

VuePress

Vue-powered Static Site Generator



Introduction

VuePress is a markdown-centered static site generator. You can write your content (documentations, blogs, etc.) in [Markdown](#), then VuePress will help you to generate a static site to host them.

The purpose of creating VuePress was to support the documentation of Vue.js and its sub-projects, but now it has been helping a large amount of users to build their documentation, blogs, and other static sites.

How It Works

A VuePress site is in fact a single-page application (SPA) powered by [Vue](#) and [Vue Router](#).

Routes are generated according to the relative path of your markdown files. Each Markdown file is compiled into HTML with [markdown-it](#) and then processed as the template of a Vue component. This allows you to directly use Vue inside your Markdown files and is great when you need to embed dynamic content.

During development, we start a normal dev-server, and serve the VuePress site as a normal SPA. If you've used Vue before, you will notice the familiar development experience when you are writing and developing with VuePress.

During build, we create a server-rendered version of the VuePress site and render the corresponding HTML by virtually visiting each route. This approach is inspired by [Nuxt](#)'s `nuxt generate` command and other projects like [Gatsby](#).

Why Not ...?

Nuxt

Nuxt is an outstanding Vue SSR framework, and it is capable of doing what VuePress does. But Nuxt is designed for building applications, while VuePress is more lightweight and focused on content-centric static sites.

VitePress

VitePress is the little brother of VuePress. It's also created and maintained by our Vue.js team. It's even more lightweight and faster than VuePress. However, as a tradeoff, it's more opinionated and less configurable. For example, it does not support plugins. But VitePress is powerful enough to make your content online if you don't need advanced customizations.

It might not be an appropriate comparison, but you can take VuePress and VitePress as Laravel and Lumen.

Docsify / Docute

Both are great projects and also Vue-powered. Except they are both fully runtime-driven and therefore not SEO-friendly. If you don't care for SEO and don't want to mess with installing dependencies, these are still great choices.

Hexo

Hexo has been serving the Vue 2.x docs well. The biggest problem is that its theming system is static and string-based - we want to take advantage of Vue for both the layout and the interactivity. Also, Hexo's Markdown rendering isn't the most flexible to configure.

GitBook

We've been using GitBook for most of our sub project docs. The primary problem with GitBook is that its development reload performance is intolerable with a large amount of files. The default theme also has a pretty limiting navigation structure, and the theming system is, again, not Vue based. The team behind GitBook is also more focused on turning it into a commercial product rather than an open-source tool.



Edit this page on GitHub [↗](#)

Last Updated: 2020/11/29 下午11:06:08





Contributors: meteorlxy

Getting Started

Prerequisites

- [Node.js v12+](#) 
- [Yarn v1 classic](#)  (Optional)

TIP

- With [pnpm](#) , you need to set `shamefully-hoist=true` in your `.npmrc`  file.
- With [yarn 2](#) , you need to set `nodeLinker: 'node-modules'` in your `.yarnrc.yml`  file.

Manual Installation

This section will help you build a basic VuePress documentation site from ground up. If you already have an existing project and would like to keep documentation inside the project, start from Step 3.

- **Step 1:** Create and change into a new directory

```
1  mkdir vuepress-starter
2  cd vuepress-starter
```

sh

- **Step 2:** Initialize your project

YARN NPM

```
1  git init
2  yarn init
```

sh

- **Step 3:** Install VuePress locally

YARN NPM

```
1 yarn add -D vuepress@next
```

sh

- **Step 4:** Add some [scripts](#) to `package.json`

```
1 {  
2   "scripts": {  
3     "docs:dev": "vuepress dev docs",  
4     "docs:build": "vuepress build docs"  
5   }  
6 }
```

json

- **Step 5:** Add the default temp and cache directory to `.gitignore` file

```
1 echo 'node_modules' >> .gitignore  
2 echo '.temp' >> .gitignore  
3 echo '.cache' >> .gitignore
```

sh

- **Step 6:** Create your first document

```
1 mkdir docs  
2 echo '# Hello VuePress' > docs/README.md
```

sh

- **Step 7:** Serve the documentation site in the local server

YARN NPM

```
1 yarn docs:dev
```

sh

VuePress will start a hot-reloading development server at <http://localhost:8080>. When you modify your markdown files, the content in the browser will be auto updated.

By now, you should have a basic but functional VuePress documentation site. Next, learn about the basics of **configuration** in VuePress.

Edit this page on GitHub [↗](#)

Last Updated: 2021/6/12 下午2:41:56

Contributors: meteorlxy, bleatingsheep

Configuration

Config File

Without any configuration, the VuePress site is pretty minimal. To customize your site, let's first create a `.vuepress` directory inside your docs directory. This is where all VuePress-specific files will be placed. Your project structure is probably like this:

```
1  | docs
2  |   | .vuepress
3  |   |   | config.js
4  |   |   | README.md
5  | .gitignore
6  | package.json
```

The essential file for configuring a VuePress site is `.vuepress/config.js`, which should export a JavaScript object. If you are using TypeScript, you can use `.vuepress/config.ts` instead to get better types hint for VuePress Config.

JS **TS**

```
1  module.exports = {
2    // site config
3    lang: 'en-US',
4    title: 'Hello, VuePress!',
5    description: 'This is my first VuePress site',
6
7    // theme and its config
8    theme: '@vuepress/theme-default',
9    themeConfig: {
10      logo: 'https://vuejs.org/images/logo.png',
11    },
12  }
```

js

TIP

Check out the [Config Reference](#) for a full list of VuePress config.

Config Scopes

You may have noticed that there is a `themeConfig` option in VuePress config.

Options outside `themeConfig` are **Site Config**, while options inside `themeConfig` are **Theme Config**.

Site Config

Site config means that, no matter what theme you are using, these configurations are always valid.

As we know, every site should have its own `lang`, `title`, `description`, etc. Thus, VuePress has built-in support for those options.

Theme Config

Theme config will be processed by VuePress theme, so it depends on the theme you are using.

If you don't specify the `theme` option of VuePress config, the default theme will be used.

TIP

Check out the [Default Theme > Config Reference](#) for theme config of default theme.

Edit this page on GitHub [↗](#)

Last Updated: 2021/11/8 上午11:43:20

Contributors: meteorlxy

Page

VuePress is markdown-centered. Each markdown file inside your project is a standalone page.

Routing

By default, the route path of a page is determined by the relative path of your markdown file.

Assuming this is the directory structure of your markdown files:

```
1  └─ docs
2    └─ guide
3      └─ getting-started.md
4        └─ README.md
5      └─ contributing.md
6    └─ README.md
```

Take the `docs` directory as your **sourceDir**, e.g. you are running `vuepress dev docs` command. Then the route paths of your markdown files would be:

Relative Path	Route Path
<code>/README.md</code>	<code>/</code>
<code>/contributing.md</code>	<code>/contributing.html</code>
<code>/guide/README.md</code>	<code>/guide/</code>
<code>/guide/page.md</code>	<code>/guide/page.html</code>

Frontmatter

A markdown file could contain a [YAML](#) frontmatter. The frontmatter must be at the top of the Markdown file and must be wrapped with a couple of triple-dashed lines. Here is a basic example:

```
1 ---
2 lang: en-US
3 title: Title of this page
4 description: Description of this page
5 ---
```

md

You must have noticed that those fields are similar with the [Site Config](#) in the [Config File](#). You can override `lang`, `title`, `description`, etc., of current page via frontmatter. So you can take frontmatter as page scope config.

Also, VuePress has built-in support for some frontmatter fields, and your theme may have its own special frontmatter, too.

TIP

Check out the [Frontmatter Reference](#) for a full list of VuePress built-in frontmatter.

Check out the [Default Theme > Frontmatter Reference](#) for the frontmatter of default theme.

Content

The main content of your page is written in Markdown. VuePress will firstly transform your Markdown to HTML code, then treat the HTML code as `<template>` of Vue SFC.

With the power of [markdown-it](#) and Vue template syntax, the basic Markdown can be extended a lot. Next, check out the [Markdown](#) guide for all the extensions of Markdown in VuePress.

[Edit this page on GitHub](#)

Last Updated: 2021/11/4 下午7:44:24

Markdown

Make sure you already know Markdown well before reading this section. If not, please learn some [Markdown tutorials](#) first.

Syntax Extensions

The Markdown content in VuePress will be parsed by [markdown-it](#), which supports [syntax extensions](#) via markdown-it plugins.

This section will introduce built-in Markdown syntax extensions of VuePress.

You can also configure those built-in extensions, load more markdown-it plugins and implement your own extensions via [markdown](#) option and [extendsMarkdown](#) option.

Embedded

Embedded by markdown-it:

- [Tables](#) (GFM)
- [Strikethrough](#) (GFM)

Header Anchors

You might have noticed that, a `#` anchor is displayed when you hover the mouse on the headers of each section. By clicking the `#` anchor, you can jump to the section directly.

TIP

This header anchors extension is supported by [markdown-it-anchor](#).

Config reference: [markdown.anchor](#)

Links

When using Markdown [link syntax](#), VuePress will implement some conversions for you.

Take our documentation source files as an example:

```
1  └─ docs
2    └─ guide
3      └─ getting-started.md
4      └─ markdown.md    # <- Here we are
5      └─ README.md
6    └─ reference
7      └─ config.md
8    └─ README.md
```

sh

Raw Markdown

```
1  <!-- relative path -->
2  [Home](../README.md)
3  [Config Reference](../reference/config.md)
4  [Getting Started](../getting-started.md)
5  <!-- absolute path -->
6  [Guide](/guide/README.md)
7  [Config Reference > markdown.links](/reference/config.md#links)
8  <!-- URL -->
9  [GitHub](https://github.com)
```

md

Converted to

```
1  <template>
2    <RouterLink to="/">Home</RouterLink>
3    <RouterLink to="/reference/config.html">Config Reference</RouterLink>
4    <RouterLink to="/guide/getting-started.html">Getting Started</RouterLink>
5    <RouterLink to="/guide/">Guide</RouterLink>
6    <RouterLink to="/reference/config.html#links">Config Reference &gt; markdown.links</RouterLink>
7    <a href="https://github.com" target="_blank" rel="noopener noreferrer">GitHub</a>
8  </template>
```

vue

Rendered as

[Home](#)

[Config Reference](#)

[Getting Started](#)

[Guide](#)

[Config Reference > markdown.links](#)

[GitHub](#) 

Explanation

- Internal links will be converted to `<RouterLink>` for SPA navigation.
- Internal links to `.md` files will be converted to the [page route path](#), and both absolute path and relative path are supported.
- External links will get `target="_blank" rel="noopener noreferrer"` attrs.

Suggestion

Try to use relative paths instead of absolute paths for internal links.

- Relative paths are a valid links to the target files, and they can navigate correctly when browsing the source files in your editor or repository.
- Relative paths are consistent in different locales, so you don't need to change the locale path when translating your content.
- When using absolute paths, if the [base](#) of your site is not `"/"`, you will need to prepend the `base` manually or use [base helper](#).

TIP

This links extension is supported by our built-in plugin.

Config reference: [markdown.links](#)

Emoji

You can add emoji to your Markdown content by typing `:EMOJICODE:`.

For a full list of available emoji and codes, check out [emoji-cheat-sheet](#) .

Input

```
1 VuePress 2 is out :tada: !
```

md

Output

VuePress 2 is out 🎉 !

TIP

This emoji extension is supported by [markdown-it-emoji](#).

Config reference: [markdown.emoji](#)

Table of Contents

If you want to put the table of contents (TOC) of your current page inside your Markdown content, you can use the `[[toc]]` syntax.

Input

```
1 [[toc]]
```

md

Output

- [Syntax Extensions](#)
 - [Embedded](#)
 - [Header Anchors](#)
 - [Links](#)
 - [Emoji 🎉](#)
 - [Table of Contents](#)
 - [Code Blocks](#)
 - [Import Code Blocks](#)
- [Using Vue in Markdown](#)
 - [Template Syntax](#)
 - [Components](#)
- [Cautions](#)

- **Non-Standard HTML Tags**

The headers in TOC will link to the corresponding **header anchors**, so TOC won't work well if you disable header anchors.

TIP

This toc extension is supported by our built-in plugin, which is forked and modified from [markdown-it-toc-done-right](#).

Config reference: [markdown.toc](#)

Code Blocks

Following code blocks extensions are implemented during markdown parsing in Node side. That means, the code blocks won't be processed in client side.

Line Highlighting

You can highlight specified lines of your code blocks by adding line ranges mark in your fenced code blocks:

Input

```
1  ```ts{1,6-8}
2  import type { UserConfig } from '@vuepress/cli'
3
4  export const config: UserConfig = {
5    title: 'Hello, VuePress',
6
7    themeConfig: {
8      logo: 'https://vuejs.org/images/logo.png',
9    },
10  }
11  ```
```

md

Output

```
1  import type { UserConfig } from '@vuepress/cli'
2
3  export const config: UserConfig = {
4    title: 'Hello, VuePress',
5
6    themeConfig: {
7      logo: 'https://vuejs.org/images/logo.png',
8    },
9  }
```

Examples for line ranges mark:

- Line ranges: `{5-8}`
- Multiple single lines: `{4,7,9}`
- Combined: `{4,7-13,16,23-27,40}`

TIP

This line highlighting extension is supported by our built-in plugin, which is forked and modified from [markdown-it-highlight-lines](#).

Config reference: [markdown.code.highlightLines](#)

Line Numbers

You must have noticed that the number of lines is displayed on the left side of code blocks. This is enabled by default and you can disable it in config.

You can add `:line-numbers` / `:no-line-numbers` mark in your fenced code blocks to override the value set in config.

Input

```
1  ```ts
2  // line-numbers is enabled by default
3  const line2 = 'This is line 2'
4  const line3 = 'This is line 3'
5  ```
6
```

```
7  ``ts:no-line-numbers
8  // line-numbers is disabled
9  const line2 = 'This is line 2'
10 const line3 = 'This is line 3'
11  ``
```

Output

```
1  // line-numbers is enabled by default
2  const line2 = 'This is line 2'
3  const line3 = 'This is line 3'
```

ts

```
// line-numbers is disabled
const line2 = 'This is line 2'
const line3 = 'This is line 3'
```

ts

TIP

This line numbers extension is supported by our built-in plugin.

Config reference: [markdown.code.lineNumbers](#)

Wrap with v-pre

As [template syntax is allowed in Markdown](#), it would also work in code blocks, too.

To avoid your code blocks being compiled by Vue, VuePress will add **v-pre** directive to your code blocks by default, which can be disabled in config.

You can add `:v-pre` / `:no-v-pre` mark in your fenced code blocks to override the value set in config.

WARNING

The template syntax characters, for example, the "Mustache" syntax (double curly braces) might be parsed by the syntax highlighter. Thus, as the following example, `:no-v-pre` might not work well in some languages.

If you want to make Vue syntax work in those languages anyway, try to disable the default syntax highlighting and implement your own syntax highlighting in client side.

Input

```
1  ```md
2  <!-- This will be kept as is by default -->
3  1 + 2 + 3 = {{ 1 + 2 + 3 }}
4  ```
5
6  ```md:no-v-pre
7  <!-- This will be compiled by Vue -->
8  1 + 2 + 3 = {{ 1 + 2 + 3 }}
9  ```
10
11 ```js:no-v-pre
12 // This won't be compiled correctly because of js syntax highlighting
13 const onePlusTwoPlusThree = {{ 1 + 2 + 3 }}
14 ```
```

Output

```
1  <!-- This will be kept as is -->
2  1 + 2 + 3 = {{ 1 + 2 + 3 }}
```

```
1  <!-- This will be compiled by Vue -->
2  1 + 2 + 3 = 6
```

```
1  // This won't be compiled correctly because of js syntax highlighting
2  const onePlusTwoPlusThree = {{ 1 + 2 + 3 }}
```

TIP

This v-pre extension is supported by our built-in plugin.

Config reference: [markdown.code.vPre](https://github.com/markdown-it/markdown-it-v-pre)

Import Code Blocks

You can import code blocks from files with following syntax:

```
1 <!-- minimal syntax -->
2 @[code](../foo.js)
```

md

If you want to partially import the file:

```
1 <!-- partial import, from line 1 to line 10 -->
2 @[code{1-10}](../foo.js)
```

md

The code language is inferred from the file extension, while it is recommended to specify it explicitly:

```
1 <!-- specify the code language -->
2 @[code js](../foo.js)
```

md

In fact, the second part inside the `[]` will be treated as the mark of the code fence, so it supports all the syntax mentioned in the above [Code Blocks](#) section:

```
1 <!-- line highlighting -->
2 @[code js{2,4-5}](../foo.js)
```

md

Here is a complex example:

- import line 3 to line 10 of the `'../foo.js'` file
- specify the language as `'js'`
- highlight line 3 of the imported code, i.e. line 5 of the `'../foo.js'` file
- disable line numbers

```
1 @[code{3-10} js{3}:no-line-numbers](../foo.js)
```

md

Notice that path aliases are not available in import code syntax. You can use following config to handle path alias yourself:

```
1  module.exports = {
2    markdown: {
3      importCode: {
4        handleImportPath: (str) =>
5          str.replace(/^@src/, path.resolve(__dirname, 'path/to/src')),
6      },
7    },
8  }
```

```
1  <!-- it will be resolved to 'path/to/src/foo.js' -->
2  @[code](@src/foo.js)
```

TIP

This import code extension is supported by our built-in plugin.

Config reference: [markdown.importCode](#)

Using Vue in Markdown

This section will introduce some basic usage of Vue in Markdown.

Check out [Cookbook > Markdown and Vue SFC](#) for more details.

Template Syntax

As we know:

- HTML is allowed in Markdown.
- Vue template syntax is compatible with HTML.

That means, [Vue template syntax](#) is allowed in Markdown.

Input

```
1 One plus one equals: {{ 1 + 1 }}
2
3 <span v-for="i in 3"> span: {{ i }} </span>
```

md

Output

One plus one equals: 2

span: 1 span: 2 span: 3

Components

You can use Vue components directly in Markdown.

Input

```
1 This is default theme built-in `<Badge />` component <Badge text="demo" />
```

md

Output

This is default theme built-in `<Badge />` component `demo`

TIP

Check out the [Built-in Components](#) for a full list of built-in components.

Check out the [Default Theme > Built-in Components](#) for a full list of default theme built-in components.

Cautions

Non-Standard HTML Tags

Non-standard HTML tags would not be recognized as native HTML tags by Vue template compiler. Instead, Vue will try to resolve those tags as Vue components, and obviously these components usually don't exist. For example:

- Deprecated HTML tags such as `<center>` and ``.
- **MathML tags**, which might be used by some markdown-it LaTeX plugin.

If you want to use those tags anyway, try either of the following workarounds:

- Adding a **v-pre** directive to skip the compilation of the element and its children. Notice that the template syntax would also be invalid.
- Using **compilerOptions.isCustomElement** to tell Vue template compiler not try to resolve them as components.
 - For `@bundler-webpack`, set **vue.compilerOptions**
 - For `@bundler-vite`, set **vuePluginOptions.template.compilerOptions**

[Edit this page on GitHub](#)

Last Updated: 2022/1/20 下午3:44:01

Contributors: meteorlxy, Matej Marjanovic, cabbage

Assets

Relative URLs

You can reference any assets using relative URLs in your Markdown content:

```
1  ![An image](./image.png)
```

md

This is generally the suggested way to import images, as users usually place images near the Markdown file that references them.

Public Files

You can put some static assets inside public directory, and they will be copied to the root of the generated directory.

The default public directory is `.vuepress/public`, which can be changed in config.

It would be useful in some cases:

- You may need to provide static assets that are not directly referenced in any of your Markdown files, for example, favicon and PWA icons.
- You may need to serve some shared static assets, which may even be referenced outside your site, for example, logo images.
- You may want to reference images using absolute URLs in your Markdown content.

Take our documentation source files as an example, we are putting the logo of VuePress inside the public directory:

```
1  └ docs
2    └ .vuepress
3      └ public
```

sh

```
4 |   └─ images
5 |     └─ hero.png # <- Logo file
6 └─ guide
7   └─ assets.md   # <- Here we are
```

We can reference our logo in current page like this:

Input

```
1 ![VuePress Logo](/images/hero.png)
```

md

Output



TIP

Config reference: [public](#)

Base Helper

If your site is deployed to a non-root URL, i.e. the **base** is not `"/"`, you will need to prepend the `base` to the absolute URLs of your public files.

For example, if you plan to deploy your site to `https://foo.github.io/bar/`, then `base` should be set to `"/bar/"`, and you have to reference your public files in Markdown like this:

```
1 ![VuePress Logo](/bar/images/hero.png)
```

md

Obviously, it is brittle if you ever decide to change the `base`. This is the reason why we suggest to reference static assets using relative URLs.

To help with that, VuePress provides a built-in helper `$withBase` that generates the correct path:

```
1 
```

md

The helper is verbose in Markdown. So it might be more helpful for theme and plugin authors.

TIP

Config reference: [base](#)

Packages and Path Aliases

Although it is not a common usage, you can reference images from dependent packages:

```
1 npm install -D package-name
```

sh

```
1 ![Image from dependency](package-name/image.png)
```

md

The path aliases that set in config file are also supported:

```
1  module.exports = {  
2    alias: {  
3      '@alias': path.resolve(__dirname, './path/to/some/dir'),  
4    },  
5  }
```

js

```
1  ![Image from path alias](@alias/image.png)
```

md

TIP

Config reference: [alias](#)

[Edit this page on GitHub](#) 

Last Updated: 2021/3/24 下午7:03:33

Contributors: meteorlxy

I18n

Site I18n Config

To take advantage of multi-language support in VuePress, you first need to use the following file and directory structure:

```
1 docs
2   └─ README.md
3   └─ foo.md
4   └─ nested
5     └─ README.md
6   └─ zh
7     └─ README.md
8     └─ foo.md
9     └─ nested
10        └─ README.md
```

Then, specify the `locales` option in your **config file**:

```
1 module.exports = {
2   locales: {
3     // The key is the path for the locale to be nested under.
4     // As a special case, the default locale can use '/' as its path.
5     '/': {
6       lang: 'en-US',
7       title: 'VuePress',
8       description: 'Vue-powered Static Site Generator',
9     },
10    '/zh/': {
11      lang: 'zh-CN',
12      title: 'VuePress',
13      description: 'Vue 驱动的静态网站生成器',
14    },
15  },
16 }
```

js

```
15     },
16   }
```

If a locale does not have a `lang`, `title`, `description` or `head`, VuePress will fallback to the root-level values. You can omit the root level config as long as they are provided in each locale.

TIP

Config reference: [locales](#)

Theme I18n Config

VuePress does not restrict how themes provide multi-language support, so each theme may have different way to handle i18n, and some themes may not provide multi-language support at all. You'd better refer to the theme documentation for detailed guide.

If you are using default theme, the multi-language support is the same as above:

```
1  module.exports = {
2    themeConfig: {
3      locales: {
4        '/': {
5          selectLanguageName: 'English',
6        },
7        '/zh/': {
8          selectLanguageName: '简体中文',
9        },
10     },
11   },
12 }
```

js

TIP

Config reference: [Default Theme > locales](#)

Edit this page on GitHub [↗](#)

Last Updated: 2021/2/20 下午7:13:23

Contributors: meteorlxy

Deployment

The following guides are based on some shared assumptions:

- You are placing your Markdown source files inside the `docs` directory of your project;
- You are using the default build output location (`.vuepress/dist`);
- You are using [yarn classic](#) as package manager, while npm is also supported;
- VuePress is installed as a local dependency in your project, and you have setup the following script in `package.json` :

```
1  {  
2    "scripts": {  
3      "docs:build": "vuepress build docs"  
4    }  
5  }
```

json

GitHub Pages

1. Set the correct `base` config.

If you are deploying to `https://<USERNAME>.github.io/`, you can omit this step as `base` defaults to `"/"`.

If you are deploying to `https://<USERNAME>.github.io/<REPO>/`, for example your repository is at `https://github.com/<USERNAME>/<REPO>`, then set `base` to `"/<REPO>/"`.

2. Choose your preferred CI tools. Here we take [GitHub Actions](#) as an example.

Create `.github/workflows/docs.yml` to set up the workflow.

▼ Click to expand sample config

```
1  name: docs
2
3  on:
4    # trigger deployment on every push to main branch
5    push:
6      branches: [main]
7    # trigger deployment manually
8    workflow_dispatch:
9
10 jobs:
11   docs:
12     runs-on: ubuntu-latest
13
14     steps:
15       - uses: actions/checkout@v2
16         with:
17           # fetch all commits to get last updated time or other git log info
18           fetch-depth: 0
19
20       - name: Setup Node.js
21         uses: actions/setup-node@v1
22         with:
23           # choose node.js version to use
24           node-version: '14'
25
26       # cache node_modules
27       - name: Cache dependencies
28         uses: actions/cache@v2
29         id: yarn-cache
30         with:
31           path: |
32             **/node_modules
33           key: ${ runner.os }-yarn-${ hashFiles('**/yarn.lock') }
34           restore-keys: |
35             ${ runner.os }-yarn-
36
37       # install dependencies if the cache did not hit
38       - name: Install dependencies
39         if: steps.yarn-cache.outputs.cache-hit != 'true'
40         run: yarn --frozen-lockfile
41
42       # run build script
43       - name: Build VuePress site
44         run: yarn docs:build
```

```

45
46     # please check out the docs of the workflow for more details
47     # @see https://github.com/crazy-max/ghaction-github-pages
48     - name: Deploy to GitHub Pages
49       uses: crazy-max/ghaction-github-pages@v2
50       with:
51         # deploy to gh-pages branch
52         target_branch: gh-pages
53         # deploy the default output dir of VuePress
54         build_dir: docs/.vuepress/dist
55       env:
56         # @see https://docs.github.com/en/actions/reference/authentication
57         GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }

```

TIP

Please refer to [GitHub Pages official guide](#) for more details.

GitLab Pages

1. Set the correct **base** config.

If you are deploying to `https://<USERNAME>.gitlab.io/`, you can omit `base` as it defaults to `"/"`.

If you are deploying to `https://<USERNAME>.gitlab.io/<REPO>/`, for example your repository is at `https://gitlab.com/<USERNAME>/<REPO>`, then set `base` to `"/<REPO>/"`.

2. Create `.gitlab-ci.yml` to set up **GitLab CI** workflow.

▼ Click to expand sample config

```

1     # choose a docker image to use
2     image: node:14-buster
3
4     pages:
5       # trigger deployment on every push to main branch

```

yml

```
6   only:
7     - main
8
9     # cache node_modules
10    cache:
11      paths:
12        - node_modules/
13
14    # install dependencies and run build script
15    script:
16      - yarn --frozen-lockfile
17      - yarn docs:build --dest public
18
19    artifacts:
20      paths:
21        - public
```

TIP

Please refer to [GitLab Pages official guide](#) for more details.

Google Firebase

1. Make sure you have [firebase-tools](#) installed.
2. Create `firebase.json` and `.firebaserc` at the root of your project with the following content:

`firebase.json` :

```
1  {
2    "hosting": {
3      "public": "./docs/.vuepress/dist",
4      "ignore": []
5    }
6  }
```

json

`.firebaserc` :

```
1  {
2    "projects": {
3      "default": "<YOUR_FIREBASE_ID>"
4    }
5  }
```

json

3. After running `yarn docs:build` , deploy using the command `firebase deploy` .

TIP

Please refer to [Firebase CLI official guide](#) for more details.

Heroku

1. Install [Heroku CLI](#).
2. Create a Heroku account by [signing up](#).
3. Run `heroku login` and fill in your Heroku credentials:

```
1  heroku login
```

sh

4. Create a file called `static.json` in the root of your project with the below content:

`static.json` :

```
1  {
2    "root": "./docs/.vuepress/dist"
3  }
```

json

This is the configuration of your site; read more at [heroku-buildpack-static](#).

Layer0

See [Layer0 Documentation > Framework Guides > VuePress](#).

Netlify

1. On [Netlify](#), set up a new project from GitHub with the following settings:

- **Build Command:** `yarn docs:build`
- **Publish directory:** `docs/.vuepress/dist`

2. Set [Environment variables](#) to choose node version:

- `NODE_VERSION` : 14

3. Hit the deploy button.

Vercel

See [Creating and Deploying a VuePress App with Vercel](#).

[Edit this page on GitHub](#)

Last Updated: 2021/12/24 下午7:22:41

Contributors: meteorlxy, Rishi Raj Jain

Theme

VuePress theme can provide layouts, styles and many other features for you, helping you to focus on writing Markdown content.

VuePress has a default theme out of the box, which is applied to our documentation site you are currently browsing. The default theme provides basic but useful features for documentation site, you can check out [Default Theme Config Reference](#) for a full list of config.

However, you might think it is not good enough. Or, you want to build a different type of site, for example, a blog, instead of a documentation. Then, you can try to [use a community theme](#) or [create a local theme](#).

Community Theme

Community users have created lots of theme and published them to [NPM](#)[↗]. You should check the theme's own documentation for detailed guide.

In general, you need to specify the name of the theme to use in **theme** option:

```
1 module.exports = {  
2   theme: 'foo',  
3 }
```

js

You can use either theme name or its shorthand:

Theme Name	Shorthand
<code>vuepress-theme-foo</code>	<code>foo</code>
<code>@org/vuepress-theme-bar</code>	<code>@org/bar</code>
<code>@vuepress/theme-default</code>	<code>@vuepress/default</code>

Local Theme

If you want to use your own custom theme but don't want to publish it, you can create a local theme.

First, create the local theme directory, typically `.vuepress/theme` :

```
1  └─ docs
2    └─ .vuepress
3      └─ theme
4        └─ index.js
5      └─ config.js
6    └─ README.md
```

Then, set the `theme` option to the absolute path of the `theme entry` to use it:

```
1  module.exports = {
2    theme: path.resolve(__dirname, './path/to/docs/.vuepress/theme/index.js'),
3  }
```

js

Next, refer to [Advanced > Writing a Theme](#) for how to write your own theme.

[Edit this page on GitHub](#) 

Last Updated: 2021/11/8 上午11:43:20

Contributors: meteorlxy

Plugin

With the help of [Plugin API](#), VuePress plugin can provide different features for you.

Community Plugin

Community users have created lots of plugins and published them to [NPM](#). VuePress team also maintains some official plugins under the [@vuepress](#) scope. You should check the plugin's own documentation for detailed guide.

In general, you need to specify the name of the plugin to use in [plugins](#) option:

```
1  module.exports = {  
2    plugins: [  
3      'foo',  
4      ['bar', { /* options */ }]  
5    ],  
6  }
```

js

You can use either plugin name or its shorthand:

Plugin Name	Shorthand
vuepress-plugin-foo	foo
@org/vuepress-plugin-bar	@org/bar
@vuepress/plugin-foobar	@vuepress/foobar

TIP

Most plugins can only be used once. If the same plugin is used multiple times, only the last one will take effect.

However, some plugins can be used multiple times (e.g. [@vuepress/plugin-container](#)), and you should check the documentation of the plugin itself for detailed guide.

Local Plugin

If you want to use your own plugin but don't want to publish it, you can create a local plugin.

It is recommended to use the [Config File](#) directly as a plugin, because [almost all of the Plugin APIs are available](#), which would be more convenient in most cases.

But if you have too many things to do in your config file, it's better to extract them into separate plugins, and use them by setting the absolute path to them or requiring them:

```
1  module.exports = {  
2    plugins: [  
3      path.resolve(__dirname, './path/to/your-plugin.js'),  
4      require('./another-plugin'),  
5    ],  
6  }
```

js

You can refer to [Advanced > Writing a Plugin](#) for how to write your own plugin.

[Edit this page on GitHub](#) [↗](#)

Last Updated: 2021/6/9 下午5:16:39

Contributors: meteorlxy

Bundler

VuePress has been using [Webpack](#) as the bundler to dev and build sites. Since VuePress v2, other bundlers are also supported, and now we are using [Vite](#) as the default bundler. Of course, you can still choose to use Webpack.

Choose a Bundler

When using the [vuepress](#) package, Vite bundler is installed and used automatically.

If you want to use Webpack bundler instead, you can switch to [vuepress-webpack](#) package:

```
YARN  NPM
1  yarn remove vuepress
2  yarn add -D vuepress-webpack@next
```

Bundler Config

Generally, you could use a bundler without extra configuration, because we have already configured them properly to work with VuePress.

Similar to [themeConfig](#), VuePress also allows users to set bundler config via [bundlerConfig](#).

You can refer to [Bundlers > Vite](#) and [Bundlers > Webpack](#) to check out all options of the corresponding bundler.

[Edit this page on GitHub](#)

Last Updated: 2021/12/17 下午5:49:12

Migrating from v1

WARNING

Plugins and themes of VuePress v1 are not compatible with VuePress v2. You need to update them to corresponding v2 version.

Some major changes and enhancements of VuePress v2:

- VuePress v2 is now using Vue 3, so make sure your components and other client files are compatible with Vue 3.
- VuePress v2 is developed with TypeScript, so it provides better TS support now. It's highly recommended to use TypeScript to develop plugins and themes. VuePress config file also supports TypeScript, and you can use `.vuepress/config.ts` directly.
- VuePress v2 add supports both Webpack and Vite as bundler. Now Vite is the default bundler, while you can still choose use Webpack. You can even use Vite in dev mode to get better development experience, and use Webpack in build mode to get better browser compatibility.

Core ideas and processes of VuePress v2 are the same with v1, while v2 API has been re-designed and becomes more normalized. So you might encounter breaking changes when migrating an existing v1 project to v2. This guide is here to help you migrating v1 sites / plugins / themes to v2.

- If you are a common user, you need to read the guide [for users](#).
- If you are a plugin author, you need to read the guide [for plugin authors](#).
- If you are a theme author, you need to read the guide [for theme authors](#).

For Users

User Config Change

`shouldPrefetch`

Default value is changed from `() => true` to `false` .

extraWatchFiles

Removed.

You can watch files manually in `onWatched` hook.

plugins

Only `Babel Style` [↗](#) config is allowed.

`Object Style` [↗](#) config is **NOT** supported in v2.

patterns

Renamed to `pagePatterns`

markdown.lineNumbers

Moved to `markdown.code.lineNumbers`.

Default value is changed from `false` to `true` .

markdown.slugify

Removed.

If you want to change the slugify function anyway, set the following options separately:

- `markdown.anchor.slugify`
- `markdown.toc.slugify`
- `markdown.extractHeaders.slugify`

markdown.pageSuffix

Removed.

markdown.externalLinks

Moved to [markdown.links.externalAttrs](#).

markdown.toc

Changed.

See [Config > markdown.toc](#)

markdown.plugins

Removed.

Use markdown-it plugins in [extendsMarkdown](#) hook.

markdown.extendMarkdown

Removed.

Use [extendsMarkdown](#) hook.

markdown.extractHeaders

Changed.

See [Config > markdown.extractHeaders](#)

Webpack Related Configs

All webpack related configs are moved to options of `@vuepress/bundler-webpack`, so you should set them in [bundlerConfig](#):

- `postcss` : moved to `bundlerConfig.postcss`
- `stylus` : moved to `bundlerConfig.stylus`
- `scss` : moved to `bundlerConfig.scss`
- `sass` : moved to `bundlerConfig.sass`

- `less` : moved to `bundlerConfig.less`
- `chainWebpack` : moved to `bundlerConfig.chainWebpack`
- `configureWebpack` : moved to `bundlerConfig.configureWebpack`
- `evergreen` : moved to `bundlerConfig.evergreen` , and default value is changed from `false` to `true` .

See [Bundlers > Webpack](#)

Frontmatter Change

meta

Removed.

Use [head](#) instead. For example:

```
1  head:
2    - - meta
3      - name: foo
4        content: bar
5    - - link
6      - rel: canonical
7        href: foobar
8    - - script
9      - {}
10   - console.log('hello from frontmatter');
```

yml

Has the same structure with:

```
1  // .vuepress/config.js
2  module.exports = {
3    // ...
4    head: [
5      ['meta', { name: 'foo', content: 'bar' }],
6      ['link', { rel: 'canonical', href: 'foobar' }],
7      ['script', {}, `console.log('hello from frontmatter');`],
8    ],
9    // ...
10 }
```

js

Permalink Patterns Change

- `:i_month` : removed
- `:i_day` : removed
- `:minutes` : removed (undocumented in v1)
- `:seconds` : removed (undocumented in v1)
- `:regular` : renamed to `:raw`

See [Frontmatter > permalinkPattern](#).

Palette System Change

The stylus palette system of VuePress v1 (i.e. `styles/palette.styl` and `styles/index.styl`) is no longer provided by VuePress Core.

The palette system is extracted to [@vuepress/plugin-palette](#).

Theme authors can use their own way to allow users to customize styles, and not be limited with stylus.

If you are using default theme, the palette system is still available but migrated to SASS, while most variables have been migrated to CSS variables. See [Default Theme > Styles](#).

Conventional Files Change

`.vuepress/enhanceApp.js`

Renamed to `.vuepress/clientAppEnhance.{js,ts}` .

The arguments of the function are changed, too.

See [Client API > defineClientAppEnhance](#).

`.vuepress/components/`

Files in this directory will not be registered as Vue components automatically.

You need to use [@vuepress/plugin-register-components](#), or register your components manually in `.vuepress/clientAppEnhance.{js,ts}` .

`.vuepress/theme/`

This directory will not be used as local theme implicitly if it is existed.

You need to set the path to the local theme explicitly via [theme](#) option.

Markdown slot Change

Markdown slot is no longer supported.

Plugin API Change

- `plugins` : removed
- `ready` : renamed to `onPrepared`
- `updated` : renamed to `onWatched`
- `generated` : renamed to `onGenerated`
- `additionalPages` : removed, use `app.pages.push(createPage())` in `onInitialized` hook
- `clientDynamicModules` : removed, use `app.writeTemp()` in `onPrepared` hook
- `enhanceAppFiles` : renamed to `clientAppEnhanceFiles`
- `globalUIComponents` : renamed to `clientAppRootComponentFiles`
- `clientRootMixin` : renamed to `clientAppSetupFiles`
- `extendMarkdown` : renamed to `extendsMarkdown`
- `chainMarkdown` : removed
- `extendPageData` : renamed to `extendsPage`
- `extendsCli` : removed
- `configureWebpack` : removed
- `chainWebpack` : removed
- `beforeDevServer` : removed
- `afterDevServer` : removed

See [Plugin API](#).

Theme API Change

layouts

Now you need to specify the layouts directory or layout components manually.

See [Theme API > layouts](#).

extend

Renamed to `extends` .

You can still inherit a parent theme with `extends: 'parent-theme'` , which will extends the plugins, layouts, etc.

The `@theme` and `@parent-theme` aliases are removed by default, but you can still replace components in default theme with similar approach.

You can refer to [Default Theme > Extending](#) for how to extend default theme.

CLI Change

eject command

Removed.

cache options

- `-c, --cache [cache]` : changed to `--cache <cache>` , which means that the shorthand `-c` is not for `cache` option, and the value of `cache` option is not optional.
- `--no-cache` : renamed to `--clean-cache` .

Default Theme Change

Built-in Components

- `<CodeGroup />` and `<CodeBlock />` renamed to `<CodeGroup />` and `<CodeGroupItem />`
- `<Badge />`

- `$badgeErrorColor` palette variable renamed to `$badgeDangerColor`
- `type` prop only accepts `tip`, `warning` and `danger` now

Palette System

The palette system of default theme has migrated to SASS and CSS variables.

See [Default Theme > Styles](#).

Theme Config

Default theme config has changed a lot.

See [Default Theme > Config](#).

Official Plugins Change

See [Official Plugins](#).

Community Themes and Plugins

Themes and plugins of v1 is not compatible with v2.

Please make sure that those themes and plugins you are using have supported v2, and refer to their own documentation for migration guide.

For Plugin Authors

Read the [Plugin API Change](#) first.

Some major breaking changes:

- You cannot use other plugins in your plugin anymore, which avoids lots of potential issues caused by plugin nesting. If your plugin depends on other plugins, you should list them in the docs.
- Most of the v1 hooks have equivalents in v2. The only exception is `extendsCli`, which has been removed.

- Webpack related hooks are removed, because VuePress Core has decoupled with webpack. If you still want to modify webpack config in plugin, try modifying `app.options.bundlerConfig` directly.

For Theme Authors

Read the [Plugin API Change](#) and [Theme API Change](#) first.

Although we do not allow using other plugins in a plugin, you can still use plugins in your theme.

Some major breaking changes:

- There is no **conventional theme directory structure** anymore.
 - The file `theme/enhanceApp.js` or `theme/clientAppEnhance.{js,ts}` will not be used as client app enhance file implicitly. You need to specify it explicitly in `clientAppEnhanceFiles` hook.
 - Files in `theme/global-components/` directory will not be registered as Vue components automatically. You need to use [@vuepress/plugin-register-components](#), or register components manually in `clientAppEnhance.{js,ts}`.
 - Files in `theme/layouts/` directory will not be registered as layout components automatically. You need to specify it explicitly in `layouts` option.
 - Files in `theme/templates/` directory will not be used as dev / ssr template automatically.
 - Always provide a theme entry file, and do not use `"main": "layouts/Layout.vue"` as the theme entry.
- Stylus is no longer the default CSS pre-processor, and the stylus palette system is not embedded. If you still want to use similar palette system as v1, [@vuepress/plugin-palette](#) may help.
- Markdown code blocks syntax highlighting by Prism.js is not embedded by default. You can use either [@vuepress/plugin-prismjs](#) or [@vuepress/plugin-shiki](#), or implement syntax highlighting in your own way.
- For scalability concern, `$site.pages` is not available any more.

[Edit this page on GitHub](#) 

Last Updated: 2022/1/11 下午8:02:48

Contributors: meteorlxy, Julio

