VuePress

Vue-powered Static Site Generator

Introduction

VuePress is composed of two parts: a minimalistic static site generator with a Vue-powered theming system and Plugin API, and a default theme optimized for writing technical documentation. It was created to support the documentation needs of Vue's own sub projects.

Each page generated by VuePress has its own pre-rendered static HTML, providing great loading performance and is SEO-friendly. Yet, once the page is loaded, Vue takes over the static content and turns it into a full Single-Page Application (SPA). Extra pages are fetched on demand as the user navigates around the site.

How It Works

A VuePress site is in fact a SPA powered by Vue , Vue Router and webpack . If you've used Vue before, you will notice the familiar development experience when you are writing or developing custom themes (you can even use Vue DevTools to debug your custom theme!).

During the build, we create a server-rendered version of the app 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 .

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.

Features

Built-in Markdown extensions

- Table of Contents
- Custom Containers
- Line Highlighting
- Line Numbers
- Import Code Snippets

Using Vue in Markdown

- Templating
- Using Components

Vue-powered custom theme system

- Metadata
- Content Excerpt

Default theme

- Responsive layout
- Optional Homepage
- Simple out-of-the-box header-based search
- Algolia Search
- Customizable navbar and sidebar
- Auto-generated GitHub link and page edit links
- PWA: Popup UI to refresh contents
- Last Updated
- Multi-Language Support

Blog theme

- Documentation
- Live Example

Plugin

- Powerful Plugin API
- Blog Plugin
- Search Plugin
- PWA Plugin
- Google Analytics Plugin
- ...

Why Not ...?

Nuxt

Nuxt is capable of doing what VuePress does, but it’s designed for building applications. VuePress is focused on content-centric static sites and provides features tailored for technical documentation out of the box.

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 docs well - in fact, we are probably still a long way to go from migrating away from it for our main site. 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.

Last Updated: 11/29/2021, 7:08:57 AM

Getting Started

Prerequisites

- Node.js 10+
- Yarn Classic (Optional)*

** If your project is using webpack 3.x, you may notice some installation issues with `npm`. In this case, we recommend using Yarn.*

Quick Start

The fastest way to get your VuePress project set up is to use our create-vuepress-site generator , which will help scaffold the basic VuePress site structure for you.

To use it, open up your terminal in the desired directory and run the following command:

YARN NPM

```sh
yarn create vuepress-site [optionalDirectoryName]
```

The command will interactively ask for details to configure your VuePress site's metadata such as:

- Project Name
- Description
- Maintainer Email
- Maintainer Name
- Repository URL

Once this done, a scaffolded documentation site will be created in the `docs` directory (or custom directory name, if passed) under the current directory.

To see it in action, navigate into newly scaffolded directory, install the dependencies and start the local server:

YARN NPM

```sh
cd docs
yarn install
yarn dev
```

Manual Installation

If you prefer, you can build a basic VuePress documentation site from ground up instead of using the generator mentioned above.

Note: If you already have an existing project and would like to keep documentation inside the project, start from Step 3.

1. Create and change into a new directory

   ```sh
   mkdir vuepress-starter && cd vuepress-starter
   ```

2. Initialize with your preferred package manager

   YARN NPM

   ```sh
   yarn init
   ```

3. Install VuePress locally

   YARN NPM

   ```sh
   yarn add -D vuepress
   ```

4. Create your first document

   ```sh
   mkdir docs && echo '# Hello VuePress' > docs/README.md
   ```

5. Add helper scripts to `package.json`

This step is optional but highly recommended, as the rest of the documentation will assume those scripts being present.

```json
{
  "scripts": {
    "docs:dev": "vuepress dev docs",
    "docs:build": "vuepress build docs"
  }
}
```

6. Serve the documentation site in the local server

YARN NPM

```sh
yarn docs:dev
```

VuePress will start a hot-reloading development server at http://localhost:8080 .

By now, you should have a basic but functional VuePress documentation site. Next, learn about VuePress' recommended directory structure and the basics of configuration in VuePress.

Once you're familiar with those concepts mentioned above, learn how to enrich your content with static assets, Markdown extensions and vue components.

And when your documentation site starts to take shape, be sure to read about multi-language support and the deployment guide.

Last Updated: 7/9/2021, 4:21:25 AM

Directory Structure

VuePress follows the principle of **"Convention is better than configuration"**. The recommended structure is as follows:

```
.
├── docs
│   ├── .vuepress (Optional)
│   │   ├── components (Optional)
│   │   ├── theme (Optional)
│   │   │   └── Layout.vue
│   │   ├── public (Optional)
│   │   ├── styles (Optional)
│   │   │   ├── index.styl
│   │   │   └── palette.styl
│   │   ├── templates (Optional, Danger Zone)
│   │   │   ├── dev.html
│   │   │   └── ssr.html
│   │   ├── config.js (Optional)
│   │   └── enhanceApp.js (Optional)
│   │
│   ├── README.md
│   ├── guide
│   │   └── README.md
│   └── config.md
│
└── package.json
```

Note

Please note the capitalization of the directory name.

- `docs/.vuepress`: Used to store global configuration, components, static resources, etc.
- `docs/.vuepress/components`: The Vue components in this directory will be automatically registered as global components.

- `docs/.vuepress/theme` : Used to store local theme.
- `docs/.vuepress/styles` : Stores style related files.
- `docs/.vuepress/styles/index.styl` : Automatically applied global style files, generated at the ending of the CSS file, have a higher priority than the default style.
- `docs/.vuepress/styles/palette.styl` : The palette is used to override the default color constants and to set the color constants of Stylus.
- `docs/.vuepress/public` : Static resource directory.
- `docs/.vuepress/templates` : Store HTML template files.
- `docs/.vuepress/templates/dev.html` : HTML template file for development environment.
- `docs/.vuepress/templates/ssr.html` : Vue SSR based HTML template file in the built time.
- `docs/.vuepress/config.js` : Entry file of configuration, can also be `yml` or `toml` .
- `docs/.vuepress/enhanceApp.js` : App level enhancement.

> **Note**
>
> When customizing `templates/ssr.html` , or `templates/dev.html` , it's best to edit it on the basis of the **default template files** , otherwise it may cause a build failure.

Default Page Routing

Here we use the `docs` directory as the `targetDir` (see Command-line Interface). All the "Relative Paths" below are relative to the `docs` directory. Add `scripts` in `package.json` which is located in your project's root directory:

```json
{
  "scripts": {
    "dev": "vuepress dev docs",
    "build": "vuepress build docs"
  }
}
```

For the above directory structure, the default page routing paths are as follows:

Relative Path	Page Routing
`/README.md`	`/`

Relative Path	Page Routing
`/guide/README.md`	`/guide/`
`/config.md`	`/config.html`

Also see:

- Config
- Theme
- Default Theme Config

Last Updated: 7/23/2020, 7:11:42 AM

Configuration

Config File

Without any configuration, the page is pretty minimal, and the user has no way to navigate around the site. 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:

```
.
├─ docs
│  ├─ README.md
│  └─ .vuepress
│     └─ config.js
└─ package.json
```

The essential file for configuring a VuePress site is `.vuepress/config.js`, which should export a JavaScript object:

```js
module.exports = {
  title: 'Hello VuePress',
  description: 'Just playing around'
}
```

If you've got the dev server running, you should see the page now has a header with the title and a search box. VuePress comes with built-in headers-based search: it automatically builds a simple search index from the title, `h2`, and `h3` headers on all pages.

Check out the Config Reference for a full list of options.

Alternative Config Formats

> You can also use TS(`.vuepress/config.ts`), YAML (`.vuepress/config.yml`) or TOML (`.vuepress/config.toml`) formats for the configuration file, for detail of TS support, you can move TypeScript as Config.

Theme Configuration

A VuePress theme owns all the layout and interactivity details of your site. VuePress ships with a default theme (you are looking at it right now), designed for technical documentation. It exposes many options that allow you to customize the navbar, sidebar and homepage, etc. For details, check out the Default Theme Config page.

To develop a custom theme, see **Writing a theme**.

App Level Enhancements

Since the VuePress app is a standard Vue app, you can apply app-level enhancements by creating a file `.vuepress/enhanceApp.js` , which will be imported into the app if present. The file should `export default` a hook function which will receive an object containing some app-level values. You can use this hook to install extra Vue plugins, register global components, or add extra router hooks:

```js
// async function is also supported, too
export default ({
  Vue, // the version of Vue being used in the VuePress app
  options, // the options for the root Vue instance
  router, // the router instance for the app
  siteData, // site metadata
  isServer // is this enhancement applied in server-rendering or client
}) => {
  // ...apply enhancements to the app
}
```

Related:

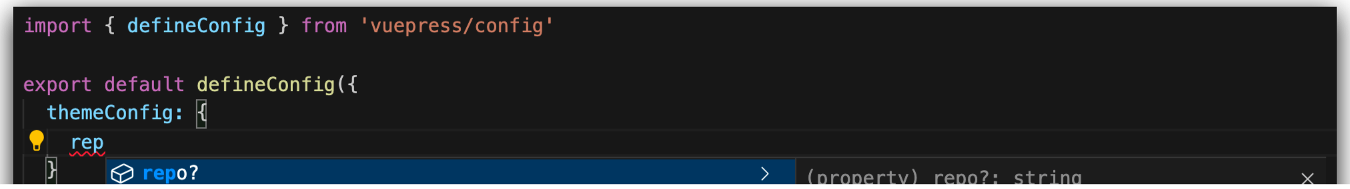
- App Level Enhancements in Plugin API

Last Updated: 12/22/2021, 8:05:29 PM

TypeScript as Config 1.9.0+

Overview

VuePress supports type prompt and type checking for config file, as well as type prompt for default theme or custom theme.



```
import { defineConfig } from 'vuepress/config'

export default defineConfig({
  themeConfig: {
    rep
  }
```

Quick Start

Creating `.vuepress/config.ts` with following contents:

```ts
import { defineConfig } from "vuepress/config";

export default defineConfig({
  // ...
});
```

Type Inferences for Theme

By default, `defineConfig` helper leverages the theme config type from default theme:

```js
import { defineConfig } from "vuepress/config";

export default defineConfig({
  /**
   * Type is `DefaultThemeConfig`
   */
  themeConfig: {
    repo: "vuejs/vuepress",
    editLinks: true,
    docsDir: "packages/docs/docs"
  }
});
```

If you use a custom theme, you can use the `defineConfig4CustomTheme` helper with ability to pass generic type for your theme:

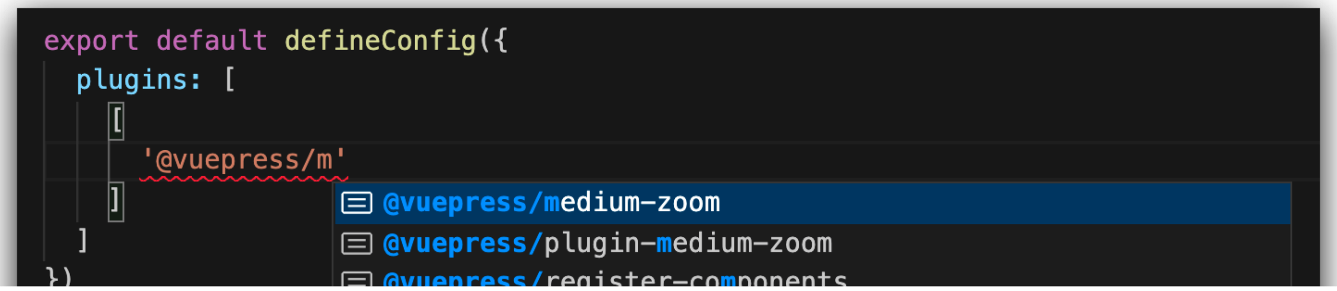
```ts
import { defineConfig4CustomTheme } from "vuepress/config";

interface MyThemeConfig {
  hello: string;
}

export default defineConfig4CustomTheme<MyThemeConfig>({
  /**
   * Type is `MyThemeConfig`
   */
  themeConfig: {
    hello: "vuepress"
  }
});
```

Type Inferences for Official Plugins

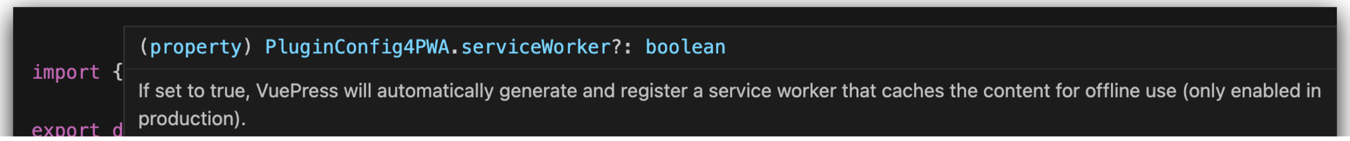
You'll be able to enjoy the type prompt of the official plugins:



```
export default defineConfig({
  plugins: [
    [
      '@vuepress/m'
    ]
  ]
})
```

Options of the official plugins certainly have type prompts, **Both Tuple Style, Object Style, and Plugin Shorthand support type inference!**:

- Tuple Style:



```
(property) PluginConfig4PWA.serviceWorker?: boolean
```

If set to true, VuePress will automatically generate and register a service worker that caches the content for offline use (only enabled in production).

```ts
import { defineConfig } from "vuepress/config";

export default defineConfig({
  plugins: [
    [
      "@vuepress/pwa",
      {
        serviceWorker: true
      }
    ]
  ]
});
```

- Object Style:

```ts
import { defineConfig } from "vuepress/config";

export default defineConfig({
  plugins: {
    "@vuepress/pwa": {
      serviceWorker: true
    }
  }
});
```

The illustration snapshot is omitted here, you can try it yourself.

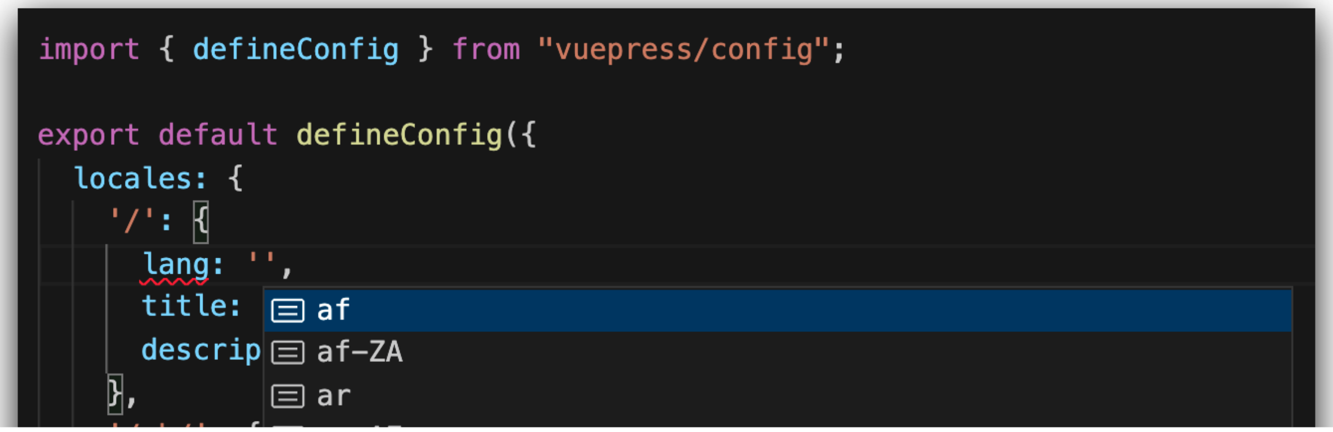
Third Plugins

It is worth noting that third-party plugins do not support Plugin Shorthand if you're using Tuple Style to write your config, this is because from the perspective of the type system, the unknown shortcut is equivalent to `string`, which results in the failure of type inference.

By default, only officially maintained and plugins under VuePress Community support shortcut, feel free to submit pull request to add your plugin at this file .

ISO Language Code

Type inference supports ISO Language Code for i18n .



```
import { defineConfig } from "vuepress/config";

export default defineConfig({
  locales: {
    '/': {
      lang: '',
      title:  af
      descrip af-ZA
    },        ar
```

Context API

VuePress's configuration can also be a function, while its first parameter is the current app context:

```ts
import { defineConfig } from "vuepress/config";

export default defineConfig(ctx => ({
  // do not execute babel compilation under development
  evergreen: ctx.isProd
}));
```

Last Updated: 12/23/2021, 6:50:55 PM

Asset Handling

Relative URLs

All Markdown files are compiled into Vue components and processed by webpack . You can, **and should**, reference any assets using relative URLs:

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

This would work the same way as in `*.vue` file templates. The image will be processed with `url-loader` and `file-loader` , and copied to appropriate locations in the generated static build.

You can also use the `~` prefix to explicitly specify this is a webpack module request, allowing you to reference files with webpack aliases or from npm dependencies:

```md
![Image from alias](~@alias/image.png)
![Image from dependency](~some-dependency/image.png)
```

One alias that is added by default is `@source` , if you follow the recommended Directory Structure this is the `docs` folder.

```md
![Image from images folder](~@source/images/image.png)
```

Webpack aliases can be configured via configureWebpack in `.vuepress/config.js` . Example:

```js
module.exports = {
  configureWebpack: {
    resolve: {
      alias: {
        '@alias': 'path/to/some/dir'
      }
    }
  }
}
```

Public Files

Sometimes you may need to provide static assets that are not directly referenced in any of your Markdown or theme components (for example, favicons and PWA icons). In such cases, you can put them inside `.vuepress/public` and they will be copied to the root of the generated directory.

Base URL

If your site is deployed to a non-root URL, you will need to set the `base` option in `.vuepress/config.js`. For example, if you plan to deploy your site to `https://foo.github.io/bar/`, then `base` should be set to `"/bar/"` (it should always start and end with a slash).

With a base URL, to reference an image in `.vuepress/public`, you'd have to use URLs like `/bar/image.png`. But this is brittle if you ever decide to change the `base`. To help with that, VuePress provides a built-in helper `$withBase` (injected onto Vue's prototype) that generates the correct path:

```vue
<img :src="$withBase('/foo.png')" alt="foo">
```

Note you can use the above syntax not only in theme components, but in your Markdown files as well.

Also, if a `base` is set, it's automatically prepended to all asset URLs in `.vuepress/config.js` options.

Last Updated: 12/20/2021, 5:49:39 PM

Markdown Extensions

Header Anchors

Headers automatically get anchor links applied. Rendering of anchors can be configured using the `markdown.anchor` option.

Links

Internal Links

Internal links are converted to `<router-link>` for SPA navigation. Also, every `README.md` or `index.md` contained in each sub-directory will automatically be converted to `index.html`, with corresponding URL `/`.

For example, given the following directory structure:

```
.
├─ README.md
├─ foo
│  ├─ README.md
│  ├─ one.md
│  └─ two.md
└─ bar
   ├─ README.md
   ├─ three.md
   └─ four.md
```

And providing you are in `foo/one.md`:

```md
[Home](/) <!-- Sends the user to the root README.md -->
[foo](/foo/) <!-- Sends the user to index.html of directory foo -->
[foo heading](./#heading) <!-- Anchors user to a heading in the foo README file -->
```

```
  [bar - three](../bar/three.md) <!-- You can append .md (recommended) -->
  [bar - four](../bar/four.html) <!-- Or you can append .html -->
```

Redirection for URLs

VuePress supports redirecting to clean links. If a link `/foo` is not found, VuePress will look for a existing `/foo/` or `/foo.html`. Conversely, when one of `/foo/` or `/foo.html` is not found, VuePress will try the other. With this feature, we can customize your website's URLs with the official plugin vuepress-plugin-clean-urls .

> **TIP**
>
> Regardless of whether the permalink and clean-urls plugins are used, your relative path should be defined by the current file structure. In the above example, even though you set the path of `/foo/one.md` to `/foo/one/`, you should still access `/foo/two.md` via `./two.md`.

Page Suffix

Pages and internal links get generated with the `.html` suffix by default.

You can customize this by setting config.markdown.pageSuffix.

External Links

Outbound links automatically get `target="_blank" rel="noopener noreferrer"`:

- vuejs.org
- VuePress on GitHub

You can customize the attributes added to external links by setting config.markdown.externalLinks.

Frontmatter

YAML frontmatter is supported out of the box:

```yaml
---
title: Blogging Like a Hacker
lang: en-US
---
```

This data will be available to the rest of the page, along with all custom and theming components.

For more details, see Frontmatter.

GitHub-Style Tables

Input

```
| Tables        | Are           | Cool  |
| ------------- |:-------------:| -----:|
| col 3 is      | right-aligned | $1600 |
| col 2 is      | centered      |   $12 |
| zebra stripes | are neat      |    $1 |
```

Output

Tables	Are	Cool
col 3 is	right-aligned	$1600
col 2 is	centered	$12
zebra stripes	are neat	$1

Emoji 🎉

Input

```
:tada: :100:
```

Output

🎉 💯

A list of all emojis is available.

Table of Contents

Input

```
[[toc]]
```

Output

- Header Anchors
- Links
  - Internal Links
  - Redirection for URLs
  - Page Suffix
  - External Links
- Frontmatter
- GitHub-Style Tables
- Emoji 🎉
- Table of Contents
- Custom Containers default theme
  - Default Title
  - Custom Title
- Syntax Highlighting in Code Blocks
- Line Highlighting in Code Blocks
- Line Numbers
- Import Code Snippets beta
- Advanced Configuration

Rendering of the TOC can be configured using the `markdown.toc` option.

Custom Containers default theme

Custom containers can be defined by their types, titles, and contents.

Default Title

Input

```md
::: tip
This is a tip
:::

::: warning
This is a warning
:::

::: danger
This is a dangerous warning
:::

::: details
This is a details block, which does not work in IE / Edge
:::
```

Output

TIP

This is a tip

WARNING

This is a warning

DANGER

This is a dangerous warning

▼ DETAILS

This is a details block, which does not work in Internet Explorer or Edge.

Custom Title

Input

~~~md
::: danger STOP
Danger zone, do not proceed
:::

::: details Click me to view the code
```js
console.log('Hello, VuePress!')
~~~
:::

Output

STOP

Danger zone, do not proceed

▼ Click me to view the code

```js
console.log('Hello, VuePress!')
```

Also see:

- vuepress-plugin-container

Syntax Highlighting in Code Blocks

VuePress uses Prism to highlight language syntax in Markdown code blocks, using coloured text. Prism supports a wide variety of programming languages. All you need to do is append a valid language alias to the beginning backticks for the code block:

Input

~~~
``` js
export default {
  name: 'MyComponent',
  // ...
}
```
~~~

Output

js

```js
export default {
  name: 'MyComponent',
  // ...
}
```

Input

~~~
``` html
<ul>
  <li
    v-for="todo in todos"
    :key="todo.id"
  >
    {{ todo.text }}
  </li>
</ul>
```
~~~

Output

html

```html
<ul>
  <li
    v-for="todo in todos"
    :key="todo.id"
```

```
  >
    {{ todo.text }}
  </li>
</ul>
```

A list of valid languages is available on Prism's site.

Line Highlighting in Code Blocks

Input

~~~
``` js{4}
export default {
  data () {
    return {
      msg: 'Highlighted!'
    }
  }
}
```
~~~

Output

```js
export default {
  data () {
    return {
      msg: 'Highlighted!'
    }
  }
}
```

In addition to a single line, you can also specify multiple single lines, ranges, or both:

- Line ranges: for example `{5-8}`, `{3-10}`, `{10-17}`
- Multiple single lines: for example `{4,7,9}`
- Line ranges and single lines: for example `{4,7-13,16,23-27,40}`

Input

~~~
``` js{1,4,6-7}
export default { // Highlighted
  data () {
    return {
      msg: `Highlighted!
      This line isn't highlighted,
      but this and the next 2 are.`,
      motd: 'VuePress is awesome',
      lorem: 'ipsum',
    }
  }
}
```
~~~

Output

```js
export default { // Highlighted
  data () {
    return {
      msg: `Highlighted!
      This line isn't highlighted,
      but this and the next 2 are.`,
      motd: 'VuePress is awesome',
      lorem: 'ipsum',
    }
  }
}
```

Line Numbers

You can enable line numbers for each code block via config:

```js
module.exports = {
  markdown: {
    lineNumbers: true
  }
}
```

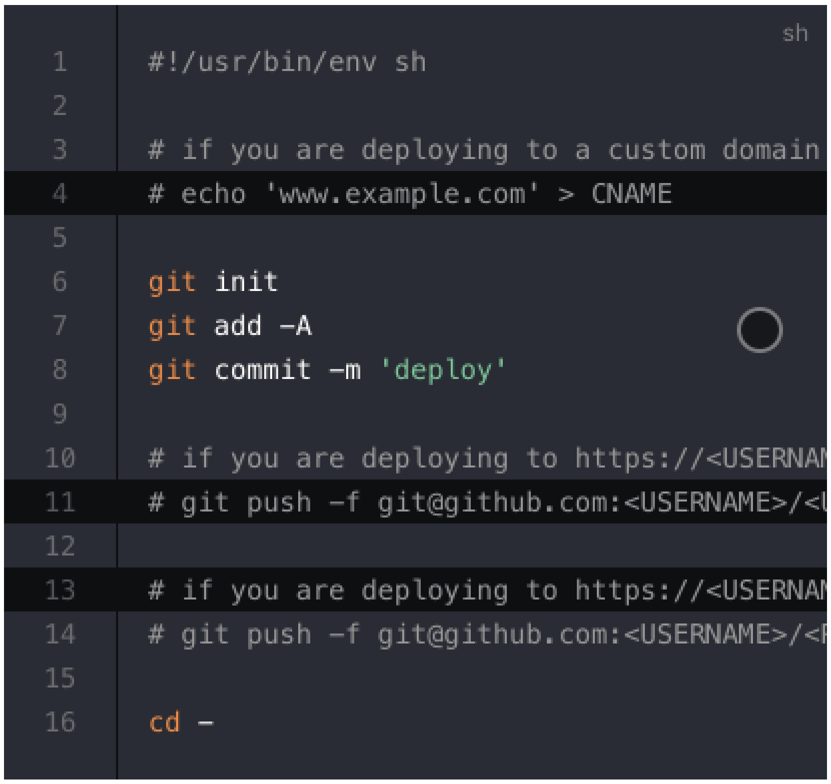
- Demo:

```sh
#!/usr/bin/env sh

# if you are deploying to a custom domain
# echo 'www.example.com' > CNAME

git init
git add -A
git commit -m 'deploy'

```



```sh
#!/usr/bin/env sh

# if you are deploying to a custom domain
# echo 'www.example.com' > CNAME

git init
git add -A
git commit -m 'deploy'

# if you are deploying to https://<USERNAM
# git push -f git@github.com:<USERNAME>/<U

# if you are deploying to https://<USERNAM
# git push -f git@github.com:<USERNAME>/<R

cd -
```

Import Code Snippets beta

You can import code snippets from existing files via following syntax:

```md
<<< @/filepath
```

It also supports line highlighting:

```md
<<< @/filepath{highlightLines}
```

Input

```md
<<< @/../@vuepress/markdown/__tests__/fragments/snippet.js{2}
```

Output

```js
export default function () {
  // ..
}
```

> **TIP**
>
> Since the import of the code snippets will be executed before webpack compilation, you can't use the path alias in webpack. The default value of `@` is `process.cwd()`.

You can also use a VS Code region to only include the corresponding part of the code file. You can provide a custom region name after a `#` following the filepath (`snippet` by default):

Input

```md
<<< @/../@vuepress/markdown/__tests__/fragments/snippet-with-region.js#snippet{1}
```

Code file

```js
// #region snippet
function foo () {
  return ({
    dest: '../../vuepress',
    locales: {
      '/': {
        lang: 'en-US',
        title: 'VuePress',
        description: 'Vue-powered Static Site Generator'
      },
      '/zh/': {
        lang: 'zh-CN',
        title: 'VuePress',
        description: 'Vue 驱动的静态网站生成器'
      }
    },
    head: [
      ['link', { rel: 'icon', href: `/logo.png` }],
      ['link', { rel: 'manifest', href: '/manifest.json' }],
      ['meta', { name: 'theme-color', content: '#3eaf7c' }],
      ['meta', { name: 'apple-mobile-web-app-capable', content: 'yes' }],
      ['meta', { name: 'apple-mobile-web-app-status-bar-style', content: 'black' }],
      ['link', { rel: 'apple-touch-icon', href: `/icons/apple-touch-icon-152x152.png` }],
      ['link', { rel: 'mask-icon', href: '/icons/safari-pinned-tab.svg', color: '#3eaf7c'
      ['meta', { name: 'msapplication-TileImage', content: '/icons/msapplication-icon-144
      ['meta', { name: 'msapplication-TileColor', content: '#000000' }]
    ]
  })
}
// #endregion snippet

export default foo
```

Output

```js
function foo () {
  return ({
    dest: '../../vuepress',
    locales: {
      '/': {
        lang: 'en-US',
        title: 'VuePress',
        description: 'Vue-powered Static Site Generator'
      },
      '/zh/': {
        lang: 'zh-CN',
        title: 'VuePress',
        description: 'Vue 驱动的静态网站生成器'
      }
    },
    head: [
      ['link', { rel: 'icon', href: `/logo.png` }],
      ['link', { rel: 'manifest', href: '/manifest.json' }],
      ['meta', { name: 'theme-color', content: '#3eaf7c' }],
      ['meta', { name: 'apple-mobile-web-app-capable', content: 'yes' }],
      ['meta', { name: 'apple-mobile-web-app-status-bar-style', content: 'black' }],
      ['link', { rel: 'apple-touch-icon', href: `/icons/apple-touch-icon-152x152.png` }],
      ['link', { rel: 'mask-icon', href: '/icons/safari-pinned-tab.svg', color: '#3eaf7c'
      ['meta', { name: 'msapplication-TileImage', content: '/icons/msapplication-icon-144
      ['meta', { name: 'msapplication-TileColor', content: '#000000' }]
    ]
  })
}
```

Advanced Configuration

VuePress uses markdown-it as the Markdown renderer. A lot of the extensions above are implemented via custom plugins. You can further customize the `markdown-it` instance using the `markdown` option in `.vuepress/config.js`:

```js
module.exports = {
  markdown: {
    // options for markdown-it-anchor
    anchor: { permalink: false },
    // options for markdown-it-toc
    toc: { includeLevel: [1, 2] },
    extendMarkdown: md => {
      // use more markdown-it plugins!
      md.use(require('markdown-it-xxx'))
    }
  }
}
```

Last Updated: 12/18/2021, 7:53:49 PM

Using Vue in Markdown

Browser API Access Restrictions

Because VuePress applications are server-rendered in Node.js when generating static builds, any Vue usage must conform to the universal code requirements . In short, make sure to only access Browser / DOM APIs in `beforeMount` or `mounted` hooks.

If you are using or demoing components that are not SSR-friendly (for example, contain custom directives), you can wrap them inside the built-in `<ClientOnly>` component:

```md
<ClientOnly>
  <NonSSRFriendlyComponent/>
</ClientOnly>
```

Note this does not fix components or libraries that access Browser APIs **on import**. To use code that assumes a browser environment on import, you need to dynamically import them in proper lifecycle hooks:

```vue
<script>
export default {
  mounted () {
    import('./lib-that-access-window-on-import').then(module => {
      // use code
    })
  }
}
</script>
```

If your module `export default` a Vue component, you can register it dynamically:

```vue
<template>
  <component v-if="dynamicComponent" :is="dynamicComponent"></component>
</template>
```

```
<script>
export default {
  data() {
    return {
      dynamicComponent: null
    }
  },

  mounted () {
    import('./lib-that-access-window-on-import').then(module => {
      this.dynamicComponent = module.default
    })
  }
}
</script>
```

Also see:

- Vue.js > Dynamic Components

Templating

Interpolation

Each Markdown file is first compiled into HTML and then passed on as a Vue component to `vue-loader`. This means you can use Vue-style interpolation in text:

Input

```md
{{ 1 + 1 }}
```

Output

```
2
```

Directives

Directives also work:

Input

```md
<span v-for="i in 3">{{ i }} </span>
```

Output

```
1 2 3
```

Access to Site & Page Data

The compiled component does not have any private data but does have access to the site metadata and computed properties. For example:

Input

```md
{{ $page }}
```

Output

```json
{
  "path": "/using-vue.html",
  "title": "Using Vue in Markdown",
  "frontmatter": {}
}
```

Escaping

By default, fenced code blocks are automatically wrapped with `v-pre`. To display raw mustaches or Vue-specific syntax inside inline code snippets or plain text, you need to wrap a paragraph with the `v-pre` custom container:

Input

```md
::: v-pre
`{{ This will be displayed as-is }}`
:::
```

Output

`{{ This will be displayed as-is }}`

Using Components

Any `*.vue` files found in `.vuepress/components` are automatically registered as global , async components. For example:

```
.
└─ .vuepress
   └─ components
      ├─ demo-1.vue
      ├─ OtherComponent.vue
      └─ Foo
         └─ Bar.vue
```

Inside any Markdown file you can then directly use the components (names are inferred from filenames):

```md
<demo-1/>
<OtherComponent/>
<Foo-Bar/>
```

Hello this is <demo-1>

This is another component

Hello this is <Foo-Bar>

IMPORTANT

Make sure a custom component's name either contains a hyphen or is in PascalCase. Otherwise it will be treated as an inline element and wrapped inside a `<p>` tag, which will lead to hydration mismatch because `<p>` does not allow block elements to be placed inside it.

Using Components In Headers

You can use Vue components in the headers, but note the difference between the following syntaxes:

Markdown	Output HTML	Parsed Header
`# text <Tag/>`	`<h1>text <Tag/></h1>`	`text`
``# text `<Tag/>` ``	`<h1>text <code>&lt;Tag/&gt;</code></h1>`	`text <Tag/>`

The HTML wrapped by `<code>` will be displayed as-is; only the HTML that is **not** wrapped will be parsed by Vue.

TIP

The output HTML is accomplished by markdown-it , while the parsed headers are handled by VuePress (and used for both the sidebar and document title).

Using Pre-processors

VuePress has built-in webpack support for the following pre-processors: `sass` , `scss` , `less` , `stylus` and `pug` . All you need to do is installing the corresponding dependencies. For example, to enable `sass` :

```sh
yarn add -D sass-loader node-sass
```

Now you can use the following in Markdown and theme components:

```vue
<style lang="sass">
.title
  font-size: 20px
</style>
```

Using `<template lang="pug">` requires installing `pug` and `pug-plain-loader`:

```sh
yarn add -D pug pug-plain-loader
```

> **TIP**
>
> If you are a Stylus user, you don't need to install `stylus` and `stylus-loader` in your project; VuePress uses Stylus internally.
>
> For pre-processors that do not have built-in webpack config support, you will need to extend the internal webpack config and install the necessary dependencies.

Script & Style Hoisting

Sometimes you may need to apply some JavaScript or CSS only to the current page. In those cases, you can directly write root-level `<script>` or `<style>` blocks in the Markdown file. These will be hoisted out of the compiled HTML and used as the `<script>` and `<style>` blocks for the resulting Vue single-file component:

This is rendered by inline script and styled by inline CSS

Built-In Components

OutboundLink stable

The indicator ↗ is used to denote external links. In VuePress, this component has been followed by every external link.

ClientOnly stable

See Browser API Access Restrictions.

Content

- **Props**:
  - `pageKey` - string, page's hash key, defaults to current page's key.
  - `slotKey` - string, key of Markdown slot. Defaults to default slot.
- **Usage**：

Specify a specific slot for a specific page (.md) for rendering. This is useful when using a Custom Layout or Writing a theme:

```vue
<Content/>
```

Also see:

- Global Computed > $page
- Markdown Slot
- Writing a theme > Content Outlet

Badge beta default theme

- **Props**:
  - `text` - string
  - `type` - string, optional value: `"tip"|"warning"|"error"`, defaults to `"tip"`.
  - `vertical` - string, optional value: `"top"|"middle"`, defaults to `"top"`.
- **Usage**:

You can use this component in a header to add some status for an API:

```md
### Badge <Badge text="beta" type="warning"/> <Badge text="default theme"/>
```

Also see:

- Using Components In Headers

Last Updated: 11/10/2020, 3:42:04 PM

Internationalization

Site Level i18n Config

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

```
docs
├─ README.md
├─ foo.md
├─ nested
│  └─ README.md
└─ zh
   ├─ README.md
   ├─ foo.md
   └─ nested
      └─ README.md
```

Then, specify the `locales` option in `.vuepress/config.js`:

```js
module.exports = {
  locales: {
    // The key is the path for the locale to be nested under.
    // As a special case, the default locale can use '/' as its path.
    '/': {
      lang: 'en-US', // this will be set as the lang attribute on <html>
      title: 'VuePress',
      description: 'Vue-powered Static Site Generator'
    },
    '/zh/': {
      lang: 'zh-CN',
      title: 'VuePress',
      description: 'Vue 驱动的静态网站生成器'
    }
  }
}
```

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

Default Theme i18n Config

The default theme also has built-in i18n support via `themeConfig.locales`, using the same `{ path: config }` format. Each locale can have its own nav and sidebar config, along with some other text values used across the site:

```js
module.exports = {
  locales: { /* ... */ },
  themeConfig: {
    locales: {
      '/': {
        // text for the language dropdown
        selectText: 'Languages',
        // label for this locale in the language dropdown
        label: 'English',
        // Aria Label for locale in the dropdown
        ariaLabel: 'Languages',
        // text for the edit-on-github link
        editLinkText: 'Edit this page on GitHub',
        // config for Service Worker
        serviceWorker: {
          updatePopup: {
            message: "New content is available.",
            buttonText: "Refresh"
          }
        },
        // algolia docsearch options for current locale
        algolia: {},
        nav: [
          { text: 'Nested', link: '/nested/' , ariaLabel: 'Nested' }
        ],
        sidebar: {
          '/': [/* ... */],
          '/nested/': [/* ... */]
        }
      },
      '/zh/': {
        selectText: '选择语言',
        label: '简体中文',
        editLinkText: '在 GitHub 上编辑此页',
        serviceWorker: {
          updatePopup: {
            message: "发现新内容可用.",
            buttonText: "刷新"
          }
        },
        nav: [
          { text: '嵌套', link: '/zh/nested/' }
        ],
        algolia: {},
        sidebar: {
          '/zh/': [/* ... */],
          '/zh/nested/': [/* ... */]
        }
      }
    }
  }
}
```

Last Updated: 7/19/2020, 5:50:19 AM

Deploying

The following guides are based on some shared assumptions:

- You are placing your docs inside the `docs` directory of your project;
- You are using the default build output location (`.vuepress/dist`);
- VuePress is installed as a local dependency in your project, and you have setup the following npm scripts:

```json
{
  "scripts": {
    "docs:build": "vuepress build docs"
  }
}
```

GitHub Pages

1. Set the correct `base` in `docs/.vuepress/config.js`.

   If you are deploying to `https://<USERNAME>.github.io/`, you can omit `base` as it 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. Inside your project, create `deploy.sh` with the following content (with highlighted lines uncommented appropriately), and run it to deploy:

```sh
#!/usr/bin/env sh

# abort on errors
set -e

# build
npm run docs:build
```

```
# navigate into the build output directory
cd docs/.vuepress/dist

# if you are deploying to a custom domain
# echo 'www.example.com' > CNAME

git init
git add -A
git commit -m 'deploy'

# if you are deploying to https://<USERNAME>.github.io
# git push -f git@github.com:<USERNAME>/<USERNAME>.github.io.git master

# if you are deploying to https://<USERNAME>.github.io/<REPO>
# git push -f git@github.com:<USERNAME>/<REPO>.git master:gh-pages

cd -
```

TIP

You can also run the above script in your CI setup to enable automatic deployment on each push.

TIP

When you use a **Custom Domain name**, you MUST add the CNAME file into /docs/.vuepress/public folder (Create the folder if it isn't there). Otherwise, your CNAME file will always be removed with each deploy and never work.

GitHub Pages and GitHub Actions

1. Create a personal access token ;
2. Create encrypted secrets under your repository;
3. Create a `.yml` or `.yaml` file in the `.github/workflows` directory in the root of your repository. e.g. `vuepress-deploy.yml` :

```yaml
name: Build and Deploy
on: [push]
jobs:
  build-and-deploy:
    runs-on: ubuntu-latest
    steps:
    - name: Checkout
      uses: actions/checkout@master

    - name: vuepress-deploy
      uses: jenkey2011/vuepress-deploy@master
      env:
        ACCESS_TOKEN: ${{ secrets.ACCESS_TOKEN }}
        TARGET_REPO: username/repo
        TARGET_BRANCH: master
        BUILD_SCRIPT: yarn && yarn build
        BUILD_DIR: docs/.vuepress/dist
        CNAME: https://www.xxx.com
```

For more information, you can checkout jenkey2011/vuepress-deploy .

GitHub Pages and Travis CI

1. Set the correct `base` in `docs/.vuepress/config.js` .

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

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

2. Create a file named `.travis.yml` in the root of your project.

3. Run `yarn` or `npm install` locally and commit the generated lockfile (that is `yarn.lock` or `package-lock.json`).

4. Use the GitHub Pages deploy provider template, and follow the Travis CI documentation .

```yaml
language: node_js
node_js:
  - lts/*
```

```
install:
  - yarn install # npm ci
script:
  - yarn docs:build # npm run docs:build
deploy:
  provider: pages
  skip_cleanup: true
  local_dir: docs/.vuepress/dist
  github_token: $GITHUB_TOKEN # A token generated on GitHub allowing Travis to push code
  keep_history: true
  on:
    branch: master
```

GitLab Pages and GitLab CI

1. Set the correct `base` in `docs/.vuepress/config.js`.

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

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

2. Set `dest` in `.vuepress/config.js` to `public`.

3. Create a file called `.gitlab-ci.yml` in the root of your project with the content below. This will build and deploy your site whenever you make changes to your content:

```yaml
image: node:10.22.0
pages:
  cache:
    paths:
    - node_modules/

  script:
  - yarn install # npm install
  - yarn docs:build # npm run docs:build
  artifacts:
    paths:
    - public
  only:
  - master
```

Netlify

1. On Netlify, setup up a new project from GitHub with the following settings:

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

2. Hit the deploy button.

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`:

```json
{
 "hosting": {
   "public": "./docs/.vuepress/dist",
   "ignore": []
 }
}
```

`.firebaserc`:

```js
{
 "projects": {
   "default": "<YOUR_FIREBASE_ID>"
 }
}
```

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

Surge

1. First install surge , if you haven't already.
2. Run `yarn docs:build` or `npm run docs:build` .
3. Deploy to surge by typing `surge docs/.vuepress/dist` .

You can also deploy to a custom domain by adding `surge docs/.vuepress/dist yourdomain.com` .

Heroku

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

   ```sh
   heroku login
   ```

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

   `static.json` :

   ```json
   {
     "root": "./docs/.vuepress/dist"
   }
   ```

This is the configuration of your site; read more at heroku-buildpack-static .

5. Set up your Heroku git remote:

```sh
# version change
git init
git add .
```

```
git commit -m "My site ready for deployment."

# creates a new app with a specified name
heroku apps:create example

# set buildpack for static sites
heroku buildpacks:set https://github.com/heroku/heroku-buildpack-static.git
```

6. Deploy your site:

```sh
# publish site
git push heroku master

# opens a browser to view the Dashboard version of Heroku CI
heroku open
```

Vercel

See Creating and Deploying a VuePress App with Vercel .

Layer0

See Creating and Deploying a VuePress App with Layer0 .

Last Updated: 12/23/2021, 8:32:09 PM

Frontmatter

Any Markdown file that contains a YAML frontmatter block will be processed by gray-matter .
The frontmatter must be at the top of the Markdown file, and must take the form of valid YAML set between triple-dashed lines. Example:

```md
---
title: Blogging with VuePress
lang: en-US
---
```

Between the triple-dashed lines, you can set predefined variables, or even create custom ones of your own. These variables can be used via the `$frontmatter` variable.

Here's an example of how you could use it in your Markdown file:

```md
---
title: Blogging with VuePress
lang: en-US
---

# {{ $frontmatter.title }}

My blog post is written in {{ $frontmatter.lang }}.
```

Alternative frontmatter Formats

VuePress also supports JSON and TOML frontmatter syntax.

JSON frontmatter needs to start and end in curly braces:

```
---
{
  "title": "Blogging Like a Hacker",
  "lang": "en-US"
}
---
```

TOML frontmatter needs to be explicitly marked as TOML:

```
---toml
title = "Blogging Like a Hacker"
lang = "en-US"
---
```

Predefined Variables

title

- Type: `string`
- Default: `h1_title || siteConfig.title`

Title of the current page.

lang

- Type: `string`
- Default: `en-US`

Language of the current page.

description

- Type: `string`
- Default: `siteConfig.description`

Description of the current page.

layout

- Type: `string`

- Default: `Layout`

Set the layout component of the current page.

permalink

- Type: `string`
- Default: `siteConfig.permalink`

See Permalinks for details.

metaTitle

- Type: `string`
- Default: ``` `${page.title} | ${siteConfig.title}` ```

Override the default meta title.

meta

- Type: `array`
- Default: `undefined`

Specify extra meta tags to be injected:

```yaml
---
meta:
  - name: description
    content: hello
  - name: keywords
    content: super duper SEO
---
```

canonicalUrl 1.7.1+

- Type: `string`
- Default: `undefined`

Set the canonical URL for the current page.

Predefined Variables Powered By Default Theme

navbar

- Type: `boolean`
- Default: `undefined`

See Default Theme Config > Disable the Navbar for details.

sidebar

- Type: `boolean|'auto'`
- Default: `undefined`

See Default Theme Config > Sidebar for details.

prev

- Type: `boolean|string`
- Default: `undefined`

See Default Theme Config > Prev / Next Links for details.

next

- Type: `boolean|string`
- Default: `undefined`

See Default Theme Config > Prev / Next Links for details.

search

- Type: `boolean`
- Default: `undefined`

See Default Theme Config > Built-in Search for details.

tags

- Type: `array`
- Default: `undefined`

See Default Theme Config > Built-in Search. For details.

Last Updated: 12/18/2021, 7:16:12 PM

Permalinks

Background

Prior to VuePress version 1.0.0, VuePress retrieved all Markdown files in the documents source directory and defined the page links based on the file hierarchy. For example, if you had the following file and directory structure:

```
├── package.json
└── source
    ├── _post
    │   └── intro-vuepress.md
    ├── index.md
    └── tags.md
```

You would then get the following pages:

```
/source/
/source/tags.html
/source/_post/intro-vuepress.html
```

But for blogs, a customized link of a post would be highly preferrable. VuePress version 1.0.0 introduced support for this feature, known as a **permalink**. With version 1.0.0 or newer, you instead get the following pages:

```
/source/
/source/tags/
/source/2018/4/1/intro-vuepress.html
```

This describes the beginning of how VuePress can be used for a blog!

Permalinks

A permalink is a URL that is intended to remain unchanged for a long time, yielding a hyperlink that is less susceptible to what is known as link rot . VuePress supports a flexible way to build permalinks, allowing you to use template variables.

The default permalink is `/:regular` .

Configure Permalinks

You can enable permalinks globally for all pages:

```js
// .vuepress/config.js
module.exports = {
  permalink: '/:year/:month/:day/:slug'
}
```

You can also set set a permalink for a single page only. This overrides the aforementioned global setting:

hello.md:

```md
---
title: Hello World
permalink: /hello-world
---

Hello!
```

Template Variables

Variable	Description
:year	Published year of post (4-digit)
:month	Published month of post (2-digit)
:i_month	Published month of post (without leading zeros)
:day	Published day of post (2-digit)

Variable	Description
:i_day	Published day of post (without leading zeros)
:slug	Slugified file path (without extension)
:regular	Permalink generated by VuePress by default. See fileToPath.ts for details

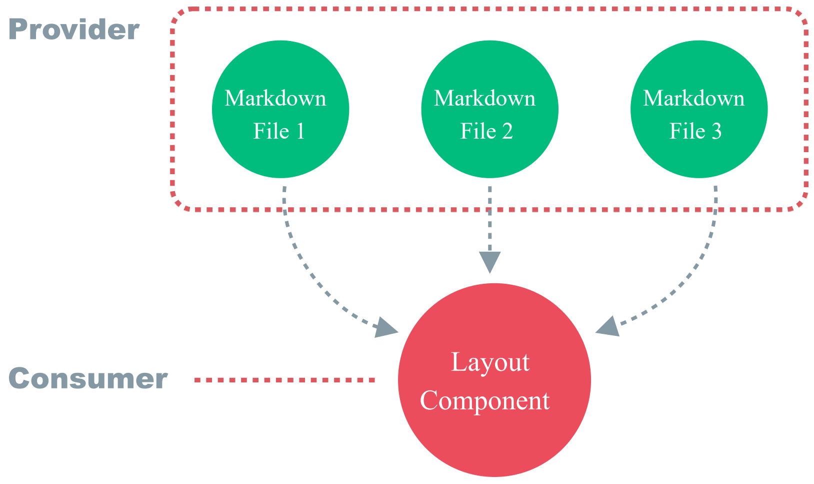
Last Updated: 7/23/2020, 7:11:42 AM

Markdown Slot

VuePress implements a content distribution API for Markdown. With this feature, you can split your document into fragments, allowing flexible composition in the layout component.

Why do I need Markdown Slot?

First, let's review the relationship between layout components and Markdown files:



Markdown files are providers of metadata (page content, configuration, etc.), while layout components consume them. We can use `frontmatter` to define some metadata for common data types, but `frontmatter` is hard to do something about Markdown / HTML, a complex metadata that involves differences before and after compilation.

Markdown Slots solve this problem.

Named Slots

You can define a named Markdown slot through the following Markdown syntax:

```md
::: slot name
```

```
:::
```

Use the `Content` component to use the slot in the layout component:

```vue
<Content slot-key="name"/>
```

> **TIP**
>
> Here we are using `slot-key` instead of `slot`, because in Vue, `slot` is a reserved prop name.

Default Slot Content

By default, the slot-free part of a Markdown file becomes the default content of a Markdown slot. You can access this directly using the `Content` component:

```vue
<Content/>
```

Example

Suppose your layout component is as follows:

```vue
<template>
  <div class="container">
    <header>
      <Content slot-key="header"/>
    </header>
    <main>
      <Content/>
    </main>
    <footer>
      <Content slot-key="footer"/>
    </footer>
  </div>
</template>
```

If the Markdown content of a page is this:

```md
::: slot header
# Here might be a page title
:::

- A Paragraph
- Another Paragraph

::: slot footer
Here's some contact info
:::
```

Then the rendered HTML of this page will be:

```html
<div class="container">
  <header>
    <div class="content header">
      <h1>Here might be a page title</h1>
    </div>
  </header>
  <main>
    <div class="content default">
      <ul>
        <li>A Paragraph</li>
        <li>Another Paragraph</li>
      </ul>
    </div>
  </main>
  <footer>
    <div class="content footer">
      <p>Here's some contact info</p>
    </div>
  </footer>
</div>
```

Note that:

1. Unlike the slot mechanism provided by Vue itself, each content distribution is wrapped in a `div` whose class is `content` with the name of the slot.
2. You need to ensure the uniqueness of the slot defined.

Last Updated: 7/19/2020, 5:50:19 AM

Global Computed

In VuePress, some core computed properties can be used by the default theme or custom themes. Or in Markdown pages using vue.

$site

This is the `$site` value of the site you're currently reading:

```json
{
  "title": "VuePress",
  "description": "Vue-powered static site generator",
  "base": "/",
  "pages": [
    {
      "lastUpdated": 1524027677000,
      "path": "/",
      "title": "VuePress",
      "frontmatter": {}
    },
    ...
  ]
}
```

$page

This is the `$page` value of the page you're currently reading:

```json
{
  "title": "Global Computed",
  "frontmatter": {},
  "regularPath": "/guide/global-computed.html",
  "key": "v-d4cbeb69eff3d",
  "path": "/guide/global-computed.html",
  "headers": [
    {
      "level": 2,
      "title": "$site",
      "slug": "site"
    },
    {
      "level": 2,
      "title": "$page",
      "slug": "$page"
    },
    ...
  ]
}
```

$frontmatter

Reference of $page.frontmatter.

$lang

The language of the current page. Default: `en-US`.

For more information, see Internationalization.

$localePath

The locale path prefix for the current page. Default: `/`.

For more information, see Internationalization.

$title

Value of the `<title>` label used for the current page.

$description

The `content` value of the `<meta name= "description" content= "...">` for the current page.

$themeConfig

Refers to `siteConfig.themeConfig`.

Last Updated: 11/10/2020, 3:42:04 PM