)
 - [Sidebar object](#)
 - [Using multiple sidebars](#)
 - [Understanding sidebar items](#)
 - [Doc: link to a doc](#)
 - [Ref: link to a doc, without sidebar](#)
 - [Link: link to any page](#)
 - [Category: create a hierarchy](#)
 - [Autogenerated: generate a sidebar](#)
 - [Hideable sidebar](#)
 - [Passing custom props](#)
 - [Complex sidebars example](#)
- [Versioning](#)
 - [Directory structure](#)
 - [Tagging a new version](#)
 - [Docs](#)
 - [Creating new docs](#)
 - [Linking docs](#)
 - [Versions](#)
 - [Updating an existing version](#)
 - [Deleting an existing version](#)
 - [Recommended practices](#)
 - [Figure out the behavior for the "current" version](#)
 - [Version your documentation only when needed](#)
 - [Keep the number of versions small](#)
 - [Use absolute import within the docs](#)
 - [Global or versioned colocated assets](#)
- [Docs Markdown Features](#)
 - [Markdown frontmatter](#)
 - [Referencing other documents](#)
- [Docs Multi-instance](#)
 - [Use-cases](#)
 - [Mobile SDKs documentation](#)
 - [Versioned and unversioned doc](#)
 - [Setup](#)
 - [Versioned paths](#)
 - [Tagging new versions](#)
 - [Docs navbar items](#)
- [Blog](#)
 - [Initial setup](#)
 - [Adding posts](#)
 - [Header options](#)
 - [Summary truncation](#)

- [Feed](#)
- [Advanced topics](#)
 - [Blog-only mode](#)
 - [Multiple blogs](#)
- [Markdown Features introduction](#)
- [Using React](#)
- [Tabs](#)
 - [Syncing tab choices](#)
 - [Customizing tabs](#)
- [Code blocks](#)
 - [Code title](#)
 - [Syntax highlighting](#)
 - [Line highlighting](#)
 - [Interactive code editor](#)
 - [It is 13:30:21.](#)
 - [Imports](#)
 - [Multi-language support code blocks](#)
- [Admonitions](#)
 - [Specifying title](#)
- [Headings](#)
 - [Markdown headings](#)
 - [Heading ids](#)
 - [Generated ids](#)
 - [Explicit ids](#)
- [Inline TOC](#)
 - [Full table of contents](#)
 - [Custom table of contents](#)
 - [Example Section 1](#)
 - [Example Subsection 1 a](#)
 - [Example Subsection 1 b](#)
 - [Example Subsection 1 c](#)
 - [Example Section 2](#)
 - [Example Subsection 2 a](#)
 - [Example Subsection 2 b](#)
 - [Example Subsection 2 c](#)
 - [Example Section 3](#)
 - [Example Subsection 3 a](#)
 - [Example Subsection 3 b](#)
 - [Example Subsection 3 c](#)
- [Assets](#)
 - [Images](#)
 - [Files](#)
 - [Inline SVGs](#)
 - [Themed Images](#)
- [Plugins](#)
 - [Configuring plugins](#)
 - [Configuring plugin options](#)
- [Styling and Layout](#)
 - [Traditional CSS](#)
 - [Styling your site with Infima](#)
 - [Dark Mode](#)
 - [Styling approaches](#)
 - [Global styles](#)
 - [CSS modules](#)
 - [CSS-in-JS](#)

- [Sass/SCSS](#)
- [Static Assets](#)
 - [Referencing your static asset](#)
 - [JSX example](#)
 - [Markdown example](#)
 - [Caveats](#)
- [Search](#)
 - [Using Algolia DocSearch](#)
 - [Connecting Algolia](#)
 - [Contextual search](#)
 - [Custom Application ID](#)
 - [Styling your Algolia search](#)
 - [Customizing the Algolia search behavior](#)
 - [Editing the Algolia search component](#)
 - [Using your own search](#)
- [Browser support](#)
 - [Purpose](#)
 - [Default values](#)
 - [Read more](#)
- [Deployment](#)
 - [Testing Build Local](#)
 - [Self Hosting](#)
 - [Deploying to GitHub Pages](#)
 - [docusaurus.config.js settings](#)
 - [Environment settings](#)
 - [Deploy](#)
 - [Triggering deployment with GitHub Actions](#)
 - [Triggering deployment with Travis CI](#)
 - [Using Azure Pipelines](#)
 - [Using Drone](#)
 - [Deploying to Netlify](#)
 - [Deploying to Vercel](#)
 - [Deploying to Render](#)
 - [Deploying to Qovery](#)
 - [Deploying to Hostman](#)
 - [Deploying to Surge](#)
 - [Using your domain](#)
 - [Setting up CNAME file](#)
 - [Deploying to QuantCDN](#)
- [i18n - Introduction](#)
 - [Goals](#)
 - [i18n goals](#)
 - [i18n non-goals](#)
 - [Translation workflow](#)
 - [Overview](#)
 - [Translation files](#)
 - [Translation files location](#)
- [i18n - Tutorial](#)
 - [Configure your site](#)
 - [Site configuration](#)
 - [Theme configuration](#)
 - [Start your site](#)
 - [Translate your site](#)
 - [Use the translation APIs](#)
 - [Translate JSON files](#)

- [Translate Markdown files](#)
- [Use explicit heading ids](#)
- [Deploy your site](#)
 - [Single-domain deployment](#)
 - [Multi-domain deployment](#)
 - [Hybrid](#)
- [i18n - Using git](#)
 - [Tradeoffs](#)
 - [Git tutorial](#)
 - [Prepare the Docusaurus site](#)
 - [Initialize the i18n folder](#)
 - [Translate the files](#)
 - [Repeat](#)
 - [Maintain the translations](#)
 - [Markdown translations](#)
 - [JSON translations](#)
 - [Localize edit urls](#)
- [i18n - Using Crowdin](#)
 - [Crowdin overview](#)
 - [Crowdin tutorial](#)
 - [Prepare the Docusaurus site](#)
 - [Create a Crowdin project](#)
 - [Create the Crowdin configuration](#)
 - [Install the Crowdin CLI](#)
 - [Upload the sources](#)
 - [Translate the sources](#)
 - [Download the translations](#)
 - [Automate with CI](#)
 - [Advanced Crowdin topics](#)
 - [MDX](#)
 - [Docs versioning](#)
 - [Multi-instance plugins](#)
 - [Maintaining your site](#)
 - [VCS \(Git\) integrations](#)
 - [In-Context localization](#)
 - [Localize edit urls](#)
 - [Example configuration](#)
- [Plugins](#)
 - [Available plugins](#)
 - [Installing a plugin](#)
 - [Configuring plugins](#)
 - [Multi-instance plugins and plugin ids](#)
 - [Plugins design](#)
 - [Creating plugins](#)
 - [Module definition](#)
- [Themes](#)
 - [Available themes](#)
 - [Using themes](#)
 - [Theme components](#)
 - [Wrapping your site with <Root>](#)
 - [Swizzling theme components](#)
 - [Wrapping theme components](#)
 - [For site owners](#)
 - [For plugin authors](#)
 - [Themes design](#)

- [Writing customized Docusaurus themes](#)
- [Presets](#)
 - [Using presets](#)
 - [Presets -> themes and plugins](#)
 - [Official presets](#)
 - [@docusaurus/preset-classic](#)
 - [@docusaurus/preset-bootstrap](#)
- [Migration overview](#)
 - [Main differences](#)
 - [Docusaurus 1 structure](#)
 - [Docusaurus 2 structure](#)
 - [Migration process](#)
 - [Automated migration process](#)
 - [Manual migration process](#)
 - [Support](#)
 - [Example migration PRs](#)
- [Automated migration](#)
- [Manual migration](#)
 - [Project setup](#)
 - [package.json](#)
 - [Update references to the build directory](#)
 - [.gitignore](#)
 - [README](#)
 - [Site configurations](#)
 - [docusaurus.config.js](#)
 - [Updated fields](#)
 - [Removed fields](#)
 - [Urls](#)
 - [Components](#)
 - [Sidebar](#)
 - [Footer](#)
 - [Pages](#)
 - [Content](#)
 - [Replace AUTOGENERATED TABLE OF CONTENTS](#)
 - [Update Markdown syntax to be MDX-compatible](#)
 - [Language-specific code tabs](#)
 - [Front matter](#)
 - [Deployment](#)
 - [Test your site](#)
- [Versioned sites](#)
 - [Migrate your versioned_docs front matter](#)
 - [Migrate your versioned_sidebars](#)
 - [Populate your versioned_sidebars and versioned_docs](#)
 - [Convert style attributes to style objects in MDX](#)
- [Translated sites](#)
 - [i18n differences](#)
 - [Different filesystem paths](#)
 - [Updated translation APIs](#)
 - [Stricter Markdown parser](#)
 - [Migration strategies](#)
 - [Create a new Crowdin project](#)
 - [Use the existing Crowdin project](#)
 - [Use Git instead of Crowdin](#)

Introduction

⚡ Docusaurus will help you ship a **beautiful documentation site in no time**.

🔧 Building a custom tech stack is expensive. Instead, **focus on your content** and just write Markdown files.

🔗 Ready for more? Use **advanced features** like versioning, i18n, search and theme customizations.

📖 Check the [best Docusaurus sites](#) for inspiration and read some [testimonials](#).

🕒 Docusaurus is a **static-site generator**. It builds a **single-page application** with a fast client-side navigation, leveraging the full power of **React** to make your site interactive. It provides out-of-the-box **documentation features**, but can be used to create **any kind of site** (personal website, product, blog, marketing landing pages, etc).

Fast Track 🕒#

Understand Docusaurus in **5 minutes** by playing!

Create a new Docusaurus site and follow the **very short** embedded tutorial.

Install [Node.js](#) and create a new Docusaurus site:

Copy

Start the site:

Copy

Open <http://localhost:3000> and follow the tutorial.

💡 tip

Use new.docusaurus.io to test Docusaurus immediately in your browser!

Or read the [5 minutes tutorial](#) online.

Disclaimer#

Docusaurus v2 is **beta** but already quite stable and widely used.

We highly encourage you to **use Docusaurus v2 over Docusaurus v1**, as Docusaurus v1 will be deprecated soon.

A [lot of users](#) are already using Docusaurus v2 ([trends](#)).

Use **Docusaurus v2** if:

- You want a modern Jamstack documentation site
- You want a single-page application (SPA) with client-side routing
- You want the full power of React and MDX
- You do not need support for IE11

Use [Docusaurus v1](#) if:

- ✗ You don't want a single-page application (SPA)
- ✗ You need support for IE11

Features#

Docusaurus is built with high attention to the developer and contributor experience.

- 📦 **Built with ❤️ and React**
 - Extend and customize with React
 - Gain full control of your site's browsing experience by providing your own React components
- **Pluggable**
 - Bootstrap your site with a basic template, then use advanced features and plugins
 - Open source your plugins to share with the community
- ✂️ **Developer experience**
 - Start writing your docs right now
 - Universal configuration entry point to make it more maintainable by contributors
 - Hot reloading with lightning fast incremental build on changes
 - Route-based code and data splitting
 - Publish to GitHub Pages, Netlify, Vercel and other deployment services with ease

Our shared goal — to help your users find what they need fast, and understand your products better. We share with you our best practices helping you build your doc site right and well.

- 🌐 **SEO friendly**
 - HTML files are statically generated for every possible path
 - page-specific SEO to help your users land on your official docs directly relating their problems at hand
- 📄 **Powered by MDX**
 - Write interactive components via JSX and React embedded in markdown
 - Share your code in live editors to get your users love your products on the spot
- 🔍 **Search** - Your full site is searchable
- 📅 **Document Versioning** - Helps you keep documentation in sync with project releases.
- 🌍 **i18n** - Translate your site in multiple locales

Docusaurus 2 is born to be compassionately accessible to all your users, and lightning fast.

- ⚡ **Lightning fast** - Docusaurus 2 follows the [PRPL Pattern](#) that makes sure your content loads blazing fast
- 🙋 **Accessible** - Attention to accessibility, making your site equally accessible to all users

Design principles#

- **Little to learn** - Docusaurus should be easy to learn and use as the API is quite small. Most things will still be achievable by users, even if it takes them more code and more time to write. Not having abstractions is better than having the wrong abstractions, and we don't want users to have to hack around the wrong abstractions. Mandatory talk - [Minimal API Surface Area](#)
- **Intuitive** - Users will not feel overwhelmed when looking at the project directory of a Docusaurus project or adding new features. It should look intuitive and easy to build on top of, using approaches they are familiar with.
- **Layered architecture** - The separations of concerns between each layer of our stack (content/theming/styling)

should be clear - well-abstracted and modular.

- **Sensible defaults** - Common and popular performance optimizations and configurations will be done for users but they are given the option to override them.
- **No vendor-lock in** - Users are not required to use the default plugins or CSS, although they are highly encouraged to. Certain core lower-level infra level pieces like React Loadable, React Router cannot be swapped because we do default performance optimization on them. But not higher level ones, such as choice of Markdown engines, CSS frameworks, CSS methodology will be entirely up to users.

We believe that as developers, knowing how a library works is helpful in allowing us to become better at using it. Hence we're dedicating effort into explaining the architecture and various components of Docusaurus with the hope that users reading it will gain a deeper understanding of the tool and be even more proficient in using it.

Comparison with other tools#

Across all static site generators, Docusaurus has a unique focus on documentation sites and has many out-of-the-box features.

We've also studied other main static site generators and would like to share our insights on the comparison, hopefully to help you navigate through the prismatic choices out there.

Gatsby#

[Gatsby](#) is packed with a lot of features, has a rich ecosystem of plugins and is capable of doing everything that Docusaurus does. Naturally, that comes at a cost of a higher learning curve. Gatsby does many things well and is suitable for building many types of websites. On the other hand, Docusaurus tries to do one thing super well - be the best tool for writing and publishing content.

GraphQL is also pretty core to Gatsby, although you don't necessarily need GraphQL to build a Gatsby site. In most cases when building static websites, you won't need the flexibility that GraphQL provides.

Many aspects of Docusaurus 2 were inspired by the best things about Gatsby and it's a great alternative.

[Dokz](#) is a Gatsby theme to build documentation website. It is currently less featured than Docusaurus.

Next.js#

[Next.js](#) is another very popular hybrid React framework. It can help you build a good documentation website, but it is not opinionated toward the documentation use-case, and it will require a lot more work to implement what Docusaurus provides out-of-the-box.

[Nextra](#) is an opinionated static-site-generator built on top of Next.js. It is currently less featured than Docusaurus.

VuePress#

[VuePress](#) has many similarities with Docusaurus - both focus heavily on content-centric website and provides tailored documentation features out of the box. However, VuePress is powered by Vue, while Docusaurus is powered by React. If you want a Vue-based solution, VuePress would be a decent choice.

MkDocs#

[MkDocs](#) is a popular Python static-site-generator with value proposition similar to Docusaurus.

It is a good option if you don't need a single-page application, and don't plan to leverage React.

[Material for MkDocs](#) is a beautiful theme.

Docsify#

[Docsify](#) makes it easy to create a documentation website, but is not a static-site generator and is not SEO friendly.

GitBook#

[GitBook](#) has very clean design and has been used by many open source projects. With its focus shifting towards a commercial product rather than an open-source tool, many of its requirements no longer fit the needs as an open source project's documentation site. As a result, many have turned to other products. You may read about Redux's switch to Docusaurus [here](#).

Currently, GitBook is only free for open-source and non-profit teams. Docusaurus is free for everyone.

Jekyll#

[Jekyll](#) is one of the most mature static site generators around and has been a great tool to use — in fact, before Docusaurus, most of Facebook's Open Source websites are/were built on Jekyll! It is extremely simple to get started. We want to bring a similar developer experience as building a static site with Jekyll.

In comparison with statically generated HTML and interactivity added using `<script />` tags, Docusaurus sites are React apps. Using modern JavaScript ecosystem tooling, we hope to set new standards on doc sites performance, asset build pipeline and optimizations, and ease to setup.

Staying informed#

- [GitHub](#)
- [Twitter](#)
- [Blog](#)
- [Discord](#)

Something missing?#

If you find issues with the documentation or have suggestions on how to improve the documentation or the project in general, please [file an issue](#) for us, or send a tweet mentioning the [@docusaurus](#) Twitter account.

For new feature requests, you can create a post on our [Canny board](#), which is a handy tool for roadmapping and allows for sorting by upvotes, which gives the core team a better indicator of what features are in high demand, as compared to GitHub issues which are harder to triage. Refrain from making a Pull Request for new features (especially large ones) as someone might already be working on it or will be part of our roadmap. Talk to us first!

Version: 2.0.0-beta.0

Installation

Docusaurus is essentially a set of npm [packages](#)

💡 tip

Use the [Fast Track](#) to understand Docusaurus in 5 minutes 🕒!

Use new.docusaurus.io to test Docusaurus immediately in your browser!

Requirements#

- [Node.js](#) version $\geq 12.13.0$ or above (which can be checked by running `node -v`). You can use [nvm](#) for managing multiple Node versions on a single machine installed
- [Yarn](#) version ≥ 1.5 (which can be checked by running `yarn --version`). Yarn is a performant package manager for JavaScript and replaces the `npm` client. It is not strictly necessary but highly encouraged.

Scaffold project website#

The easiest way to install Docusaurus is to use the command line tool that helps you scaffold a skeleton Docusaurus website. You can run this command anywhere in a new empty repository or within an existing repository, it will create a new directory containing the scaffolded files.

Copy

Example:

Copy

If you do not specify `name` or `template`, it will prompt you for them. We recommend the `classic` template so that you can get started quickly and it contains features found in Docusaurus 1. The `classic` template contains `@docusaurus/preset-classic` which includes standard documentation, a blog, custom pages, and a CSS framework (with dark mode support). You can get up and running extremely quickly with the classic template and customize things later on when you have gained more familiarity with Docusaurus.

[FB-Only]: If you are setting up a new Docusaurus website for a Facebook open source project, use the `facebook` template instead, which comes with some useful Facebook-specific defaults:

Copy

[Experimental]: If you want setting up a new website using [bootstrap](#) use the `bootstrap` template, like the following:

Copy

If you want to skip installing dependencies, use the `--skip-install` option, like the following:

Copy

Project structure#

Assuming you chose the classic template and named your site `my-website`, you will see the following files generated under a new directory `my-website/`:

Copy

Project structure rundown#

- `/blog/` - Contains the blog Markdown files. You can delete the directory if you do not want/need a blog. More details can be found in the [blog guide](#)
- `/docs/` - Contains the Markdown files for the docs. Customize the order of the docs sidebar in `sidebars.js`.

More details can be found in the [docs guide](#)

- `/src/` - Non-documentation files like pages or custom React components. You don't have to strictly put your non-documentation files in here but putting them under a centralized directory makes it easier to specify in case you need to do some sort of linting/processing
 - `/src/pages` - Any files within this directory will be converted into a website page. More details can be found in the [pages guide](#)
- `/static/` - Static directory. Any contents inside here will be copied into the root of the final `build` directory
- `/docusaurus.config.js` - A config file containing the site configuration. This is the equivalent of `siteConfig.js` in Docusaurus v1
- `/package.json` - A Docusaurus website is a React app. You can install and use any npm packages you like in them
- `/sidebar.js` - Used by the documentation to specify the order of documents in the sidebar

Running the development server#

To preview your changes as you edit the files, you can run a local development server that will serve your website and reflect the latest changes.

- npm
- Yarn

Copy

By default, a browser window will open at `http://localhost:3000`.

Congratulations! You have just created your first Docusaurus site! Browse around the site to see what's available.

Build#

Docusaurus is a modern static website generator so we need to build the website into a directory of static contents and put it on a web server so that it can be viewed. To build the website:

- npm
- Yarn

Copy

and contents will be generated within the `/build` directory, which can be copied to any static file hosting service like [GitHub pages](#), [Vercel](#) or [Netlify](#). Check out the docs on [deployment](#) for more details.

Updating your Docusaurus version#

There are many ways to update your Docusaurus version. One guaranteed way is to manually change the version number in `package.json` to the desired version. Note that all `@docusaurus/-namespaced` packages should be using the same version.

 important

Please update to the latest Docusaurus 2 version shown at the top of the page, not what is shown below.

package.json

Copy

Then, in the directory containing `package.json`, run your package manager's install command:

- `npm`
- `Yarn`

Copy

To check that the update occurred successfully, run:

- `npm`
- `Yarn`

Copy

You should see the correct version as output.

Alternatively, if you are using Yarn, you can do:

Copy

Problems?#

Ask for help on [Stack Overflow](#), on our [GitHub repository](#) or [Twitter](#)

Version: 2.0.0-beta.0

Configuration

Docusaurus has a unique take on configurations. We encourage you to congregate information of your site into one place. We guard the fields of this file, and facilitate making this data object accessible across your site.

Keeping a well-maintained `docusaurus.config.js` helps you, your collaborators, and your open source contributors be able to focus on documentation while still being able to customize the site.

What goes into a `docusaurus.config.js`?#

You should not have to write your `docusaurus.config.js` from scratch even if you are developing your site. All templates come with a `docusaurus.config.js` that includes defaults for the common options.

However, it can be helpful if you have a high-level understanding of how the configurations are designed and implemented.

The high-level overview of Docusaurus configuration can be categorized into:

- [Site metadata](#)
- [Deployment configurations](#)
- [Theme, plugin, and preset configurations](#)
- [Custom configurations](#)

For exact reference to each of the configurable fields, you may refer to [docusaurus.config.js API reference](#).

Site metadata#

Site metadata contains the essential global metadata such as `title`, `url`, `baseUrl` and `favicon`.

They are used in a number of places such as your site's title and headings, browser tab icon, social sharing (Facebook, Twitter) information or even to generate the correct path to serve your static files.

Deployment configurations#

Deployment configurations such as `projectName` and `organizationName` are used when you deploy your site with the `deploy` command.

It is recommended to check the [deployment docs](#) for more information.

Theme, plugin, and preset configurations#

List the [theme](#) [plugins](#) and [presets](#) for your site in the `themes`, `plugins`, and `presets` fields, respectively. These are typically npm packages:

```
docusaurus.config.js
```

Copy

They can also be loaded from local directories:

```
docusaurus.config.js
```

Copy

To specify options for a plugin or theme, replace the name of the plugin or theme in the config file with an array containing the name and an options object:

```
docusaurus.config.js
```

Copy

To specify options for a plugin or theme that is bundled in a preset, pass the options through the `presets` field. In this example, `docs` refers to `@docusaurus/plugin-content-docs` and `theme` refers to `@docusaurus/theme-classic`.

```
docusaurus.config.js
```

Copy

For further help configuring themes, plugins, and presets, see [Using Themes](#), [Using Plugins](#) and [Using Presets](#).

Custom configuration:#

Docusaurus guards `docusaurus.config.js` from unknown fields. To add custom fields, define them in `customFields`.

Example:

```
docusaurus.config.js
```

Copy

Accessing configuration from components#

Your configuration object will be made available to all the components of your site. And you may access them via React context as `siteConfig`.

Basic example:

Copy

 tip

If you just want to use those fields on the client side, you could create your own JS files and import them as ES6 modules, there is no need to put them in `docusaurus.config.js`.

Customizing Babel Configuration#

For new Docusaurus projects, we automatically generated a `babel.config.js` in project root.

`babel.config.js`

Copy

Most of the times, this configuration will work just fine. If you want to customize it, you can directly edit this file to customize babel configuration. For your changes to take effect, you need to restart Docusaurus devserver.

Version: 2.0.0-beta.0

TypeScript Support

Setup#

Docusaurus supports writing and using TypeScript theme components. To start using TypeScript, add `@docusaurus/module-type-aliases` and some `@types` dependencies to your project:

- npm
- Yarn

Copy

Then add `tsconfig.json` to your project root with the following content:

`tsconfig.json`

Copy

Docusaurus doesn't use this `tsconfig.json` to compile your project. It is added just for a nicer Editor experience, although you can choose to run `tsc` to type check your code for yourself or on CI.

Now you can start writing TypeScript theme components.

Swizzling TypeScript theme components#

For themes that supports TypeScript theme components, you can add the `--typescript` flag to the end of swizzling command to get TypeScript source code. For example, the following command will generate `index.tsx` and `styles.module.css` into `src/theme/Footer`.

- npm
- Yarn

Copy

At this moment, the only official Docusaurus theme that supports TypeScript theme components is `@docusaurus/theme-classic`. If you are a Docusaurus theme package author who wants to add TypeScript support, see the [Lifecycle APIs docs](#)

Version: 2.0.0-beta.0

Creating Pages

In this section, we will learn about creating pages in Docusaurus.

This is useful for creating **one-off standalone pages** like a showcase page, playground page or support page.

The functionality of pages is powered by `@docusaurus/plugin-content-pages`.

You can use React components, or Markdown.

📌note

Pages do not have sidebars, only [docs](#) have.

Add a React page#

Create a file `/src/pages/helloReact.js`:

```
/src/pages/helloReact.js
```

Copy

Once you save the file, the development server will automatically reload the changes. Now open `http://localhost:3000/helloReact`, you will see the new page you just created.

Each page doesn't come with any styling. You will need to import the `Layout` component from `@theme/Layout` and wrap your contents within that component if you want the navbar and/or footer to appear.

💡tip

You can also create TypeScript pages with the `.tsx` extension (`helloReact.tsx`).

Add a Markdown page#

Create a file `/src/pages/helloMarkdown.md`:

```
/src/pages/helloMarkdown.md
```

Copy

In the same way, a page will be created at `http://localhost:3000/helloMarkdown`.

Markdown pages are less flexible than React pages, because it always uses the theme layout.

Here's an [example Markdown page](#)



You can use the full power of React in Markdown pages too, refer to the [MDX](#) documentation.

Routing#

If you are familiar with other static site generators like Jekyll and Next, this routing approach will feel familiar to you. Any JavaScript file you create under `/src/pages/` directory will be automatically converted to a website page, following the `/src/pages/` directory hierarchy. For example:

- `/src/pages/index.js` → `<baseUrl>`
- `/src/pages/foo.js` → `<baseUrl>/foo`
- `/src/pages/foo/test.js` → `<baseUrl>/foo/test`
- `/src/pages/foo/index.js` → `<baseUrl>/foo/`

In this component-based development era, it is encouraged to co-locate your styling, markup and behavior together into components. Each page is a component, and if you need to customize your page design with your own styles, we recommend co-locating your styles with the page component in its own directory. For example, to create a "Support" page, you could do one of the following:

- Add a `/src/pages/support.js` file
- Create a `/src/pages/support/` directory and a `/src/pages/support/index.js` file.

The latter is preferred as it has the benefits of letting you put files related to the page within that directory. For example, a CSS module file (`styles.module.css`) with styles meant to only be used on the "Support" page. **Note:** this is merely a recommended directory structure and you will still need to manually import the CSS module file within your component module (`support/index.js`). By default, any Markdown or Javascript file starting with `_` will be ignored, and no routes will be created for that file (see the `exclude` option).

Copy



All JavaScript/TypeScript files within the `src/pages/` directory will have corresponding website paths generated for them. If you want to create reusable components into that directory, use the `exclude` option (by default, files prefixed with `_`, test files (`.test.js`) and files in `__tests__` directory are not turned into pages).

Using React#

React is used as the UI library to create pages. Every page component should export a React component, and you can leverage on the expressiveness of React to build rich and interactive content.

Duplicate Routes#

You may accidentally create multiple pages that are meant to be accessed on the same route. When this happens, Docusaurus will warn you about duplicate routes when you run `yarn start` or `yarn build`, but the site will still be built successfully. The page that was created last will be accessible, but it will override other conflicting pages. To resolve this issue, you should modify or remove any conflicting routes.

Version: 2.0.0-beta.0

Docs Introduction

The docs feature provides users with a way to organize Markdown files in a hierarchical format.

Document ID#

Every document has a unique `id`. By default, a document `id` is the name of the document (without the extension) relative to the root docs directory.

For example, `greeting.md` `id` is `greeting` and `guide/hello.md` `id` is `guide/hello`.

Copy

However, the **last part** of the `id` can be defined by user in the front matter. For example, if `guide/hello.md`'s content is defined as below, its final `id` is `guide/part1`.

Copy

If you want more control over the last part of the document URL, it is possible to add a `slug` (defaults to the `id`).

Copy

 note

It is possible to use:

- absolute slugs: `slug: /mySlug`, `slug: /...`
- relative slugs: `slug: mySlug`, `slug: ../../mySlug...`

Home page docs#

If you want a document to be available at the root, and have a path like `https://docusaurus.io/docs/`, you can use the `slug` frontmatter:

Copy

Docs-only mode#

If you only want the documentation feature, you can run your Docusaurus 2 site without a landing page and display your documentation page as the index page instead.

To enable docs-only mode, set the docs plugin `routeBasePath: '/'`, and use the frontmatter `slug: /` on the document that should be the index page ([more infos](#)).

 caution

You should delete the existing homepage at `./src/pages/index.js`, or else there will be two files mapping to the same route!

 tip

There's also a "blog-only mode" for those who only want to use the blog feature of Docusaurus 2. You can use the same method detailed above. Follow the setup instructions on [Blog-only mode](#).

Version: 2.0.0-beta.0

Create a doc

Create a markdown file, `greeting.md`, and place it under the `docs` directory.

Copy

At the top of the file, specify `id` and `title` in the front matter, so that Docusaurus will pick them up correctly when generating your site.

Copy

This will render in the browser as follows:

`http://localhost:3000`

Hello from Docusaurus

Are you ready to create the documentation site for your open source project?

Headers

will show up on the table of contents on the upper right

So that your users will know what this page is all about without scrolling down or even without reading too much.

Only h2 and h3 will be in the toc

The headers are well-spaced so that the hierarchy is clear.

- lists will help you
- present the key points
- that you want your users to remember
 - and you may nest them
 - multiple times

Version: 2.0.0-beta.0

Sidebar

Creating a sidebar is useful to:

- Group multiple **related documents**
- **Display a sidebar** on each of those documents
- Provide a **paginated navigation**, with next/previous button

To use sidebars on your Docusaurus site:

1. Define a file that exports a [sidebar object](#).
2. Pass this object into the `@docusaurus/plugin-docs` plugin directly or via `@docusaurus/preset-classic`.

docusaurus.config.js

Copy

Default sidebar <#>

By default, Docusaurus [automatically generates a sidebar](#) for you, by using the filesystem structure of the docs folder:

sidebars.js

Copy

You can also define your sidebars explicitly.

Sidebar object <#>

A sidebar is a tree of [sidebar items](#).

Copy

A sidebars file can contain **multiple sidebar objects**.

Copy

Example:

sidebars.js

Copy

Notice the following:

- There is a single sidebar `mySidebar`, containing 5 [sidebar items](#)
- `Getting Started` and `Docusaurus` are sidebar categories
- `doc1`, `doc2` and `doc3` are sidebar documents

 tip

Use the **shorthand syntax** to express this sidebar more concisely:

sidebars.js

Copy

Using multiple sidebars <#>

You can create a sidebar for each **set of markdown files** that you want to **group together**.

 tip

The Docusaurus site is a good example of using multiple sidebars:

- [Docs](#)
- [API](#)

Example:

```
sidebars.js
```

Copy

📌_{note}

The keys `tutorialSidebar` and `apiSidebar` are sidebar **technical ids** and do not matter much.

When browsing:

- `doc1` or `doc2`: the `tutorialSidebar` will be displayed
- `doc3` or `doc4`: the `apiSidebar` will be displayed

A **paginated navigation** link documents inside the same sidebar with **next** and **previous** buttons.

Understanding sidebar items#

`sidebarItem` is an item defined in a `Sidebar` tree.

There are different types of sidebar items:

- **Doc**: link to a doc page, assigning it to the sidebar
- **Ref**: link to a doc page, without assigning it to the sidebar
- **Link**: link to any internal or external page
- **Category**: create a hierarchy of sidebar items
- **Autogenerated**: generate a sidebar slice automatically

Doc: link to a doc#

Use the `doc` type to link to a doc page and assign that doc to a sidebar:

Copy

Example:

```
sidebars.js
```

Copy

The `sidebar_label` markdown frontmatter has a higher precedence over the `label` key in `SidebarItemDoc`.

📌_{note}

Don't assign the same doc to multiple sidebars: use a **ref** instead.

Ref: link to a doc, without sidebar#

Use the `ref` type to link to a doc page without assigning it to a sidebar.

Copy

Example:

```
sidebars.js
```

Copy

When browsing `doc1`, Docusaurus will not display the `mySidebar` sidebar.

Link: link to any page <#>

Use the `link` type to link to any page (internal or external) that is not a doc.

Copy

Example:

```
sidebars.js
```

Copy

Category: create a hierarchy <#>

Use the `category` type to create a hierarchy of sidebar items.

Copy

Example:

```
sidebars.js
```

Copy

Tip

Use the shorthand syntax when you don't need category options:

```
sidebars.js
```

Copy

Collapsible categories <#>

For sites with a sizable amount of content, we support the option to expand/collapse a category to toggle the display of its contents. Categories are collapsible by default. If you want them to be always expanded, set

```
themeConfig.sidebarCollapsible to false:
```

```
docusaurus.config.js
```

Copy

Expanded categories by default <#>

For docs that have collapsible categories, you may want more fine-grain control over certain categories. If you want specific categories to be always expanded, you can set `collapsed` to `false`:

```
sidebars.js
```

Copy

Autogenerated: generate a sidebar <#>

Docusaurus can create a **sidebar automatically** from your **filesystem structure**: each folder creates a sidebar category.

An autogenerated item is converted by Docusaurus to a **sidebar slice**: a list of items of type `doc` and `category`.

Copy

Docusaurus can generate a sidebar from your docs folder:

```
sidebars.js
```

Copy

You can also use **multiple autogenerated items** in a sidebar, and interleave them with regular sidebar items:

```
sidebars.js
```

Copy

Autogenerated sidebar metadata: <#>

By default, the sidebar slice will be generated in **alphabetical order** (using files and folders names).

If the generated sidebar does not look good, you can assign additional metadatas to docs and categories.

For docs: use additional frontmatter:

```
docs/tutorials/tutorial-easy.md
```

Copy

For categories: add a `_category_.json` or `_category_.yaml` file in the appropriate folder:

```
docs/tutorials/_category_.json
```

Copy

```
docs/tutorials/_category_.yaml
```

Copy

ⓘ info

The position metadata is only used **inside a sidebar slice**: Docusaurus does not re-order other items of your sidebar.

Using number prefixes: <#>

A simple way to order an autogenerated sidebar is to prefix docs and folders by number prefixes:

Copy

To make it **easier to adopt**, Docusaurus supports **multiple number prefix patterns**.

By default, Docusaurus will **remove the number prefix** from the doc id, title, label and url paths.

⚠ caution

Prefer using [additional metadatas](#).

Updating a number prefix can be annoying, as it can require **updating multiple existing markdown links**:

docs/02-Tutorial Easy/01-First Part.md

Copy

Customize the sidebar items generator#

You can provide a custom `sidebarItemsGenerator` function in the docs plugin (or preset) config:

docsaurus.config.js

Copy

 tip

Re-use and enhance the default generator instead of writing a generator from scratch.

Add, update, filter, re-order the sidebar items according to your use-case:

docsaurus.config.js

Copy

Hideable sidebar#

Using the enabled `themeConfig.hideableSidebar` option, you can make the entire sidebar hidden, allowing you to better focus your users on the content. This is especially useful when content consumption on medium screens (e.g. on tablets).

docsaurus.config.js

Copy

Passing custom props#

To pass in custom props to a swizzled sidebar item, add the optional `customProps` object to any of the items:

Copy

Complex sidebars example#

Real-world example from the Docusaurus site:

sidebars.js

Copy

Version: 2.0.0-beta.0

Versioning

You can use the `version` script to create 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.

 caution

Think about it before starting to version your documentation - it can become difficult for contributors to help improve it!

Most of the time, you don't need versioning as it will just increase your build time, and introduce complexity to your codebase. Versioning is **best suited for websites with high-traffic and rapid changes to documentation between versions**. If your documentation rarely changes, don't add versioning to your documentation.

To better understand how versioning works and see if it suits your needs, you can read on below.

Directory structure#

Copy

The table below explains how a versioned file maps to its version and the generated URL.

| Path | Version | URL |
|--|----------------|--------------------------------|
| <code>versioned_docs/version-1.0.0/hello.md</code> | 1.0.0 | <code>/docs/1.0.0/hello</code> |
| <code>versioned_docs/version-1.1.0/hello.md</code> | 1.1.0 (latest) | <code>/docs/hello</code> |
| <code>docs/hello.md</code> | next | <code>/docs/next/hello</code> |

Tagging a new version#

1. First, make sure your content in the `docs` directory is ready to be frozen as a version. A version always should be based from `master`.
2. Enter a new version number.
 - npm
 - Yarn

Copy

When tagging a new version, the document versioning mechanism will:

- Copy the full `docs/` folder contents into a new `versioned_docs/version-<version>/` folder.
- Create a versioned `sidebars` file based from your current [sidebar](#) configuration (if it exists) - saved as `versioned_sidebars/version-<version>-sidebars.json`.
- Append the new version number to `versions.json`.

Docs#

Creating new docs#

1. Place the new file into the corresponding version folder.
2. Include the reference for the new file into the corresponding sidebar file, according to version number.

Master docs

Copy

Older docs

Copy

Linking docs#

- Remember to include the `.md` extension.
- Files will be linked to correct corresponding version.
- Relative paths work as well.

Copy

Versions#

Each directory in `versioned_docs/` will represent a documentation version.

Updating an existing version#

You can update multiple docs versions at the same time because each directory in `versioned_docs/` represents specific routes when published.

1. Edit any file.
2. Commit and push changes.
3. It will be published to the version.

Example: When you change any file in `versioned_docs/version-2.6/`, it will only affect the docs for version 2.6.

Deleting an existing version#

You can delete/remove versions as well.

1. Remove the version from `versions.json`.

Example:

Copy

2. Delete the versioned docs directory. Example: `versioned_docs/version-1.8.0`.
3. Delete the versioned sidebars file. Example: `versioned_sidebars/version-1.8.0-sidebars.json`.

Recommended practices#

Figure out the behavior for the "current" version#

The "current" version is the version name for the `./docs` folder.

There are different ways to manage versioning, but two very common patterns are:

- You release v1, and start immediately working on v2 (including its docs)
- You release v1, and will maintain it for some time before thinking about v2.

Docusaurus defaults work great for the first usecase.

For the 2nd usecase: if you release v1 and don't plan to work on v2 anytime soon, instead of versioning v1 and having to maintain the docs in 2 folders (`./docs` + `./versioned_docs/version-1.0.0`), you may consider using the following configuration instead:

Copy

The docs in `./docs` will be served at `/docs/1.0.0` instead of `/docs/next`, and `1.0.0` will become the default version we link to in the navbar dropdown, and you will only need to maintain a single `./docs` folder.

See [docs plugin configuration](#) for more details.

Version your documentation only when needed#

For example, you are building a documentation for your npm package `foo` and you are currently in version `1.0.0`. You then release a patch version for a minor bug fix and it's now `1.0.1`.

Should you cut a new documentation version `1.0.1`? **You probably shouldn't.** `1.0.1` and `1.0.0` docs shouldn't differ according to semver because there are no new features!. Cutting a new version for it will only just create unnecessary duplicated files.

Keep the number of versions small#

As a good rule of thumb, try to keep the number of your versions below 10. **It is very likely** that you will have a lot of obsolete versioned documentation that nobody even reads anymore. For example, [Jest](#) is currently in version `24.9`, and only maintains several latest documentation version with the lowest being `22.x`. Keep it small 😊

Use absolute import within the docs#

Don't use relative paths import within the docs. Because when we cut a version the paths no longer work (the nesting level is different, among other reasons). You can utilize the `@site` alias provided by docusaurus, that points to the `website` directory. Example:

Copy

Global or versioned colocated assets#

You should decide if assets like images and files are per version or shared between versions

If your assets should be versioned, put them in the docs version, and use relative paths:

Copy

If your assets are global, put them in `/static` and use absolute paths:

Copy

Version: `2.0.0-beta.0`

Docs Markdown Features

Docs can use any [Markdown feature](#) and have a few additional Docs-specific markdown features.

Markdown frontmatter#

Markdown docs have their own [Markdown frontmatter](#)

Referencing other documents#

If you want to reference another document file, you could use the name of the document you want to reference. Docusaurus will convert the file path to be the final website path (and remove the `.md`).

For example, if you are in `doc2.md` and you want to reference `doc1.md` and `folder/doc3.md`:

Copy

One benefit of this approach is that the links to external files will still work if you are viewing the file on GitHub.

Another benefit, for versioned docs, is that one versioned doc will link to another doc of the exact same version.

Version: 2.0.0-beta.0

Docs Multi-instance

The `@docusaurus/plugin-content-docs` plugin can support [multi-instance](#).

 **note**

This feature is only useful for [versioned documentations](#). It is recommended to be familiar with docs versioning before reading this page.

Use-cases#

Sometimes you want a Docusaurus site to host 2 distinct sets of documentation (or more).

These documentations may even have different versioning/release lifecycles.

Mobile SDKs documentation#

If you build a cross-platform mobile SDK, you may have 2 documentations:

- Android SDK documentation (v1.0, v1.1)
- iOS SDK documentation (v1.0, v2.0)

In such case, you can use a distinct docs plugin instance per mobile SDK documentation.

 **caution**

If each documentation instance is very large, you should rather create 2 distinct Docusaurus sites.

If someone edits the iOS documentation, is it really useful to rebuild everything, including the whole Android documentation that did not change?

Versioned and unversioned doc#

Sometimes, you want some documents to be versioned, while other documents are more "global", and it feels useless to version them.

We use this pattern on the Docusaurus website itself:

- The [/docs/*](#) section is versioned
- The [/community/*](#) section is unversioned

Setup#

Suppose you have 2 documentations:

- Product: some versioned doc about your product
- Community: some unversioned doc about the community around your product

In this case, you should use the same plugin twice in your site configuration.

⚠caution

@docusaurus/preset-classic already includes a docs plugin instance for you!

When using the preset:

```
docusaurus.config.js
```

Copy

When not using the preset:

```
docusaurus.config.js
```

Copy

Don't forget to assign a unique `id` attribute to plugin instances.

📌note

We consider that the `product` instance is the most important one, and make it the "default" instance by not assigning any `id`.

Versioned paths#

Each plugin instance will store versioned docs in a distinct folder.

The default plugin instance will use these paths:

- `website/versions.json`
- `website/versioned_docs`
- `website/versioned_sidebars`

The other plugin instances (with an `id` attribute) will use these paths:

- `website/<pluginId>_versions.json`
- `website/<pluginId>_versioned_docs`
- `website/<pluginId>_versioned_sidebars`

💡tip

You can omit the `id` attribute (defaults to `default`) for one of the docs plugin instances.

The instance paths will be simpler, and retro-compatible with a single-instance setup.

Tagging new versions#

Each plugin instance will have its own cli command to tag a new version. They will be displayed if you run:

- npm
- Yarn

Copy

To version the product/default docs plugin instance:

- npm
- Yarn

Copy

To version the non-default/community docs plugin instance:

- npm
- Yarn

Copy

Docs navbar items#

Each docs-related [theme navbar items](#) take an optional `docsPluginId` attribute.

For example, if you want to have one version dropdown for each mobile SDK (iOS and Android), you could do:

```
docusaurus.config.js
```

Copy

```
Version: 2.0.0-beta.0
```

Blog

Initial setup#

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

Then, add an item link to your blog within `docusaurus.config.js`:

```
docusaurus.config.js
```

Copy

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 `my-website/blog/2019-09-05-hello-docusaurus-v2.md`:

Copy

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 author name to be displayed.
- `author_url` - The URL that the author's name will be linked to. This could be a GitHub, Twitter, Facebook profile URL, etc.
- `author_image_url` - The URL to the author's thumbnail image.
- `author_title` - A description of the author.
- `title` - The blog post title.
- `tags` - A list of strings to tag to your post.
- `draft` - A boolean flag to indicate that the blog post is work-in-progress and therefore should not be published yet. However, draft blog posts will be displayed during development.
- `description`: The description of your post, 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.
- `image`: Cover or thumbnail image that will be used when displaying the link to your post.
- `hide_table_of_contents`: Whether to hide the table of contents to the right. By default it is `false`.

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:

Copy

Feed#

You can generate RSS/Atom feed by passing `feedOptions`. By default, RSS and Atom feeds are generated. To disable feed generation, set `feedOptions.type` to `null`.

Copy

Example usage:

```
docusaurus.config.js
```

Copy

Accessing the feed:

The feed for RSS can be found at:

Copy

and for Atom:

Copy

Advanced topics#

Blog-only mode#

You can run your Docusaurus 2 site without a landing page and instead have your blog's post list page as the index page. Set the `routeBasePath` to be `'/'` to indicate it's the root path.

```
docusaurus.config.js
```

Copy

⚠caution

Don't forget to delete the existing homepage at `./src/pages/index.js` or else there will be two files mapping to the same route!

You can also add meta description to the blog list page for better SEO:

```
docusaurus.config.js
```

Copy

Multiple blogs#

By default, the classic theme assumes only one blog per website and hence includes only one instance of the blog plugin. If you would like to have multiple blogs on a single website, it's possible too! You can add another blog by specifying another blog plugin in the `plugins` option for `docusaurus.config.js`.

Set the `routeBasePath` to the URL route that you want your second blog to be accessed on. Note that the `routeBasePath` here has to be different from the first blog or else there could be a collision of paths! Also, set `path` to the path to the directory containing your second blog's entries.

As documented for [multi-instance plugins](#), you need to assign a unique id to the plugins.

```
docusaurus.config.js
```

Copy

As an example, we host a second blog [here](#)

Version: 2.0.0-beta.0

Markdown Features introduction

Documentation is one of your product's interfaces with your users. A well-written and well-organized set of docs helps your users understand your product quickly. Our aligned goal here is to help your users find and understand the information they need, as quickly as possible.

Docusaurus 2 uses modern tooling to help you compose your interactive documentations with ease. You may embed React components, or build live coding blocks where your users may play with the code on the spot. Start sharing your eureka moments with the code your audience cannot walk away from. It is perhaps the most effective way of attracting potential users.

In this section, we'd like to introduce you to the tools we've picked that we believe will help you build a powerful documentation. Let us walk you through with an example.

Markdown is a syntax that enables you to write formatted content in a readable syntax.

The [standard Markdown syntax](#) is supported, and we use [MDX](#) as the parsing engine, which can do much more than

Just parsing markdown, like rendering react components inside your documents.

📌important

This section assumes you are using the official Docusaurus content plugins.

Version: 2.0.0-beta.0

Using React

Docusaurus has built-in support for [MDX](#), which allows you to write JSX within your Markdown files and render them as React components.

📌note

While both `.md` and `.mdx` files are parsed using MDX, some of the syntax are treated slightly differently. For the most accurate parsing and better editor support, we recommend using the `.mdx` extension for files containing MDX syntax.

Try this block here:

Copy

Notice how it renders both the markup from your React component and the Markdown syntax:

`http://localhost:3000`

`Docusaurus green` and `Facebook blue` are my favorite colors.

I can write **Markdown** alongside my *JSX*!

You can also import your own components defined in other files or third-party components installed via npm! Check out the [MDX docs](#) to see what other fancy stuff you can do with MDX.

⚠caution

Since all doc files are parsed using MDX, any HTML is treated as JSX. Therefore, if you need to inline-style a component, follow JSX flavor and provide style objects. This behavior is different from Docusaurus 1. See also [Migrating from v1 to v2](#).

Version: 2.0.0-beta.0

Tabs

To show tabbed content within Markdown files, you can fall back on MDX. Docusaurus provides `<Tabs>` components out-of-the-box.

Copy

And you will get the following:

- Apple
- Orange
- Banana

This is an apple 🍏

📄 info

By default, tabs are rendered eagerly, but it is possible to load them lazily by passing the `lazy` prop to the `Tabs` component.

Syncing tab choices#

You may want choices of the same kind of tabs to sync with each other. For example, you might want to provide different instructions for users on Windows vs users on macOS, and you want to changing all OS-specific instructions tabs in one click. To achieve that, you can give all related tabs the same `groupId` prop. Note that doing this will persist the choice in `localStorage` and all `<Tab>` instances with the same `groupId` will update automatically when the value of one of them is changed. Note that `groupId` are globally-namespaced.

Copy

- Windows
- macOS

Use Ctrl + C to copy.

- Windows
- macOS

Use Ctrl + V to paste.

For all tab groups that have the same `groupId`, the possible values do not need to be the same. If one tab group with chooses an value that does not exist in another tab group with the same `groupId`, the tab group with the missing value won't change its tab. You can see that from the following example. Try to select Linux, and the above tab groups doesn't change.

Copy

- Windows
- macOS
- Linux

I am Windows.

Tab choices with different `groupIds` will not interfere with each other:

Copy

- Windows
- macOS

Windows in windows.

- Windows
- Unix

Windows is windows.

Customizing tabs#

You might want to customize the appearance of certain set of tabs. To do that you can pass the string in `className` prop and the specified CSS class will be added to the `Tabs` component:

Copy

- Apple
- Orange
- Banana

This is an apple 🍏
Version: 2.0.0-beta.0

Code blocks

Code blocks within documentation are super-powered .

Code title#

You can add a title to the code block by adding `title` key after the language (leave a space between them).

Copy

```
/src/components/HelloCodeTitle.js
```

Copy

Syntax highlighting#

Code blocks are text blocks wrapped around by strings of 3 backticks. You may check out [this reference](#) for specifications of MDX.

Copy

Use the matching language meta string for your code block, and Docusaurus will pick up syntax highlighting automatically, powered by [Prism React Renderer](#)

Copy

By default, the Prism [syntax highlighting theme](#) we use is [Palenight](#). You can change this to another theme by

passing `theme` field in `prism.js` file according in your `docusaurus.config.js`.

For example, if you prefer to use the `dracula` highlighting theme:

```
docusaurus.config.js
```

Copy

By default, Docusaurus comes with a subset of [commonly used languages](#)

⚠caution

Some popular languages like Java, C#, or PHP are not enabled by default.

To add syntax highlighting for any of the other [Prism supported languages](#) define it in an array of additional languages.

For example, if you want to add highlighting for the `powershell` language:

```
docusaurus.config.js
```

Copy

If you want to add highlighting for languages not yet supported by Prism, you can swizzle `prism-include-languages`:

- `npm`
- `Yarn`

Copy

It will produce `prism-include-languages.js` in your `src/theme` folder. You can add highlighting support for custom languages by editing `prism-include-languages.js`:

```
src/theme/prism-include-languages.js
```

Copy

You can refer to [Prism's official language definitions](#) when you are writing your own language definitions.

Line highlighting#

You can bring emphasis to certain lines of code by specifying line ranges after the language meta string (leave a space after the language).

Copy

Copy

To accomplish this, Docusaurus adds the `docusaurus-highlight-code-line` class to the highlighted lines. You will need to define your own styling for this CSS, possibly in your `src/css/custom.css` with a custom background color which is dependent on your selected syntax highlighting theme. The color given below works for the default highlighting theme (Palenight), so if you are using another theme, you will have to tweak the color accordingly.

```
/src/css/custom.css
```

Copy

To highlight multiple lines, separate the line numbers by commas or use the range syntax to select a chunk of lines.

This feature uses the `parse-number-range` library and you can find [more syntax](#) on their project details.

Copy

Copy

You can also use comments with `highlight-next-line`, `highlight-start`, and `highlight-end` to select which lines are highlighted.

Copy

Copy

Supported commenting syntax:

| Language | Syntax |
|------------|--|
| JavaScript | <code>/* ... */</code> and <code>// ...</code> |
| JSX | <code>{/* ... */}</code> |
| Python | <code># ...</code> |
| HTML | <code><!-- ... --></code> |

If there's a syntax that is not currently supported, we are open to adding them! Pull requests welcome.

Interactive code editor[#]

(Powered by [React Live](#))

You can create an interactive coding editor with the `@docusaurus/theme-live-codeblock` plugin.

First, add the plugin to your package.

- npm
- Yarn

Copy

You will also need to add the plugin to your `docusaurus.config.js`.

Copy

To use the plugin, create a code block with `live` attached to the language meta string.

Copy

The code block will be rendered as an interactive editor. Changes to the code will reflect on the result panel live.

Live Editor

```
function Clock(props) {
  const [date, setDate] = useState(new Date());
  useEffect(() => {
    var timerID = setInterval(() => tick(), 1000);

    return function cleanup() {
      clearInterval(timerID);
    };
  });
};
```

```
function tick() {
  setDate(new Date());
}

return (
  <div>
    <h2>It is {date.toLocaleTimeString()}.</h2>
  </div>
);
}
```

Result

It is 13:30:21.

Imports#

react-live and imports

It is not possible to import components directly from the react-live code editor, you have to define available imports upfront.

By default, all React imports are available. If you need more imports available, swizzle the react-live scope:

- npm
- Yarn

Copy

src/theme/ReactLiveScope/index.js

Copy

The `ButtonExample` component is now available to use:

Live Editor

```
function MyPlayground(props) {
  return (
    <div>
      <ButtonExample onClick={() => alert('hey!')}>Click me</ButtonExample>
    </div>
  );
}
```

Result

Click me

Multi-language support code blocks#

With MDX, you can easily create interactive components within your documentation, for example, to display code in multiple programming languages and switching between them using a tabs component.

Instead of implementing a dedicated component for multi-language support code blocks, we've implemented a generic `Tabs` component in the classic theme so that you can use it for other non-code scenarios as well.

The following example is how you can have multi-language code tabs in your docs. Note that the empty lines above

and below each language block is **intentional**. This is a current limitation of MDX, you have to leave empty lines around Markdown syntax for the MDX parser to know that it's Markdown syntax and not JSX.

Copy

And you will get the following:

- JavaScript
- Python
- Java

Copy

You may want to implement your own `<MultiLanguageCode />` abstraction if you find the above approach too verbose. We might just implement one in future for convenience.

If you have multiple of these multi-language code tabs, and you want to sync the selection across the tab instances, refer to the [Syncing tab choices section](#)

Version: 2.0.0-beta.0

Admonitions

In addition to the basic Markdown syntax, we use [remark-admonitions](#) alongside MDX to add support for admonitions. Admonitions are wrapped by a set of 3 colons.

Example:

Copy

①_{note}

The content and title *can* include markdown.

📌 You can specify an optional title

Heads up! Here's a pro-tip.

①_{info}

Useful information.

⚠_{caution}

Warning! You better pay attention!

🔥_{danger}

Danger danger, mayday!

Specifying title#

You may also specify an optional title

Copy

① Your Title

The content and title *can* include Markdown.

Version: 2.0.0-beta.0

Headings

Markdown headings#

You can use regular Markdown headings.

Copy

Markdown headings appear as a table-of-contents entry.

Heading ids#

Each heading has an id that can be automatically generated, or explicitly specified.

Heading ids allow you to link to a specific document heading in Markdown or JSX:

Copy

Copy

Generated ids#

By default, Docusaurus will generate heading ids for you, based on the heading text.

```
### Hello world will have id hello-world.
```

Generated ids have **some limits**:

- The id might not look good
- You might want to **change or translate** the text without updating the existing id

Explicit ids#

A special Markdown syntax lets you set an **explicit heading id**:

Copy

💡 tip

Use the [write-heading-ids](#) CLI command to add explicit ids to all your Markdown documents.

Version: 2.0.0-beta.0

Inline TOC

Each markdown document displays a tab of content on the top-right corner.

But it is also possible to display an inline table of contents directly inside a markdown document, thanks to MDX.

Full table of contents#

The `toc` variable is available in any MDX document, and contain all the top level headings of a MDX document.

Copy

- [Full table of contents](#)
- [Custom table of contents](#)
- [Example Section 1](#)
 - [Example Subsection 1 a](#)
 - [Example Subsection 1 b](#)
 - [Example Subsection 1 c](#)
- [Example Section 2](#)
 - [Example Subsection 2 a](#)
 - [Example Subsection 2 b](#)
 - [Example Subsection 2 c](#)
- [Example Section 3](#)
 - [Example Subsection 3 a](#)
 - [Example Subsection 3 b](#)
 - [Example Subsection 3 c](#)

Custom table of contents#

The `toc` props is just a list of table of contents items:

Copy

You can create this TOC tree manually, or derive a new TOC tree from the `toc` variable:

Copy

- [Example Section 1](#)
 - [Example Subsection 1 a](#)
 - [Example Subsection 1 b](#)
 - [Example Subsection 1 c](#)
- [Example Section 3](#)
 - [Example Subsection 3 a](#)
 - [Example Subsection 3 b](#)
 - [Example Subsection 3 c](#)

caution

The underlying content is just an example to have more table-of-contents items available in current page.

Example Section 1#

Lorem ipsum

Example Subsection 1 a <#>

Lorem ipsum

Example Subsection 1 b <#>

Lorem ipsum

Example Subsection 1 c <#>

Lorem ipsum

Example Section 2 <#>

Lorem ipsum

Example Subsection 2 a <#>

Lorem ipsum

Example Subsection 2 b <#>

Lorem ipsum

Example Subsection 2 c <#>

Lorem ipsum

Example Section 3 <#>

Lorem ipsum

Example Subsection 3 a <#>

Lorem ipsum

Example Subsection 3 b <#>

Lorem ipsum

Example Subsection 3 c <#>

Lorem ipsum

Version: 2.0.0-beta.0

Assets

Sometimes you want to link to static assets directly from Markdown files, and it is convenient to co-locate the asset next to the markdown file using it.

We have setup Webpack loaders to handle most common file types, so that when you import a file, you get its url, and the asset is automatically copied to the output folder.

Let's imagine the following file structure:

Copy

Images#

You can use images in Markdown, or by requiring them and using a JSX image tag:

Copy

The ES imports syntax also works:

Copy

This results in displaying the image:

①note

If you are using [@docusaurus/plugin-ideal-image](#), you need to use the dedicated image component, as documented.

Files#

In the same way, you can link to existing assets by requiring them and using the returned url in videos, links etc.

Copy

[Download this PDF](#)

[Download this PDF using Markdown](#)

Inline SVGs#

Docusaurus supports inlining SVGs out of the box.

Copy





This can be useful, if you want to alter the part of the SVG image via CSS. For example, you can change one of the SVG colors based on the current theme.

Copy

Copy



Themed Images#

Docusaurus supports themed images: the `ThemedImage` component (included in the classic/bootstrap themes) allows you to switch the image source based on the current theme.

Copy

Version: 2.0.0-beta.0

Plugins

You can expand the MDX functionalities, using plugins.

Docusaurus content plugins support both [Remark](#) and [Rehype](#) plugins that work with MDX.

Configuring plugins#

An MDX plugin is usually a npm package, so you install them like other npm packages using npm.

First, install your [Remark](#) and [Rehype](#) plugins.

For example:

- npm
- Yarn

Copy

Next, import the plugins:

Copy

Finally, add them to the `@docusaurus/preset-classic` options in `docusaurus.config.js`:

```
docusaurus.config.js
```

Copy

Configuring plugin options#

Some plugins can be configured and accept their own options. In that case, use the `[plugin, pluginOptions]` syntax, like so:

```
docusaurus.config.js
```

Copy

See more information in the [MDX documentation](#).

Version: 2.0.0-beta.0

Styling and Layout

Traditional CSS#

If you're using `@docusaurus/preset-classic`, you can create your own CSS files (e.g. `/src/css/custom.css`) and import them globally by passing it as an option into the preset.

```
docusaurus.config.js
```

Copy

Any CSS you write within that file will be available globally and can be referenced directly using string literals. This is the most traditional approach to writing CSS and is fine for small websites that do not have much customization.

Styling your site with Infima#

`@docusaurus/preset-classic` uses [Infima](#) as the underlying styling framework. Infima provides flexible layout and common UI components styling suitable for content-centric websites (blogs, documentation, landing pages). For more details, check out the [Infima website](#).

When you `init` your Docusaurus 2 project, the website will be generated with basic Infima stylesheets and default styling. You may customize the styling by editing the `/src/css/custom.css` file.

```
/src/css/custom.css
```

Copy

Infima uses 7 shades of each color. We recommend using [ColorBox](#) to find the different shades of colors for your chosen primary color.

Alternatively, use the following tool to generate the different shades for your website and copy the variables into `/src/css/custom.css`.

Primary Color:

| CSS Variable Name | Hex | Adjustment |
|------------------------------|---------|------------|
| --ifm-color-primary-lightest | #80aaef | -30 |
| --ifm-color-primary-lighter | #5a91ea | -15 |
| --ifm-color-primary-light | #4e89e8 | -10 |
| --ifm-color-primary | #3578e5 | 0 |
| --ifm-color-primary-dark | #1d68e1 | 10 |
| --ifm-color-primary-darker | #1b62d4 | 15 |
| --ifm-color-primary-darkest | #1751af | 30 |

Replace the variables in `src/css/custom.css` with these new variables.

Copy

Dark Mode#

To customize the Infima variables for dark mode you can add the following to `src/css/custom.css`.

```
/src/css/custom.css
```

Copy

Styling approaches#

A Docusaurus site is a single-page React application. You can style it the way you style React apps.

There are a few approaches/frameworks which will work, depending on your preferences and the type of website you are trying to build. Websites that are highly interactive and behave more like web apps will benefit from a more modern styling approaches that co-locate styles with the components. Component styling can also be particularly useful when you wish to customize or swizzle a component.

Global styles#

This is the most traditional way of styling that most developers (including non-front end developers) would be familiar with.

Assuming you are using `@docusaurus/preset-classic` and `/src/css/custom.css` was passed in to the preset config, styles inside that file would be available globally.

```
/src/css/custom.css
```

Copy

Copy

Theme Class Names#

We provide some predefined CSS class names to provide access for developers to style layout of a page globally in Docusaurus. The purpose is to have stable classnames shared by all themes that are meant to be targeted by custom CSS.

Copy

CSS modules#

To style your components using [CSS Modules](#), name your stylesheet files with the `.module.css` suffix (e.g. `welcome.module.css`). Webpack will load such CSS files as CSS modules and you have to reference the class names from the imported CSS module (as opposed to using plain strings). This is similar to the convention used in [Create React App](#)

```
styles.module.css
```

Copy

Copy

The class names which will be processed by webpack into a globally unique class name during build.

CSS-in-JS#

⚠caution

This section is a work in progress. [Welcoming PRs](#).

Sass/SCSS#

To use Sass/SCSS as your CSS preprocessor, install the unofficial Docusaurus 2 plugin [docusaurus-plugin-sass](#). This plugin works for both global styles and the CSS modules approach:

1. Install [docusaurus-plugin-sass](#):

- npm
- Yarn

Copy

2. Include the plugin in your `docusaurus.config.js` file:

```
docusaurus.config.js
```

Copy

3. Write and import your stylesheets in Sass/SCSS as normal.

Global styles using Sass/SCSS#

You can now set the `customCss` property of `@docusaurus/preset-classic` to point to your Sass/SCSS file:

```
docusaurus.config.js
```

Copy

Modules using Sass/SCSS#

Name your stylesheet files with the `.module.scss` suffix (e.g. `welcome.module.scss`) instead of `.css`. Webpack will use `sass-loader` to preprocess your stylesheets and load them as CSS modules.

```
styles.module.scss
```

Copy

Copy

Version: 2.0.0-beta.0

Static Assets

Every website needs assets: images, stylesheets, favicons etc. In such cases, you can create a directory named `static` at the root of your project.

Every file you put into **that directory will be copied** into the root of the generated `build` folder with the directory hierarchy preserved. E.g. if you add a file named `sun.jpg` to the `static` folder, it will be copied to `build/sun.jpg`.

This means that:

- for site `baseUrl: '/'`, the image `/static/img/docusaurus.png` will be served at `/img/docusaurus.png`.
- for site `baseUrl: '/subpath/'`, the image `/static/img/docusaurus.png` will be served at `/subpath/img/docusaurus.png`.

Referencing your static asset#

You can reference assets from the `static` folder in your code using absolute paths, but this is not ideal because changing the site `baseUrl` will **break those links**.

You can `import / require()` the static asset (recommended), or use the `useBaseUrl` utility function: both prepend the `baseUrl` to paths for you.

JSX example#

```
MyComponent.js
```

```
Copy
```

```
MyComponent.js
```

```
Copy
```

```
MyComponent.js
```

```
Copy
```

You can also import SVG files: they will be transformed into React components.

```
MyComponent.js
```

```
Copy
```

Markdown example#

Markdown links and images referencing assets of the static folder will be converted to `require("@site/static/assetName.png")`, and the site `baseUrl` will be automatically prepended for you.

```
my-doc.md
```

```
Copy
```

Thanks to MDX, you can also use `useBaseUrl` utility function in Markdown files! You'd have to use html tags like `` instead of the Markdown image syntax though. The syntax is exactly the same as in JSX.

```
my-doc.mdx
```

```
Copy
```

Caveats#

Keep in mind that:

- By default, none of the files in `static` folder will be post-processed, hashed or minified.
- Missing files referenced via hardcoded absolute paths will not be detected at compilation time, and will result in a 404 error.
- By default, GitHub Pages runs published files through [Jekyll](#). Since Jekyll will discard any files that begin with `_`, it is recommended that you disable Jekyll by adding an empty file named `.nojekyll` file to your `static` directory if you are using GitHub pages for hosting.

Version: 2.0.0-beta.0

Search

Docusaurus' own `@docusaurus/preset-classic` supports a search integration.

There are two main options, you can use [Algolia DocSearch](#) or bring in your own `SearchBar` component.

Using Algolia DocSearch#

Algolia 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 (i.e., not behind a firewall). The service is free.

If your website is [not eligible](#) for the free, hosted version of DocSearch, or if your website sits behind a firewall, then you can [run your own](#) DocSearch crawler. For best results, you may want to use a config file based on the [Docusaurus 2 config](#)

Connecting Algolia#

To connect your docs with Algolia, add an `algolia` field in your `themeConfig`. [Apply for DocSearch](#) to get your Algolia index and API key.

```
docusaurus.config.js
```

Copy

 info

The `searchParameters` option used to be named `algoliaOptions` in Docusaurus v1.

Contextual search#

Contextual search is mostly useful for versioned Docusaurus sites.

Let's consider you have 2 docs versions, v1 and v2. When you are browsing v2 docs, it would be odd to return search results for the v1 documentation. Sometimes v1 and v2 docs are quite similar, and you would end up with duplicate search results for the same query (one result per version).

To solve this problem, the contextual search feature understands that you are browsing a specific docs version, and will create the search query filters dynamically.

- browsing `/docs/v1/myDoc`, search results will only include v1 docs (+ other unversioned pages)
- browsing `/docs/v2/myDoc`, search results will only include v2 docs (+ other unversioned pages)

docusaurus.config.js

Copy

⚠caution

When using `contextualSearch: true`, the contextual facet filters will be merged with the ones provided with `algolia.searchParameters.facetFilters`.

Custom Application ID#

When [running your own](#) DocSearch crawler, it is [required to set the `appId` configuration key](#) to your own Application ID. If left unset, the `appId` will fallback to the one used with the free, hosted version of Algolia DocSearch.

docusaurus.config.js

Copy

Styling your Algolia search#

By default, DocSearch comes with a fine-tuned theme that was designed for accessibility, making sure that colors and contrasts respect standards.

Still, you can reuse the [Infima CSS variables](#) from Docusaurus to style DocSearch by editing the `/src/css/custom.css` file.

/src/css/custom.css

Copy

Customizing the Algolia search behavior#

Algolia DocSearch supports a [list of options](#) that you can pass to the `algolia` field in the `docusaurus.config.js` file.

docusaurus.config.js

Copy

Editing the Algolia search component#

If you prefer to edit the Algolia search React component, swizzle the `SearchBar` component in `@docusaurus/theme-search-algolia`:

- npm
- Yarn

Copy

Using your own search#

To use your own search, swizzle the `SearchBar` component in `@docusaurus/theme-classic`

- npm
- Yarn

Copy

This will create a `src/themes/SearchBar` file in your project folder. Restart your dev server and edit the component, you will see that Docusaurus uses your own `SearchBar` component now.

Notes: You can alternatively [swizzle from Algolia SearchBar](#) and create your own search component from there.

Version: 2.0.0-beta.0

Browser support

Docusaurus allows sites to define the list of supported browsers through a [browserslist configuration](#).

Purpose#

Websites need to balance between backward compatibility and bundle size. As old browsers do not support modern APIs or syntax, more code is needed to implement the same functionality, penalizing all other users with increased site load time. As a tradeoff, the Docusaurus bundler only supports browser versions defined in the browser list.

The browser list by default is provided through the `package.json` file as a root `browserslist` field.

⚠caution

On old browsers, the compiled output will use unsupported (too recent) JS syntax, causing React to fail to initialize and ending up with a static website with only HTML/CSS and no JS.

Default values#

Websites initialized with the default classic template has the following in `package.json`:

`package.json`

Copy

Explained in natural language, the browsers supported in production are those:

- With more than 0.5% of market share; *and*
- Has official support or updates in the past 24 months (the opposite of "dead"); *and*
- Is not Opera Mini.

And browsers used in development are:

- The latest version of Chrome *or* Firefox *or* Safari.

You can "evaluate" any config with the `browserslist` cli to obtain the actual list:

Copy

The output are all browsers supported in production. Below is the output in May, 2021:

Copy

Read more#

[Read more](#)

You may wish to visit the [browserslist documentation](#) for more specifications, especially the accepted [query values](#) and [best practices](#).

Version: 2.0.0-beta.0

Deployment

To build the static files of your website for production, run:

- npm
- Yarn

Copy

Once it finishes, the static files will be generated within the `build/` directory.

You can deploy your site to static site hosting services such as [Vercel](#), [GitHub Pages](#), [Netlify](#), [Render](#), and [Surge](#). Docusaurus sites are statically rendered so they work without JavaScript too!

Testing Build Local#

It is important to test build before deploying to a production. Docusaurus includes a [docusaurus_serve](#) command to test build locally.

- npm
- Yarn

Copy

Self Hosting#

 warning

It is not the most performant solution

Docusaurus can be self hosted using [docusaurus_serve](#). Change port using `--port` and `--host` to change host.

- npm
- Yarn

Copy


Deploying to GitHub Pages#

Docusaurus provides an easy way to publish to [GitHub Pages](#) Which is hosting that comes for free with every GitHub repository.

`docusaurus.config.js` settings#

First, modify your `docusaurus.config.js` and add the required params:

| Name | Description |
|-------------------------------|--|
| <code>organizationName</code> | The GitHub user or organization that owns the repository. If you are the owner, it is your GitHub username. In the case of Docusaurus, it is <i>"facebook"</i> which is the GitHub organization that owns Docusaurus. |
| <code>projectName</code> | The name of the GitHub repository. For example, the repository name for Docusaurus is <i>"docusaurus"</i> , so the project name is <i>"docusaurus"</i> . |
| <code>url</code> | URL for your GitHub Page's user/organization page. This is commonly https://_username_.github.io |
| <code>baseUrl</code> | Base URL for your project. For projects hosted on GitHub pages, it follows the format <i>"/projectName/"</i> . For https://github.com/facebook/docusaurus , <code>baseUrl</code> is <i>"/docusaurus/"</i> . |

 info

In case you want to use your custom domain for GitHub Pages, create a `CNAME` file in the `static` directory. Anything within the `static` directory will be copied to the root of the `build` directory for deployment.

You may refer to GitHub Pages' documentation [User, Organization, and Project Pages](#) for more details.

Example:

```
docusaurus.config.js
```

Copy

 warning

By default, GitHub Pages runs published files through [Jekyll](#). Since Jekyll will discard any files that begin with `_`, it is recommended that you disable Jekyll by adding an empty file named `.nojekyll` file to your `static` directory.

Environment settings#

Specify the Git user as an environment variable.

| Name | Description |
|-----------------------|---|
| <code>GIT_USER</code> | The username for a GitHub account that has commit access to this repo. For your own repositories, this will usually be your GitHub username. The specified <code>GIT_USER</code> must have push access to the repository specified in the combination of <code>organizationName</code> and <code>projectName</code> . |

Optional parameters, also set as environment variables:

| Name | Description |
|--------------------------------|---|
| <code>USE_SSH</code> | Set to <code>true</code> to use SSH instead of the default HTTPS for the connection to the GitHub repo. |
| <code>DEPLOYMENT_BRANCH</code> | The branch that the website will be deployed to, defaults to <code>gh-pages</code> for normal repos and <code>master</code> for repository names ending in <code>github.io</code> . |
| <code>CURRENT_BRANCH</code> | The branch that contains the latest docs changes that will be deployed. Usually, the branch will be <code>master</code> , but it could be any branch (default or otherwise) except for <code>gh-pages</code> . If nothing is set for this variable, then the current branch will be used. |
| | Password (or token) of the <code>git</code> user (specified by <code>git</code> user). For example, to facilitate non- |

| Name | Description |
|-----------------------|---|
| <code>GIT_PASS</code> | interactive deployment (e.g. continuous deployment) |

GitHub enterprise installations should work in the same manner as `github.com`; you only need to set the organization's GitHub Enterprise host as an environment variable:

| Name | Description |
|--------------------------|---|
| <code>GITHUB_HOST</code> | The domain name of your GitHub enterprise site. |
| <code>GITHUB_PORT</code> | The port of your GitHub enterprise site. |

Deploy#

Finally, to deploy your site to GitHub Pages, run:

- Bash
- Windows
- PowerShell

Copy

Triggering deployment with GitHub Actions#

[GitHub Actions](#) allow you to automate, customize, and execute your software development workflows right in your repository.

This workflow assumes your documentation resided in `documentation` branch of your repository and your [publishing source](#) is configured for `gh-pages` branch.

1. Generate a new [SSH key](#).
2. By default, your public key should have been created in `~/.ssh/id_rsa.pub` or use the name you've provided in the previous step to add your key to [GitHub deploy keys](#)
3. Copy key to clipboard with `xclip -sel clip < ~/.ssh/id_rsa.pub` and paste it as a [deploy key](#) in your repository. Copy file content if the command line doesn't work for you. Check the box for `allow write access` before saving your deployment key.
4. You'll need your private key as a [GitHub secret](#) to allow Docusaurus to run the deployment for you.
5. Copy your private key with `xclip -sel clip < ~/.ssh/id_rsa` and paste a GitHub secret with name `GH_PAGES_DEPLOY`. Copy file content if the command line doesn't work for you. Save your secret.
6. Create your [documentation workflow file](#) in `.github/workflows/`. In this example it's `documentation.yml`.

⚠warning

Please make sure that you replace `actions@github.com` with your GitHub email and `gh-actions` with your name.

documentation.yml

Copy

1. Now when a new pull request arrives towards your repository in branch `documentation` it will automatically ensure that Docusaurus build is successful.
2. When pull request is merged to `documentation` branch or someone pushes to `documentation` branch directly it will be built and deployed to `gh-pages` branch.
3. After this step, your updated documentation will be available on the GitHub pages.

Triggering deployment with Travis CI#

Continuous integration (CI) services are typically used to perform routine tasks whenever new commits are checked

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 deploying changes to your website. All you need to do to automate the deployment of your website is to invoke the `yarn deploy` script whenever your website is updated. The following section covers how to do just that using [Travis CI](#), a popular continuous integration service provider.

1. Go to <https://github.com/settings/tokens> and generate a new [personal access token](#). When creating the token, grant it the `repo` scope so that it has the permissions it needs.
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 the following:

```
.travis.yml
```

Copy

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 `yarn deploy` 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 the `repo` 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
```

Copy

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.

Copy

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

Deploying to Netlify#

To deploy your Docusaurus 2 sites to [Netlify](#), first make sure the following options are properly configured:

```
docusaurus.config.js
```

Then, [create your site with Netlify](#).

While you set up the site, specify the build commands and directories as follows:

- build command: `npm run build`
- build directory: `build`

If you did not configure these build options, you may still go to "Site settings" -> "Build and deploy" after your site is created.

Once properly configured with the above options, your site should deploy and automatically redeploy upon merging to your deploy branch, which defaults to `master`.

ⓘimportant

Make sure to disable Netlify setting `Pretty URLs` to prevent lowercased URLs, unnecessary redirects and 404 errors.

Deploying to 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](#).

Deploying to Render#

[Render](#) offers [free static site hosting](#) with fully managed SSL, custom domains, a global CDN and continuous auto-deploy from your Git repo. Get started in just a few minutes by following [Render's guide to deploying Docusaurus](#).

Deploying to Qovery#

[Qovery](#) is a fully-managed cloud platform that runs on your AWS, GCP, Azure and Digital Ocean account where you can host static sites, backend APIs, databases, cron jobs, and all your other apps in one place.

1. Create a Qovery account.

Visit the [Qovery dashboard](#) to create an account if you don't already have one.

2. Create a project

Click on "Create a new project" and give a name to your project.

Click on "Next".

3. Add an application

Click on "Create an application" then choose "I have an application" and select your GitHub or GitLab repository where your app is located.

Click on "Next".

Skip adding services

4. Deploy

Click on "Deploy".

You can see the status in real time by clicking on deployment logs.

Deploying to Hostman#

[Hostman](#) allows you to host static websites for free. Hostman automates everything, you just need to connect your repository and follow easy steps:

1. Create a service

To deploy a Docusaurus static website, click Create in the top-left corner of your [Dashboard](#) and choose Front-end app or static website.

2. Select the project to deploy

If you are logged in to Hostman with your GitHub, GitLab or Bitbucket account, at this point you will see the repository with your projects, including the private ones.

Choose the project you want to deploy. It must contain the directory with the project's files (usually it is website or my-website).

To access a different repository, click Connect another repository.

If you didn't use your Git account credentials to log in, you'll be able to access the necessary account now, and then select the project.

3. Configure the build settings Next, the Website customization window will appear.

Choose the Static website option from the list of frameworks.

The Directory with app points at the directory that will contain the project's files after the build. You can leave it empty if during Step 2 you selected the repository with the contents of the website (or my_website) directory.

The standard build command for Docusaurus will be:

```
Copy
```

You can modify the build command if needed. You can enter multiple commands separated by &&.

4. Deploy Click Deploy to start the build process.

Once it starts, you will enter the deployment log. If there are any issues with the code, you will get warning or error messages in the log, specifying the cause of the problem.

Usually the log contains all the debugging data you'll need, but we are also here to help you solve the issues, so do not hesitate to contact us via chat.

When the deployment is complete, you will receive an e-mail notification and also see a log entry.

All done!

Your project is up and ready.

Deploying to Surge#

Surge is a [static web hosting platform](#), it is used to deploy your Docusaurus project from the command line in a minute. Deploying your project to Surge is easy and it is also free (including a custom domain and SSL).

Deploy your app in a matter of seconds using surge with the following steps:

1. First, install Surge using npm by running the following command:

Copy

2. To build the static files of your site for production in the root directory of your project, run:

Copy

3. Then, run this command inside the root directory of your project:

Copy

First-time users of Surge would be prompted to create an account from the command line(happens only once).

Confirm that the site you want to publish is in the `build` directory, a randomly generated subdomain `*.surge.sh` subdomain is always given (which can be edited).

Using your domain#

If you have a domain name you can deploy your site using surge to your domain using the command:

Copy

Your site is now deployed for free at `subdomain.surge.sh` or `yourdomain.com` depending on the method you chose.

Setting up CNAME file#

Store your domain in a CNAME file for future deployments with the following command:

Copy

You can deploy any other changes in the future with the command `surge`.

Deploying to QuantCDN#

1. Install [Quant CLI](#)
2. Create a QuantCDN account by [signing up](#)
3. Initialize your project with `quant init` and fill in your credentials:

Copy

4. Deploy your site

Copy

See [docs](#) and [blog](#) for more examples and use cases for deploying to QuantCDN.

Version: 2.0.0-beta.0

i18n - Introduction

It is **easy to translate a Docusaurus website** with its internationalization ([i18n](#)) support.

Goals#

It is important to understand the **design decisions** behind the Docusaurus i18n support.

For more context, you can read the initial [RFC](#) and [PR](#).

i18n goals#

The goals of the Docusaurus i18n system are:

- **Simple:** just put the translated files in the correct filesystem location
- **Flexible translation workflows:** use Git (monorepo, forks, or submodules), SaaS software, FTP
- **Flexible deployment options:** single, multiple domains, or hybrid
- **Modular:** allow plugin authors to provide i18n support
- **Low-overhead runtime:** documentation is mostly static and does not require a heavy JS library or polyfills
- **Scalable build-times:** allow building and deploying localized sites independently
- **Localize assets:** an image of your site might contain text that should be translated
- **No coupling:** not forced to use any SaaS, yet integrations are possible
- **Easy to use with [Crowdin](#):** multiple Docusaurus v1 sites use Crowdin, and should be able to migrate to v2
- **Good SEO defaults:** we set useful SEO headers like [hreflang](#) for you
- **RTL support:** locales reading right-to-left (Arabic, Hebrew, etc.) are supported and easy to implement
- **Default translations:** classic theme labels are translated for you in [many languages](#)

i18n non-goals#

We don't provide support for:

- **Automatic locale detection:** opinionated, and best done on the [server](#)
- **Translation SaaS software:** you are responsible to understand the external tools of your choice
- **Translation of slugs:** technically complicated, little SEO value

Translation workflow#

Overview#

Overview of the workflow to create a translated Docusaurus website:

1. **Configure:** declare the default locale and alternative locales in `docusaurus.config.js`

2. **Translate:** put the translation files at the correct filesystem location
3. **Deploy:** build and deploy your site using a single or multi-domain strategy

Translation files#

You will have to work with 2 kind of translation files.

Markdown files#

This is the main content of your Docusaurus website.

Markdown and MDX documents are translated as a whole, to fully preserve the translation context, instead of splitting each sentence as a separate string.

JSON files#

JSON is used to translate:

- your React code: using the `<Translate>` component
- your theme: the navbar, footer
- your plugins: the docs sidebar category labels

The JSON format used is called **Chrome i18n**:

Copy

The choice was made for 2 reasons:

- **Description attribute:** to help translators with additional context
- **Widely supported:** [Chrome extensions](#) [Crowdin](#) [Transifex](#) [Phrase](#) [Applanga](#)

Translation files location#

The translation files should be created at the correct filesystem location.

Each locale and plugin has its own `i18n` subfolder:

Copy

 note

For multi-instance plugins, the path is `website/i18n/<locale>/<pluginName>-<pluginId>/...`

Translating a very simple Docusaurus site in French would lead to the following tree:

Copy

The JSON files are initialized with the [docusaurus write-translations](#) CLI command.

The `code.json` file is extracted from React components using the `<Translate>` API.

 info

Notice that the `docusaurus-plugin-content-docs` plugin has a `current` subfolder and a `current.json` file, useful for the docs versioning feature.

Each content plugin or theme is different, and **define its own translation files location**:

- [Docs i18n](#)
- [Blog i18n](#)
- [Pages i18n](#)
- [Themes i18n](#)

Version: 2.0.0-beta.0

i18n - Tutorial

This tutorial will walk you through the basis of the **Docusaurus i18n system**.

We will add **French** translations to a **newly initialized English Docusaurus website**.

Initialize a new site with `npx @docusaurus/init@latest init website classic` (like [this one](#)).

Configure your site#

Modify `docusaurus.config.js` to add the i18n support for the French language.

Site configuration#

Use the [site i18n configuration](#) to declare the i18n locales:

```
docusaurus.config.js
```

Copy

Theme configuration#

Add a **navbar item** of type `localeDropdown` so that users can select the locale they want:

```
docusaurus.config.js
```

Copy

Start your site#

Start your localized site in dev mode, using the locale of your choice:

- `npm`
- `Yarn`

Copy

Your site is accessible at `http://localhost:3000/fr/`.

We haven't provided any translation, and the site is **mostly untranslated**.

Tip

Docusaurus provides **default translations** for generic theme labels, such as "Next" and "Previous" for the

pagination.

Please help us complete those [default translations](#).

⚠caution

Each locale is a **distinct standalone single-page-application**: it is not possible to start the Docusaurus sites in all locales at the same time.

Translate your site#

The French translations will be added in `website/i18n/fr`.

Docusaurus is modular, and each content plugin has its own subfolder.

①note

After copying files around, restart your site with `npm run start -- --locale fr`.

Hot-reload will work better when editing existing files.

Use the translation APIs#

Open the homepage, and use the [translation APIs](#).

```
src/pages/index.js
```

Copy

⚠caution

Docusaurus provides a **very small and lightweight translation runtime** on purpose, and only supports basic [placeholders interpolation](#), using a subset of the [ICU Message Format](#).

Most documentation websites are generally **static** and don't need advanced i18n features (**plurals, genders, etc.**). Use a library like [react-intl](#) for more advanced use-cases.

Translate JSON files:#

JSON translation files are used for everything that is not contained in a Markdown document:

- React/JSX code
- Layout navbar and footer labels
- Docs sidebar category labels
- ...

Run the [write-translations](#) command:

- npm
- Yarn

Copy

It will extract and initialize the JSON translation files that you need to translate.

The homepage translations are statically extracted from React source code.

```
i18n/fr/code.json
```

Copy

Plugins and themes will also write their own **JSON translation files**, such as:

```
i18n/fr/docusaurus-theme-classic/navbar.json
```

Copy

Translate the `message` attribute in the JSON files of `i18n/fr`, and your site layout and homepage should now be translated.

Translate Markdown files <#>

Official Docusaurus content plugins extensively use Markdown/MDX files, and allow you to translate them.

Translate the docs <#>

Copy your docs Markdown files to `i18n/fr/docusaurus-plugin-content-docs/current`, and translate them:

Copy

 info

`current` is needed for the docs versioning feature: each docs version has its own subfolder.

Translate the blog <#>

Copy your blog Markdown files to `i18n/fr/docusaurus-plugin-content-blog`, and translate them:

Copy

Translate the pages <#>

Copy your pages Markdown files to `i18n/fr/docusaurus-plugin-content-pages`, and translate them:

Copy

 caution

We only copy `.md` and `.mdx` files, as pages React components are translated through JSON translation files already.

Use explicit heading ids <#>

By default, a Markdown heading `### Hello World` will have a generated id `hello-world`.

Other documents can target it with `[link] (#hello-world)`.

The translated heading becomes `### Bonjour le Monde`, with id `bonjour-le-monde`.

Generated ids are not always a good fit for localized sites, as it requires you to localize all the anchor links:

Copy

 tip

For localized sites, it is recommended to use [explicit heading ids](#)

Deploy your site#

You can choose to deploy your site under a **single domain**, or use **multiple (sub)domains**.

Single-domain deployment#

Run the following command:

- npm
- Yarn

Copy

Docusaurus will build **one single-page application per locale**:

- `website/build`: for the default, English language
- `website/build/fr`: for the French language

You can now [deploy](#) the `build` folder to the static hosting solution of your choice.

 note

The Docusaurus v2 website use this strategy:

- <https://docusaurus.io>
- <https://docusaurus.io/fr>

 tip

Static hosting providers generally redirect `/unknown/urls` to `/404.html` by convention, always showing an **English 404 page**.

Localize your 404 pages by configuring your host to redirect `/fr/*` to `/fr/404.html`.

This is not always possible, and depends on your host: GitHub Pages can't do this, [Netlify](#) can.

Multi-domain deployment#

You can also build your site for a single locale:

- npm
- Yarn

Copy

Docusaurus will not add the `en` / URL prefix

On your [static hosting provider](#):

- create one deployment per locale
- configure the appropriate build command, using the `--locale` option
- configure the (sub)domain of your choice for each deployment

⚠caution

This strategy is **not possible** with Github Pages, as it is only possible to **have a single deployment**.

Hybrid#

It is possible to have some locales using sub-paths, and others using subdomains.

It is also possible to deploy each locale as a separate subdomain, assemble the subdomains in a single unified domain at the CDN level:

- Deploy your site as `fr.docusaurus.io`
- Configure a CDN to serve it from `docusaurus.io/fr`

Version: 2.0.0-beta.0

i18n - Using git

A possible translation strategy is to **version control the translation files** to Git (or any other [VCS](#)).

Tradeoffs#

This strategy has advantages:

- **Easy to get started**: just add the `i18n` folder to Git
- **Easy for developers**: Git, GitHub and pull requests are mainstream developer tools
- **Free** (or without any additional cost, assuming you already use Git)
- **Low friction**: does not require signing-up to an external tool
- **Rewarding**: contributors are happy to have a nice contribution history

Using Git also present some shortcomings:

- **Hard for non-developers**: they do not master Git and pull-requests
- **Hard for professional translations**: they are used to SaaS translation softwares and advanced features
- **Hard to maintain**: you have to keep the translated files **in sync** with the untranslated files

📌note

Some **large-scale technical projects** (React, Vue.js, MDN, TypeScript, Nuxt.js, etc.) use Git for translations.

Refer to the [Docusaurus i18n RFC](#) for our notes and links studying these systems.

Git tutorial#

This is a walk-through of using Git to translate a newly initialized English Docusaurus website into French, and assume you already followed the [i18n tutorial](#)

Prepare the Docusaurus site#

Initialize a new Docusaurus site:

Copy

Add the site configuration for the French language:

```
docusaurus.config.js
```

Copy

Translate the homepage:

```
src/pages/index.js
```

Copy

Initialize the i18n folder#

Use the [write-translations](#) CLI command to initialize the JSON translation files for the French locale:

- npm
- Yarn

Copy

Tip

Use the `--messagePrefix '(fr) '` option to make the untranslated strings stand out.

`Hello` will appear as `(fr) Hello` and makes it clear a translation is missing.

Copy your untranslated Markdown files to the French folder:

Copy

Add all these files to Git.

Translate the files#

Translate the Markdown and JSON files in `i18n/fr` and commit the translation.

You should now be able to start your site in French and see the translations:

- npm
- Yarn

Copy

You can also build the site locally or on your CI:

- npm
- Yarn

Copy

Repeat#

Follow the same process for each locale you need to support.

Maintain the translations#

Keeping translated files **consistent** with the originals **can be challenging**, in particular for Markdown documents.

Markdown translations#

When an untranslated Markdown document is edited, it is **your responsibility to maintain the respective translated files**, and we unfortunately don't have a good way to help you do so.

To keep your translated sites consistent, when the `website/docs/doc1.md` doc is edited, you need **backport these edits** to `i18n/fr/docusaurus-plugin-content-docs/current/doc1.md`.

JSON translations#

To help you maintain the JSON translation files, it is possible to run again the [write-translations](#) CLI command:

- npm
- Yarn

Copy

New translation will be appended, and existing ones will not be overridden.

Tip

Reset your translations with the `--override` option.

Localize edit urls#

When the user is browsing a page at `/fr/doc1`, the edit button will link by default to the unlocalized doc at `website/docs/doc1.md`.

Your translations are on Git, and you can use the `editLocalizedFiles: true` option of the docs and blog plugins.

The edit button will link to the localized doc at `i18n/fr/docusaurus-plugin-content-docs/current/doc1.md`.

Version: 2.0.0-beta.0

i18n - Using Crowdin

The i18n system of Docusaurus is **decoupled from any translation software**.

You can integrate Docusaurus with the **tools and SaaS of your choice**, as long as you put the **translation files at the correct location**.

We document the usage of [Crowdin](#) as **one possible integration example**.

⚠ caution

This is **not an endorsement of Crowdin** as the unique choice to translate a Docusaurus site, but it is successfully used by Facebook to translate documentation projects such as [Jest Docusaurus](#) and [ReasonML](#).

Refer to the [Crowdin documentation](#) and [Crowdin support](#) for help.

💡 tip

Use this [community-driven GitHub issue](#) to discuss anything related to Docusaurus + Crowdin.

Crowdin overview#

Crowdin is a translation SaaS, offering a [free plan for open-source projects](#).

We recommend the following translation workflow:

- **Upload sources** to Crowdin (untranslated files)
- Use Crowdin to **translate the content**
- **Download translations** from Crowdin (localized translation files)

Crowdin provides a [CLI](#) to **upload sources** and **download translations**, allowing you to automate the translation process.

The [crowdin.yml configuration file](#) is convenient for Docusaurus, and permits to **download the localized translation files at the expected location** (in `i18n/<locale>/..`).

Read the [official documentation](#) to know more about advanced features and different translation workflows.

Crowdin tutorial#

This is a walk-through of using Crowdin to translate a newly initialized English Docusaurus website into French, and assume you already followed the [i18n tutorial](#).

The end result can be seen at [docusaurus-crowdin-example.netlify.app](#) ([repository](#)).

Prepare the Docusaurus site#

Initialize a new Docusaurus site:

Copy

Add the site configuration for the French language:

```
docusaurus.config.js
```

Copy

Translate the homepage:

```
src/pages/index.js
```

Copy

Create a Crowdin project#

Sign up on [Crowdin](#), and create a project.

Use English as source language, and French as target language.

Your project is created, but it is empty for now. We will upload the files to translate in the next steps.

Create the Crowdin configuration#

This configuration ([doc](#)) provides a mapping for the Crowdin CLI to understand:

- Where to find the source files to upload (JSON and Markdown)
- Where to download the files after translation (in `i18n/<locale>`)

Create `crowdin.yml` in website:

```
crowdin.yml
```

Copy

Crowdin has its own syntax for declaring source/translation paths:

- `**/*`: everything in a subfolder
- `%two_letters_code%`: the 2-letters variant of Crowdin target languages (`fr` in our case)
- `**/%original_file_name%`: the translations will preserve the original folder/file hierarchy

 info

The Crowdin CLI warnings are not always easy to understand.

We advise to:

- change one thing at a time
- re-upload sources after any configuration change
- use paths starting with `/` (`./` does not work)
- avoid fancy globbing patterns like `/docs/**/*` (`md|mdx`) (does not work)

Access token#

The `api_token_env` attribute defines the **env variable name** read by the Crowdin CLI.

You can obtain a Personal Access Token on [your personal profile page](#)

 tip

You can keep the default value `CROWDIN_PERSONAL_TOKEN`, and set this environment variable and on your computer and on the CI server to the generated access token.

 caution

A Personal Access Tokens grant **read-write access to all your Crowdin projects**.

You should not commit it, and it may be a good idea to create a dedicated Crowdin profile for your company.

you should not commit it, and it may be a good idea to create a dedicated Crowdin profile for your company instead of using a personal account.

Other configuration fields#

- `project_id`: can be hardcoded, and is found on https://crowdin.com/project/<MY_PROJECT_NAME>/settings#api
- `preserve_hierarchy`: preserve the folder's hierarchy of your docs on Crowdin UI instead of flattening everything

Install the Crowdin CLI#

This tutorial use the CLI in version 3.5.2, but we expect 3.x releases to keep working.

Install the Crowdin CLI as a NPM package to your Docusaurus site:

- npm
- Yarn

Copy

Add a `crowdin` script:

```
package.json
```

Copy

Test that you can run the Crowdin CLI:

- npm
- Yarn

Copy

Set the `CROWDIN_PERSONAL_TOKEN` env variable on your computer, to allow the CLI to authenticate with the Crowdin API.

Tip

Temporarily, you can hardcode your personal token in `crowdin.yml` with `api_token: 'MY-TOKEN'`.

Upload the sources#

Generate the JSON translation files for the default language in `website/i18n/en`:

- npm
- Yarn

Copy

Upload all the JSON and Markdown translation files:

- npm
- Yarn

Copy

Copy

Your source files are now visible on the Crowdin interface:

https://crowdin.com/project/<MY_PROJECT_NAME>/settings#files

Translate the sources#

On https://crowdin.com/project/<MY_PROJECT_NAME>, click on the French target language.

Translate some Markdown files.



Use `Hide String` to make sure translators **don't translate things that should not be**:

- Frontmatter: `id, slug, tags ...`
- Admonitions: `:::, :::note, :::tip ...`

Translate some JSON files.



The `description` attribute of JSON translation files is visible on Crowdin to help translate the strings.



[Pre-translate](#) your site, and **fix pre-translation mistakes manually** (enable the Global Translation Memory in settings first).

Use the `Hide String` feature first, as Crowdin is pre-translating things too optimistically.

Download the translations#

Use the Crowdin CLI to download the translated JSON and Markdown files.

- npm
- Yarn

Copy

The translated content should be downloaded in `i18n/fr`.

Start your site on the French locale:

- npm
- Yarn

Copy

Make sure that your website is now translated in French at <http://localhost:3000/fr/>.

Automate with CI#

We will configure the CI to **download the Crowdin translations at build time**, and keep them outside of Git.

Add `website/i18n` to `.gitignore`.

Set the `CROWDIN_PERSONAL_TOKEN` env variable on your CI.

Create a npm script to `sync` Crowdin (extract sources, upload sources, download translations):

```
package.json
```

Copy

Call the `npm run crowdin:sync` script in your CI, just before building the Docusaurus site.



tip

Keep your deploy-previews fast: don't download translations, and use `npm run build -- --locale en` for feature branches.



caution

Crowdin does not support well multiple concurrent uploads/downloads: it is preferable to only include translations to your production deployment, and keep deploy previews untranslated.

Advanced Crowdin topics#

MDX#



caution

Pay special attention to the JSX fragments in MDX documents!

Crowdin **does not support officially MDX**, but they added **support for the `.mdx` extension**, and interpret such files as Markdown (instead of plain text).

MDX problems#

Crowdin thinks the JSX syntax is embedded HTML, and can mess-up with the JSX markup when you download the translations, leading to a site that fails to build due to invalid JSX.

Simple JSX fragments using simple string props like `<Username name="Sebastien"/>` will work fine.

More complex JSX fragments using object/array props like `<User person={{name: "Sebastien"}}/>` are more likely to fail due to a syntax that does not look like HTML.

MDX solutions#

We recommend moving the complex embedded JSX code as separate standalone components.

We also added a `mdx-code-block` escape hatch syntax:

Copy

This will:

- be interpreted by Crowdin as code blocks (and not mess-up with the markup on download)
- be interpreted by Docusaurus as regular JSX (as if it was not wrapped by any code block)
- unfortunately opt-out of MDX tooling (IDE syntax highlighting, Prettier...)

Docs versioning#

Configure translation files for the `website/versioned_docs` folder.

When creating a new version, the source strings will generally be quite similar to the current version (`website/docs`), and you don't want to translate the new version docs again and again.

Crowdin provides a `Duplicate Strings` setting.

We recommend using `Hide`, but the ideal setting depends on how much your versions are different.

⚠caution

Not using `Hide` leads to a much larger amount of `source strings` in quotas, and will affect the pricing.

Multi-instance plugins#

You need to configure translation files for each plugin instance.

If you have a docs plugin instance with `id=ios`, you will need to configure those source files as well

- `website/ios`
- `website/ios_versioned_docs` (if versioned)

Maintaining your site#

Sometimes, you will **remove or rename a source file** on Git, and Crowdin will display CLI warnings:

When your sources are refactored, you should use the Crowdin UI to **update your Crowdin files manually**:

VCS (Git) integrations#

Crowdin has multiple VCS integrations for [GitHub](#), GitLab, Bitbucket.

⚠warning

We recommend avoiding them.

It could have been helpful to be able to edit the translations in both Git and Crowdin, and have a **bi-directional sync** between the 2 systems.

In practice, it **didn't work very reliably** for a few reasons:

- The Crowdin -> Git sync works fine (with a pull request)
- The Git -> Crowdin sync is manual (you have to press a button)
- The heuristics used by Crowdin to match existing Markdown translations to existing Markdown sources are not 100% reliable, and you have to verify the result on Crowdin UI after any sync from Git
- 2 users concurrently editing on Git and Crowdin can lead to a translation loss
- It requires the `crowdin.yml` file to be at the root of the repository

In-Context localization#

Crowdin has an [In-Context localization](#) feature.

⚠caution

Unfortunately, it does not work yet for technical reasons, but we have good hope it can be solved.

Crowdin replaces markdown strings with technical ids such as `crowdin:id12345`, but it does so too aggressively, including hidden strings, and mess-up with the frontmatter, admonitions, jsx...

Localize edit urls#

When the user is browsing a page at `/fr/doc1`, the edit button will link by default to the unlocalized doc at `website/docs/doc1.md`.

You may prefer the edit button to link to the Crowdin interface instead, and can use the `editUrl` function to customize the edit urls on a per-locale basis.

```
docusaurus.config.js
```

Copy

📌note

It is currently **not possible to link to a specific file** in Crowdin.

Example configuration#

The **Docusaurus v2 configuration file** is a good example of using versioning and multi-instance:

```
crowdin.yml
```

Copy

```
Version: 2.0.0-beta.0
```

Plugins

Plugins are the building blocks of features in a Docusaurus 2 site. Each plugin handles its own individual feature. Plugins may work and be distributed as part of bundle via [presets](#)

Available plugins#

We maintain a [list of official plugins](#), but the community has also created some [unofficial plugins](#).

We maintain a [list of official plugins](#) but the community has also created some [unofficial plugins](#).

Installing a plugin#

A plugin is usually a npm package, so you install them like other npm packages using npm.

- npm
- Yarn

Copy

Then you add it in your site's `docusaurus.config.js`'s `plugins` option:

```
docusaurus.config.js
```

Copy

Docusaurus can also load plugins from your local directory, you can do something like the following:

```
docusaurus.config.js
```

Copy

Configuring plugins#

For the most basic usage of plugins, you can provide just the plugin name or the absolute path to the plugin.

However, plugins can have options specified by wrapping the name and an options object in an array inside your config. This style is usually called `Babel Style`.

```
docusaurus.config.js
```

Copy

Example:

```
docusaurus.config.js
```

Copy

Multi-instance plugins and plugin ids#

All Docusaurus content plugins can support multiple plugin instances.

The Docs plugin has [additional multi-instance documentation](#)

It is required to assign a unique `id` to each plugin instance.

By default, the plugin `id` is `default`.

```
docusaurus.config.js
```

Copy

 note

At most one plugin instance can be the "default plugin instance", by omitting the `id` attribute, or using `id: 'default'`.

Plugins design#

Docusaurus' implementation of the plugins system provides us with a convenient way to hook into the website's lifecycle to modify what goes on during development/build, which involves (but not limited to) extending the webpack config, modifying the data being loaded and creating new components to be used in a page.

Creating plugins#

A plugin is a module which exports a function that takes two parameters and returns an object when executed.

Module definition#

The exported modules for plugins are called with two parameters: `context` and `options` and returns a JavaScript object with defining the [lifecycle APIs](#).

For example if you have a reference to a local folder such as this in your `docusaurus.config.js`:

```
docusaurus.config.js
```

Copy

Then in the folder `my-plugin` you can create an `index.js` such as this

```
index.js
```

Copy

The `my-plugin` folder could also be a fully fledged package with it's own `package.json` and a `src/index.js` file for example

`context`:#

`context` is plugin-agnostic and the same object will be passed into all plugins used for a Docusaurus website. The `context` object contains the following fields:

Copy

`options`:#

`options` are the [second optional parameter when the plugins are used](#). `options` are plugin-specific and are specified by users when they use them in `docusaurus.config.js`. Alternatively, if `preset` contains the plugin, the `preset` will then be in charge of passing the correct options into the plugin. It is up to individual plugin to define what options it takes.

Return value#

The returned object value should implement the [lifecycle APIs](#).

Version: 2.0.0-beta.0

Themes

Like plugins, themes are designed to add functionality to your Docusaurus site. As a good rule of thumb, themes are mostly focused on client-side, where plugins are more focused on server-side functionalities. Themes are also

designed to be replace-able with other themes.

Available themes#

We maintain a [list of official themes](#).

Using themes#

To use themes, specify the themes in your `docusaurus.config.js`. You may use multiple themes:

```
docusaurus.config.js
```

Copy

Theme components#

Most of the time, theme is used to provide a set of React components, e.g. `Navbar`, `Layout`, `Footer`.

Users can use these components in their code by importing them using the `@theme` webpack alias:

Copy

The alias `@theme` can refer to a few directories, in the following priority:

1. A user's `website/src/theme` directory, which is a special directory that has the higher precedence.
2. A Docusaurus theme packages's `theme` directory.
3. Fallback components provided by Docusaurus core (usually not needed).

Given the following structure

Copy

`website/src/theme/Navbar.js` takes precedence whenever `@theme/Navbar` is imported. This behavior is called component swizzling. In iOS, method swizzling is the process of changing the implementation of an existing selector (method). In the context of a website, component swizzling means providing an alternative component that takes precedence over the component provided by the theme.

Themes are for providing UI components to present the content. Most content plugins need to be paired with a theme in order to be actually useful. The UI is a separate layer from the data schema, so it makes it easy to swap out the themes for other designs (i.e., Bootstrap).

For example, a Docusaurus blog consists of a blog plugin and a blog theme.

```
docusaurus.config.js
```

Copy

And if you want to use Bootstrap styling, you can swap out the theme with `theme-blog-bootstrap` (fictitious non-existing theme):

```
docusaurus.config.js
```

Copy

Wrapping your site with `react-helmet` #

Wrapping your site with `<Root>` <#>

A `<Root>` theme component is rendered at the very top of your Docusaurus site.

It allows you to wrap your site with additional logic, by creating a file at `src/theme/Root.js`:

```
website/src/theme/Root.js
```

Copy

This component is applied above the router and the theme `<Layout>`, and will **never unmount**.

 tip

Use this component to render React Context providers and global stateful logic.

Swizzling theme components <#>

 caution

We discourage swizzling of components during the Docusaurus 2 beta phase. The theme components APIs are likely to evolve and have breaking changes. If possible, stick with the default appearance for now.

Docusaurus Themes' components are designed to be replaceable. To make it easier for you, we created a command for you to replace theme components called `swizzle`.

To swizzle a component for a theme, run the following command in your doc site:

- `npm`
- `Yarn`

Copy

As an example, to swizzle the `<Footer />` component in `@docusaurus/theme-classic` for your site, run:

- `npm`
- `Yarn`

Copy

This will copy the current `<Footer />` component used by the theme to a `src/theme/Footer` directory under the root of your site, which is where Docusaurus will look for swizzled components. Docusaurus will then use swizzled component in place of the original one from the theme.

Although we highly discourage swizzling of all components, if you wish to do that, run:

- `npm`
- `Yarn`

Copy

Note: You need to restart your webpack dev server in order for Docusaurus to know about the new component.

Wrapping theme components <#>

Sometimes, you just want to wrap an existing theme component with additional logic, and it can be a pain to have to maintain an almost duplicate copy of the original theme component.

In such case, you should swizzle the component you want to wrap, but import the original theme component in your customized version to wrap it.

For site owners#

The `@theme-original` alias allows you to import the original theme component.

Here is an example to display some text just above the footer, with minimal code duplication.

```
src/theme/Footer.js
```

Copy

For plugin authors#

One theme can wrap a component from another theme, by importing the component from the initial theme, using the `@theme-init` import.

Here's an example of using this feature to enhance the default theme `CodeBlock` component with a `react-live` playground feature.

Copy

Check the code of `docusaurus-theme-live-codeblock` for details.

⚠caution

Unless you want publish to npm a "theme enhancer" (like `docusaurus-theme-live-codeblock`), you likely don't need `@theme-init`.

Themes design#

While themes share the exact same lifecycle methods with plugins, their implementations can look very different from those of plugins based on themes' designed objectives.

Themes are designed to complete the build of your Docusaurus site and supply the components used by your site, plugins, and the themes themselves. So a typical theme implementation would look like a `src/index.js` file that hooks it up to the lifecycle methods. Most likely they would not use `loadContent`, which plugins would use. And it is typically accompanied by a `src/theme` directory full of components.

To summarize:

- Themes share the same lifecycle methods with Plugins
- Themes are run after all existing Plugins
- Themes exist to add component aliases by extending the webpack config

Writing customized Docusaurus themes#

A Docusaurus theme normally includes an `index.js` file where you hook up to the lifecycle methods, alongside with a `theme/` directory of components. A typical Docusaurus `theme` folder looks like this:

Copy

There are two lifecycle methods that are essential to theme implementation:

- `getThemePath()`
- `getClientModules()`

These lifecycle method are not essential but recommended:

- `validateThemeConfig({themeConfig, validate})`
- `validateOptions({options, validate})`

Version: 2.0.0-beta.0

Presets

Presets are collections of plugins and themes.

Using presets#

A preset is usually a npm package, so you install them like other npm packages using npm.

- npm
- Yarn

Copy

Then, add it in your site's `docusaurus.config.js`'s `presets` option:

```
docusaurus.config.js
```

Copy

To load presets from your local directory, specify how to resolve them:

```
docusaurus.config.js
```

Copy

Presets -> themes and plugins#

Presets in some way are a shorthand function to add plugins and themes to your docusaurus config. For example, you can specify a preset that includes the following themes and plugins,

Copy

then in your Docusaurus config, you may configure the preset instead:

```
docusaurus.config.js
```

Copy

This is equivalent of doing:

```
docusaurus.config.js
```

Copy

This is especially useful when some plugins and themes are intended to be used together.

Official presets#

@docusaurus/preset-classic#

The classic preset that is usually shipped by default to new docusaurus website. It is a set of plugins and themes.

| Themes | Plugins |
|----------------------------------|--|
| @docusaurus/theme-classic | @docusaurus/plugin-content-docs |
| @docusaurus/theme-search-algolia | @docusaurus/plugin-content-blog @docusaurus/plugin-content-pages @docusaurus/plugin-debug @docusaurus/plugin-google-analytics @docusaurus/plugin-google-gtag @docusaurus/plugin-sitemap |

To specify plugin options individually, you can provide the necessary fields to certain plugins, i.e. `customCss` for `@docusaurus/theme-classic`, pass them in the `preset` field, like this:

```
docusaurus.config.js
```

Copy

In addition to these plugins and themes, `@docusaurus/theme-classic` adds [remark-admonitions](#) as a remark plugin to `@docusaurus/plugin-content-blog` and `@docusaurus/plugin-content-docs`.

The `admonitions` key will be passed as the [options](#) to `remark-admonitions`. Passing `false` will prevent the plugin from being added to MDX.

```
docusaurus.config.js
```

Copy

@docusaurus/preset-bootstrap#

The classic preset that is usually shipped by default to new docusaurus website. It is a set of plugins and themes.

| Themes | Plugins |
|-----------------------------|--|
| @docusaurus/theme-bootstrap | @docusaurus/plugin-content-docs @docusaurus/plugin-content-blog @docusaurus/plugin-content-pages @docusaurus/plugin-debug |

To specify plugin options individually, you can provide the necessary fields to certain plugins, i.e. `docs` for `@docusaurus/theme-bootstrap`, pass them in the `preset` field, like this:

```
docusaurus.config.js
```

Copy

 **caution**

This preset is work in progress

Migration overview

This doc guides you through migrating an existing Docusaurus 1 site to Docusaurus 2.

We try to make this as easy as possible, and provide a migration cli.

Main differences#

Docusaurus 1 is a pure documentation site generator, using React as a server-side template engine, but not loading React on the browser.

Docusaurus 2 is rebuilt it from the ground up, generates a single-page-application, using the full power of React in the browser. It allows for more customizability but preserved the best parts of Docusaurus 1 - easy to get started, versioned docs, and i18n.

Beyond that, Docusaurus 2 is a **performant static site generator** and can be used to create common content-driven websites (e.g. Documentation, Blogs, Product Landing and Marketing Pages, etc) extremely quickly.

While our main focus will still be helping you get your documentations right and well, it is possible to build any kind of website using Docusaurus 2 as it is just a React application. **Docusaurus can now be used to build any website, not just documentation websites.**

Docusaurus 1 structure#

Your Docusaurus 1 site should have the following structure:

Copy

Docusaurus 2 structure#

After the migration, your Docusaurus 2 site could look like:

Copy

!info

This migration does not change the `/docs` folder location, but Docusaurus v2 sites generally have the `/docs` folder inside `/website`

You are free to put the `/docs` folder anywhere you want after having migrated to v2.

Migration process#

There are multiple things to migrate to obtain a fully functional Docusaurus 2 website:

- packages
- cli commands
- site configuration
- markdown files

- sidebars file
- pages, components and CSS
- versioned docs
- i18n support 🗑️

Automated migration process#

The [migration cli](#) will handle many things of the migration for you.

However, some parts can't easily be automated, and you will have to fallback to the manual process.

📌note

We recommend running the migration cli, and complete the missing parts thanks to the manual migration process.

Manual migration process#

Some parts of the migration can't be automated (particularly the pages), and you will have to migrate them manually.

The [manual migration guide](#) will give you all the manual steps.

Support#

For any questions, you can ask in the [#docusaurus-1-to-2-migration Discord channel](#).

Feel free to tag [@slorber](#) in any migration PRs if you would like us to have a look.

We also have volunteers willing to [help you migrate your v1 site](#)

Example migration PRs#

You might want to refer to our migration PRs for [Create React App](#) and [Flux](#) as examples of how a migration for a basic Docusaurus v1 site can be done.

Version: 2.0.0-beta.0

Automated migration

The migration CLI automatically migrates your v1 website to a v2 website.

📌info

Manual work is still required after using the migration CLI, as we can't automate a full migration

The migration CLI migrates:

- Site configurations (from `siteConfig.js` to `docusaurus.config.js`)
- `package.json`
- `sidebars.json`
- `/docs`
- `/blog`

- `/static`
- `versioned_sidebar.json` and `/versioned_docs` if your site uses versioning

To use the migration CLI, follow these steps:

1. Before using the migration CLI, ensure that `/docs`, `/blog`, `/static`, `sidebars.json`, `siteConfig.js`, `package.json` follow the [structure](#) shown at the start of this page.
2. To migrate your v1 website, run the migration CLI with the appropriate filesystem paths:

Copy

3. To view your new website locally, go into your v2 website's directory and start your development server.

Copy

 danger

The migration CLI updates existing files. Be sure to have committed them first!

Option: <#>

You can add option flags to the migration CLI to automatically migrate Markdown content and pages to v2. It is likely that you will still need to make some manual changes to achieve your desired result.

| Name | Description |
|---------------------|--|
| <code>--mdx</code> | Add this flag to convert Markdown to MDX automatically |
| <code>--page</code> | Add this flag to migrate pages automatically |

Copy

 danger

The migration of pages and MDX is still a work in progress.

We recommend you to try to run the pages without these options, commit, and then try to run the migration again with the `--page` and `--mdx` options.

This way, you'd be able to easily inspect and fix the diff.

Version: 2.0.0-beta.0

Manual migration

This manual migration process should be run after the [automated migration process](#), to complete the missing parts, or debug issues in the migration CLI output.

Project setup <#>

`package.json` <#>

Scoped package name: <#>

In Docusaurus 2, we use scoped package names:

- `docusaurus -> @docusaurus/core`

This provides a clear distinction between Docusaurus' official packages and community maintained packages. In another words, all Docusaurus' official packages are namespaced under `@docusaurus/`.

Meanwhile, the default doc site functionalities provided by Docusaurus 1 are now provided by `@docusaurus/preset-classic`. Therefore, we need to add this dependency as well:

package.json

Copy

Tip

Please use the most recent Docusaurus 2 alpha version, which you can check out [here](#) (it's tagged `next`).

CLI commands#

Meanwhile, CLI commands are renamed to `docusaurus <command>` (instead of `docusaurus-command`).

The "scripts" section of your `package.json` should be updated as follows:

package.json

Copy

A typical Docusaurus 2 `package.json` may look like this:

package.json

Copy

Update references to the `build` directory#

In Docusaurus 1, all the build artifacts are located within `website/build/<PROJECT_NAME>`.

In Docusaurus 2, it is now moved to just `website/build`. Make sure that you update your deployment configuration to read the generated files from the correct `build` directory.

If you are deploying to GitHub pages, make sure to run `yarn deploy` instead of `yarn publish-gh-pages` script.

.gitignore#

The `.gitignore` in your `website` should contain:

.gitignore

Copy

README#

The D1 website may have an existing README file. You can modify it to reflect the D2 changes, or copy the default [Docusaurus v2 README](#).

Site configurations#

Site configurations <#>

`docusaurus.config.js` <#>

Rename `siteConfig.js` to `docusaurus.config.js`.

In Docusaurus 2, we split each functionality (blog, docs, pages) into plugins for modularity. Presets are bundles of plugins and for backward compatibility we built a `@docusaurus/preset-classic` preset which bundles most of the essential plugins present in Docusaurus 1.

Add the following preset configuration to your `docusaurus.config.js`.

`docusaurus.config.js`

Copy

We recommend moving the `docs` folder into the `website` folder and that is also the default directory structure in v2. [Now](#) supports [Docusaurus project deployments out-of-the-box](#) if the `docs` directory is within the `website`. It is also generally better for the docs to be within the website so that the docs and the rest of the website code are co-located within one `website` directory.

If you are migrating your Docusaurus v1 website, and there are pending documentation pull requests, you can temporarily keep the `/docs` folder to its original place, to avoid producing conflicts.

Refer to migration guide below for each field in `siteConfig.js`.

Updated fields <#>

`baseUrl`, `tagline`, `title`, `url`, `favicon`, `organizationName`, `projectName`, `githubHost`, `scripts`, `stylesheets` <#>

No actions needed, these configuration fields were not modified.

`colors` <#>

Deprecated. We wrote a custom CSS framework for Docusaurus 2 called [Infima](#) which uses CSS variables for theming. The docs are not quite ready yet and we will update here when it is. To overwrite Infima's CSS variables, create your own CSS file (e.g. `./src/css/custom.css`) and import it globally by passing it as an option to `@docusaurus/preset-classic`:

`docusaurus.config.js`

Copy

Infima uses 7 shades of each color.

`/src/css/custom.css`

Copy

We recommend using [ColorBox](#) to find the different shades of colors for your chosen primary color.

Alternatively, use the following tool to generate the different shades for your website and copy the variables into `src/css/custom.css`.

Primary Color:

CSS Variable Name

Hex

Adjustment

| CSS Variable Name | Hex | Value |
|------------------------------|---------|-------|
| --ifm-color-primary-lightest | #808080 | -30 |
| --ifm-color-primary-lighter | #5a91a8 | -15 |
| --ifm-color-primary-light | #4e89e8 | -10 |
| --ifm-color-primary | #3578e5 | 0 |
| --ifm-color-primary-dark | #1d68e1 | 10 |
| --ifm-color-primary-darker | #1b62d4 | 15 |
| --ifm-color-primary-darkest | #1751af | 30 |

Replace the variables in `src/css/custom.css` with these new variables.

Copy

`footerIcon, copyright, ogImage, twitterImage, docsSideNavCollapsible` <#>

Site meta info such as assets, SEO, copyright info are now handled by themes. To customize them, use the `themeConfig` field in your `docusaurus.config.js`:

`docusaurus.config.js`

Copy

`headerIcon, headerLinks` <#>

In Docusaurus 1, header icon and header links were root fields in `siteConfig`:

`siteConfig.js`

Copy

Now, these two fields are both handled by the theme:

`docusaurus.config.js`

Copy

`algolia` <#>

`docusaurus.config.js`

Copy

`blogSidebarCount` <#>

Deprecated. Pass it as a blog option to `@docusaurus/preset-classic` instead:

`docusaurus.config.js`

Copy

`cname` <#>

Deprecated. Create a `CNAME` file in your `static` folder instead with your custom domain. Files in the `static` folder will be copied into the root of the `build` folder during execution of the build command.

`customDocsPath, docsUrl, editUrl, enableUpdateBy, enableUpdateTime` <#>

BREAKING: `editUrl` should point to (website) Docusaurus project instead of `docs` directory

Deprecated. Pass it as an option to `@docusaurus/preset-classic` docs instead:

docusaurus.config.js

Copy

gaTrackingId#

docusaurus.config.js

Copy

gaGtag#

docusaurus.config.js

Copy

Removed fields#

The following fields are all deprecated, you may remove from your configuration file.

- `blogSidebarTitle`
- `cleanUrl` - Clean URL is used by default now.
- `defaultVersionShown` - Versioning is not ported yet. You'd be unable to migration to Docusaurus 2 if you are using versioning. Stay tuned.
- `disableHeaderTitle`
- `disableTitleTagline`
- `docsSideNavCollapsible` is available at `themeConfig.sidebarCollapsible`, and this is turned on by default now.
- `facebookAppId`
- `facebookComments`
- `facebookPixelId`
- `fonts`
- `highlight` - We now use [Prism](#) instead of [highlight.js](#).
- `markdownOptions` - We use MDX in v2 instead of Remarkable. Your markdown options have to be converted to Remark/Rehype plugins.
- `markdownPlugins` - We use MDX in v2 instead of Remarkable. Your markdown plugins have to be converted to Remark/Rehype plugins.
- `manifest`
- `onPageNav` - This is turned on by default now.
- `separateCss` - It can imported in the same manner as `custom.css` mentioned above.
- `scrollToTop`
- `scrollToTopOptions`
- `translationRecruitingLink`
- `twitter`
- `twitterUsername`
- `useEnglishUrl`
- `users`
- `usePrism` - We now use [Prism](#) instead of [highlight.js](#)
- `wrapPagesHTML`

We intend to implement many of the deprecated config fields as plugins in future. Help will be appreciated!

Urls#

In v1, all pages were available with or without the `.html` extension.

For example, these 2 pages exist:

- <https://v1.docusaurus.io/docs/en/installation>
- <https://v1.docusaurus.io/docs/en/installation.html>

If `cleanUrl` was:

- `true`: links would target `/installation`
- `false`: links would target `/installation.html`

In v2, by default, the canonical page is `/installation`, and not `/installation.html`.

If you had `cleanUrl: false` in v1, it's possible that people published links to `/installation.html`.

For SEO reasons, and avoiding breaking links, you should configure server-side redirect rules on your hosting provider.

As an escape hatch, you could use [@docusaurus/plugin-client-redirects](#) to create client-side redirects from `/installation.html` to `/installation`.

Copy

If you want to keep the `.html` extension as the canonical url of a page, docs can declare a `slug: installation.html` frontmatter.

Components#

Sidebar#

In previous version, nested sidebar category is not allowed and sidebar category can only contain doc id. However, v2 allows infinite nested sidebar and we have many types of [Sidebar Item](#) other than document.

You'll have to migrate your sidebar if it contains category type. Rename `subcategory` to `category` and `ids` to `items`.

sidebars.json

Copy

Footer#

`website/core/Footer.js` is no longer needed. If you want to modify the default footer provided by Docusaurus, [swizzle](#) it:

- `npm`
- `Yarn`

Copy

This will copy the current `<Footer />` component used by the theme to a `src/theme/Footer` directory under the root of your site, you may then edit this component for customization.

Do not swizzle the Footer just to add the logo on the left. The logo is intentionally removed in v2 and moved to the bottom. Just configure the footer in `docusaurus.config.js` with `themeConfig.footer`:

Copy

Page:

Please refer to [creating pages](#) to learn how Docusaurus 2 pages work. After reading that, notice that you have to move `pages/en` files in v1 to `src/pages` instead.

In Docusaurus v1, pages received the `siteConfig` object as props.

In Docusaurus v2, get the `siteConfig` object from `useDocusaurusContext` instead.

In v2, you have to apply the theme layout around each page. The `Layout` component takes metadata props.

`CompLibrary` is deprecated in v2, so you have to write your own React component or use Infima styles (Docs will be available soon, sorry about that! In the meanwhile, inspect the V2 website or view <https://infima.dev/> to see what styles are available).

You can migrate CommonJS to ES6 imports/exports.

Here's a typical Docusaurus v2 page:

Copy

The following code could be helpful for migration of various pages:

- Index page - [Flux](#) (recommended), [Docusaurus 2](#), [Hermes](#)
- Help/Support page - [Docusaurus 2](#), [Flux](#)

Content#

Replace AUTOGENERATED_TABLE_OF_CONTENTS#

This feature is replaced by [inline table of content](#)

Update Markdown syntax to be MDX-compatible#

In Docusaurus 2, the markdown syntax has been changed to [MDX](#). Hence there might be some broken syntax in the existing docs which you would have to update. A common example is self-closing tags like `` and `
` which are valid in HTML would have to be explicitly closed now (`` and `
`). All tags in MDX documents have to be valid JSX.

Frontmatter is parsed by [gray-matter](#). If your frontmatter use special characters like `:`, you now need to quote it:
title: Part 1: my part1 title -> title: Part 1: "my part1 title".

Tips: You might want to use some online tools like [HTML to JSX](#) to make the migration easier.

Language-specific code tabs#

Refer to the [multi-language support code blocks](#) section.

Front matter#

The Docusaurus front matter fields for the blog have been changed from camelCase to snake_case to be consistent with the docs.

The fields `authorFBID` and `authorTwitter` have been deprecated. They are only used for generating the profile image of the author which can be done via the `author_image_url` field.

Deployment#

The `CNAME` file used by GitHub Pages is not generated anymore, so be sure you have created it in `/static/CNAME` if you use a custom domain.

The blog RSS feed is now hosted at `/blog/rss.xml` instead of `/blog/feed.xml`. You may want to configure server-side redirects so that users' subscriptions keep working.

Test your site#

After migration, your folder structure should look like this:

Copy

Start the development server and fix any errors:

Copy

You can also try to build the site for production:

Copy

Version: 2.0.0-beta.0

Versioned sites

Read up <https://docusaurus.io/blog/2018/09/11/Towards-Docusaurus-2#versioning> first for problems in v1's approach.

 note

The versioned docs should normally be migrated correctly by the [migration CLI](#)

Migrate your `versioned_docs` front matter#

Unlike v1, The markdown header for each versioned doc is no longer altered by using `version-${version}-${original_id}` as the value for the actual id field. See scenario below for better explanation.

For example, if you have a `docs/hello.md`.

Copy

When you cut a new version 1.0.0, in Docusaurus v1, `website/versioned_docs/version-1.0.0/hello.md` looks like this:

Copy

In comparison, Docusaurus 2 `website/versioned_docs/version-1.0.0/hello.md` looks like this (exactly same as

original)

Copy

Since we're going for snapshot and allow people to move (and edit) docs easily inside version. The `id` frontmatter is no longer altered and will remain the same. Internally, it is set as `version-${version}/${id}`.

Essentially, here are the necessary changes in each `versioned_docs` file:

Copy

Migrate your `versioned_sidebars` <#>

- Refer to `versioned_docs` id as `version-${version}/${id}` (v2) instead of `version-${version}-${original_id}` (v1).

Because in v1 there is a good chance someone created a new file with front matter id "`version-${version}-${id}`" that can conflict with `versioned_docs` id.

For example, Docusaurus 1 can't differentiate `docs/xxx.md`

Copy

VS `website/versioned_docs/version-1.0.0/hello.md`

Copy

Since we don't allow `/` in v1 & v2 for frontmatter, conflicts are less likely to occur.

So v1 users need to migrate their `versioned_sidebars` file

Example `versioned_sidebars/version-1.0.0-sidebars.json`:

`versioned_sidebars/version-1.0.0-sidebars.json`

Copy

Populate your `versioned_sidebars` and `versioned_docs` <#>

In v2, we use snapshot approach for documentation versioning. **Every versioned docs does not depends on other version.** It is possible to have `foo.md` in `version-1.0.0` but it doesn't exist in `version-1.2.0`. This is not possible in previous version due to Docusaurus v1 fallback functionality

<https://v1.docusaurus.io/docs/en/versioning#fallback-functionality>).

For example, if your `versions.json` looks like this in v1

`versions.json`

Copy

Docusaurus v1 creates versioned docs if **and only if the doc content is different**. Your docs structure might look like this if the only doc changed from v1.0.0 to v1.1.0 is `hello.md`.

Copy

In v2, you have to populate the missing `versioned_docs` and `versioned_sidebars` (with the right frontmatter and

id reference too).

Copy

Convert style attributes to style objects in MDX#

Docusaurus 2 uses JSX for doc files. If you have any style attributes in your Docusaurus 1 docs, convert them to style objects, like this:

Copy

Version: 2.0.0-beta.0

Translated sites

This page explains how migrate a translated Docusaurus v1 site to Docusaurus v2.

i18n differences#

Docusaurus v2 i18n is conceptually quite similar to Docusaurus v1 i18n with a few differences.

It is not tightly coupled to Crowdin, and you can use Git or another SaaS instead.

Different filesystem paths#

On Docusaurus v2, localized content is generally found at `website/i18n/<locale>`.

Docusaurus v2 is modular based on a plugin system, and each plugin is responsible to manage its own translations.

Each plugin has its own i18n subfolder, like: `website/i18n/fr/docusaurus-plugin-content-blog`

Updated translation APIs#

With Docusaurus v1, you translate your pages with `<translate>`:

Copy

On Docusaurus v2, you translate your pages with `<Translate>`

Copy

 note

The `write-translations` CLI still works to extract translations from your code.

The code translations are now added to `i18n/<lang>/code.json` using Chrome i18n JSON format.

Stricter Markdown parser#

Docusaurus v2 is using [MDX](#) to parse Markdown files.

MDX compiles Markdown files to React components, is stricter than the Docusaurus v1 parser, and will make your build fail on error instead of rendering some bad content.

Also, the HTML elements must be replaced by JSX elements.

This is particularly important for i18n because if your translations are not good on Crowdin and use invalid Markup, your v2 translated site might fail to build: you may need to do some translation cleanup to fix the errors.

Migration strategies#

This section will help you figure out how to **keep your existing v1 translations after you migrate to v2**.

There are **multiple possible strategies** to migrate a Docusaurus v1 site using Crowdin, with different tradeoffs.

⚠caution

This documentation is a best-effort to help you migrate, please help us improve it if you find a better way!

Before all, we recommend to:

- Migrate your v1 Docusaurus site to v2 without the translations
- Get familiar with the [new i18n system of Docusaurus v2](#) and
- Make Crowdin work for your v2 site, using a new and untranslated Crowdin project and the [Crowdin tutorial](#)

🔥danger

Don't try to migrate without understanding both Crowdin and Docusaurus v2 i18n.

Create a new Crowdin project#

To avoid any **risk of breaking your v1 site in production**, one possible strategy is to duplicate the original v1 Crowdin project.

ℹinfo

This strategy was used to [upgrade the Jest website](#)

Unfortunately, Crowdin does not have any "Duplicate/clone Project" feature, which makes things complicated.

- Download the translation memory of your original project in .tmx format (https://crowdin.com/project/<ORIGINAL_PROJECT>/settings#tm > View Records)
- Upload the translation memory to your new project (https://crowdin.com/project/<NEW_PROJECT>/settings#tm > View Records)
- Reconfigure `crowdin.yml` for Docusaurus v2 according to the i18n docs
- Upload the Docusaurus v2 source files with the Crowdin CLI to the new project
- Mark sensitive strings like `id` or `slug` as "hidden string" on Crowdin
- On the "Translations" tab, click on "Pre-Translation > via TM" (https://crowdin.com/project/<NEW_PROJECT>/settings#translations)
- Try first with "100% match" (more content will be translated than "Perfect"), and pre-translate your sources
- Download the Crowdin translations locally
- Try to run/build your site and see if there are any errors

You will likely have errors on your first-try: the pre-translation might try to translate things that it should not be translated (frontmatter, admonition, code blocks...), and the translated md files might be invalid for the MDX parser.

You will have to fix all the errors until your site builds. You can do that by modifying the translated md files locally, and fix your site for one locale at a time using `docusaurus build --locale fr`.

There is no ultimate guide we could write to fix these errors, but common errors are due to:

- Not marking enough strings as "hidden strings" in Crowdin, leading to pre-translation trying to translate these strings.
- Having bad v1 translations, leading to invalid markup in v2: bad html elements inside translations and unclosed tags
- Anything rejected by the MDX parser, like using HTML elements instead of JSX elements (use the [MDX playground](#) for debugging)

You might want to repeat this pre-translation process, eventually trying the "Perfect" option and limiting pre-translation only some languages/files.



Use [mdx-code-block](#) around problematic markdown elements: Crowdin is less likely mess things up with code blocks.



You will likely notice that some things were translated on your old project, but are now untranslated in your new project.

The Crowdin Markdown parser is evolving other time and each Crowdin project has a different parser version, which can lead to pre-translation not being able to pre-translate all the strings.

This parser version is undocumented, and you will have to ask the Crowdin support to know your project's parser version and fix one specific version.

Using the same cli version and parser version across the 2 Crowdin projects might give better results.



Crowdin has an "upload translations" feature, but in our experience it does not give very good results for Markdown

Use the existing Crowdin project <#>

If you don't mind modifying your existing Crowdin project and risking to mess things up, it may be possible to use the Crowdin branch system.



This workflow has not been tested in practice, please report us how good it is.

This way, you wouldn't need to create a new Crowdin project, transfer the translation memory, apply pre-translations, and try to fix the pre-translations errors.

You could create a Crowdin branch for Docusaurus v2, where you upload the v2 sources, and merge the Crowdin branch to master once ready.

Use Git instead of Crowdin[#]

It is possible to migrate away of Crowdin, and add the translation files to Git instead.

Use the Crowdin CLI to download the v1 translated files, and put these translated files at the correct Docusaurus v2 filesystem location.