Docs as code: The technical side

Docs as code is an interesting way to create documentation for software and technical projects. The idea is that we use the same tools to create our documentation that engineers or technical authors use to create their projects.

According to [Write the Docs](Docs As Code):

Documentation as Code (Docs as Code) refers to a philosophy that you should be writing documentation with the same tools as code:

- Issue Trackers
- Version Control (Git)
- Plain Text Markup (Markdown, reStructuredText, Asciidoc)
- Code Reviews
- Automated Tests

This means the technical writing process follows the same workflows as development teams, and enables a culture where writers and developers both feel ownership of documentation and can work together in its development.

If we are treating documentation or any content writing as part of the application code we need a way to build our documentation.

What we need to do if we take this approach

Docs as code is as much a philosophical change as much as a technical one.

Some of the things we need to do to make docs as code work:

- Convince people that writing docs in something like Markdown or Asciidoc is worth it
- If needed, provide Markdown extensions to accomplish project specific tasks
- Provide a strong review and editing facility, something similar to what we can do in Google Docs but using the hosting platform features

- Provide a good publishing platform as part of the documentation toolchain
- Automate updates and publications of documentation updates, possibly with continuous integration or Github actions or equivalent functionality

This post will cover some of the technical aspects of docs like code. I can't, and won't, tell you how to sell the idea to your team.

Why Markdown

Markdown provides a light weight, text-based, means to write content that is easy to ready by humans and for computer programs to process and covert to HTML and other programs.

> The overriding design goal for Markdown's formatting syntax is to make it as readable as possible. The idea is that a Markdown-formatted document should be publishable as-is, as plain text, without looking like it's been marked up with tags or formatting instructions.
>
> — http://daringfireball.net/projects/markdown/

Having the files in a text-based format makes it easier to include in code repositories.

when it was first created, there was no specification of what Markdown should do or specific behavior, there was only a syntax page. In 2014, people using Markdown in large-scale deployments create Commonmark and its specification as a unified syntax that all Markdown implementations can reference and follow.

RFC 7764 provides an overview of the format and the tools to create it.

Perhaps the best feature is its extensivility. There are many available extensions and you can write your own if none of the existing ones do what you need to.

Publishing the docs

We'll publish our docs in two ways:

- Publish individual documents by converting Markdown files to HTML

- Build a website converting the Markdown files to Vue using Vuepress

Direct Markdown to HTML conversion with Gulp

In the past I've relied on Gulp to convert Markdown files into HTML, and put them inside a template file using Gulp.js.

For this example, we'll assume there is an existing project that uses NPM, so we'll start with installing the packages we need.

The current version of Gulp is 4.0.2. If you need to stay in version 3 the istructions and commands may be slightly different.

First we install gulp globally so we can just run `gulp` from the command line.

Then we install Gulp (again), gulp-remarkable and gulp-wrap as development dependencies for our current project.

```
npm i -g gulp-cli
npm i -D gulp gulp-remarkable gulp-wrap
```

Next we create a `gulpfile.js` file to put all out Gulp related code in.

The first step is to register the modules we want to use using commonjs's `require`.

```
const gulp = require('gulp');
const markdown = require('gulp-remarkable');
const wrap = require('gulp-wrap');
```

We then define two gulp tasks; one to convert markdown files into HTML (`markdown`) and another one wrap a container around the HTML files we just created (`build-template`).

```
gulp.task('markdown', () => {
  return gulp.src('src/md-content/*.md'src/md-content/*.md'wn({
```

```
        preset: 'commonmark',
        remarkableOptions: {
          typographer: true,
          linkify'commonmark'      breaks: false,
        },
      }))
      .pipe(gulp.dest('src/html-content/'));
});

gulp.task('build-template', gulp.series('markdown'), () => {
  gulp.src('./src/html-content'src/html-content/'/*.html')
      .pipe(wrap({
        src: './src/templates/template.html',
      }))
      .pipe(gulp.dest('./src/'));
});
```

This process works for converting individual files to HTML and PDF but it doesn't work when converting documentation into a website that people can use presents additional challenges and may require a different approach to building the content.

A future update to the code will include using plugins to add functionality to Remarkable without resorting to writing HTML in the Markdown files. These plugins make writing easier but require more work in the back end and a higher cognitive load in having to remember the chracter combination for each.

Using a static site generator: Vuepress

Vuepress is a static site generator using Vue.js as the front end framework. I picked it for two reasons:

- From research it appears that Vuepress gives a lot of functionality out of the box and easy ways to extend it if we need additional tools
- Vuepress was designed for the Vue.js documentation site so, I'm guessing, it'll work for other documentation sites as well

For this section, I will use Vuepress manual installation process as documented in the Vuepress Guide.

1. Create and change into a new directory
2. Initialize with your preferred package manager
3. Install VuePress locally
4. Add documents to the `docs` directory of your site
5. Add Vuepress-specific scripts to package.json
6. Serve the documentation site in the local server

Create and change to a new directory

Before we can create and configure the project we need to setup the directory where we want to put the docs. In Bash (macOS and Linux) run the following command:

```
mkdir project-docs && cd project-docs
```

Initialize with your preferred package manager

Vuepress is a Node application. To preserve information about the project, we need to create a `project.json`. We will use npm.

```
npm init
```

Even if the docs site will be part of a monorepo I still prefer configuring and building the docs as a standalone project.

Install VuePress locally

```
npm install -D vuepress
```

Add documents to the `docs` directory in the directory you just created

Vuepress works from the `docs` directory so we need to have some content before we run Vuepress.

> The first command below will create the docs director
>
> The following command will add the content we echo to the README.md file in

the docs directory.

```
mkdir docs
echo '# Demo Content' > docs/README.md
```

Add Vuepress-specific scripts to package.json

You can run Vuepress directly from the command line but you can also add the commands as NPM scripts so you don't have to remember the exact command.

Add the following entries to the `scripts` section of your `package.json`

`docs:dev` will compile the content and run WebPack's dev server with hot module reloading

`docs:build` will build a production version of the site, ready to be published wherever it will be hosted.

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

Serve the documentation site in the local server

Now we're ready to run the development version of the code. Run the following command in your terminal

```
npm run docs:dev
```

If you now go to `localhost:8080` you will see the content of the `README.md` file in your docs directory.

This is the basic configuration for Vuepress. There's more work to do in configuring the default theme with, at least, the following items:

- Site title
- Top-right navigation bar

- Sidebar navigation

We will document these items in a future post.

Pushing docs to version control and serving them from there

Because we're treating our documentation as code we can take advantage of services built for code to handle our documentation projects.

If we use Gulp to convert Markdown to HTML, we can publish the content directly to Github Pages.

If you're using Vuepress, their deployment guide gives instructions for deploying to multiple open source platforms.

Whatever hosting mechanism we choose, we now have a bare-bones working set of project documents and the site to host them on.

Links and Resources

- Docs as code philosophy
    - Docs As Code
- Books on Docs Like Code
    - Docs Like Code — Anne Gentle
    - Modern Technical Writing: An Introduction to Software Documentation — Andrew Etter
- Static Site Generators
    - Hugo
    - Vuepress
        - Vuepress Docs
    - Gatbsy
    - Static Site Generators: Nunjucks and Gulp
- Gulp-based research
    - gulp-documentation
    - gulp-pandoc
    - Generating HTML and PDF from Markdown