



# VuePress

Welcome to your VuePress site

[Get Started →](#)

---

---

## Simplicity First

Minimal setup with markdown-centered project structure helps you focus on writing.

## Vue-Powered

Enjoy the dev experience of Vue + webpack, use Vue components in markdown, and develop custom themes with Vue.

## Performant

VuePress generates pre-rendered static HTML for each page, and runs as an SPA once a page is loaded.



```
# install
yarn global add vuepress@next
# OR npm install -g vuepress@next

# create a markdown file
echo '# Hello VuePress' > README.md

# start writing
vuepress dev

# build to static files
vuepress build
```

## COMPATIBILITY NOTE

VuePress requires Node.js  $\geq 8$ .

# 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 containing custom directives), you can wrap them inside the built-in `<ClientOnly>` component:

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

md

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

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

vue

## 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



## Output

2

## Directives

Directives also work:

## Input

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

md

## 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**. For example:

## Input

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

md

## Output

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

json

## Escaping

---



paragraph with the `v-pre` custom container:

### Input

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

md

### Output

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

## Using Components

Any `*.vue` files found in `.vuepress/components` are automatically registered as **global**<sup>[↗](#)</sup>, **async**<sup>[↗](#)</sup> 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):

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

md

### 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 two ways:

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

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 done by VuePress, and used for the [sidebar](#) and the document title.

## Using Pre-processors

VuePress has built-in webpack config 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` , install the following in your project:

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

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

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

```
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 because 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** in addition to installing 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, and they 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

It([↗](#)) is used to indicate that this is an external link. In VuePress this component have been followed by every external link.

### ClientOnly stable

See **Browser API Access Restrictions**.

### Content beta

- Props:

- `custom` - boolean

- Usage:



`<Content/>`

vue

Also see:

- [Custom Themes > Content Outlet](#)

## Badge beta 0.10.1+

- **Props:**

- `text` - string
- `type` - string, optional value: `"tip"|"warn"|"error"` , defaults to `"tip"` .
- `vertical` - string, optional value: `"top"|"middle"` , defaults to `"top"` .

- **Usage:**

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

```
### Badge <Badge text="beta" type="warn"/> <Badge text="0.10.1+"/>
```

md

Also see:

- [Using Components In Headers](#)