AJYL Docstack

AsciiDoc	**J**AMstack
YAML	**L**iquid

An Open Platform for the Toughest Documentation Challenges

AJYL is a DocOps platform solution for highly complex documentation projects, solving sophisticated sourcing and output problems with a lightweight ecosystem suitable for nearly any kind of technical writing and publishing. AJYL answers the question, “What is the most capable set of technologies for implementing the docs-as-code approach to documenting and instructing software products?” No more wandering in the wilderness; you do not have to figure it out yourself.

The Components

A solution stack is only as good as its underlying technologies, so let’s explore.

AsciiDoc (via Asciidoctor)

The resurgent AsciiDoc markup format enables writing in elegant code (like Markdown, only prettier and with semantic features), along with limited programmatic powers, such as variable substitution, conditionals, and file inclusion. Asciidoctor is a suite of highly extensible tools with a growing community of developers and users, bringing the power of AsciiDoc to modern developer and publisher toolchains.

JAMstack/Jekyll

Delivering flexible docs requires much more than markup and a traditional documentation generator. You want venerable markup formats that have passed the test of time, but when it comes to functional tooling, you want the latest and greatest. Tools for managing, generating, deploying, and servicing (search, authentication, etc) your docs should adhere to the established mix of **J**avaScript, **A**PIs, and **m**arkup. Docs-as-code requires a static-site generator, and the SSG platform with the most users, contributors, and extensions happens to fit perfectly into the suite (and weirdly into the acronym, as well). **Jekyll** builds documents from YAML and AsciiDoc, and it uses Liquid as its native template engine. The *J* in *AJYL* could just as easily be a *G* for static site **g**enerator, but nearly all SSGs are JAMstack utilities, and JAMstack is so much more, including solutions for source management and continuous deployment.

YAML

Documentation is much more than written content. AJYL brings “small data” to docs: storing information in structured flat files formatted as human-readable YAML, for dynamic formatting and insertion into the docs at build time. This provides much of the content-management power of databases, all taking full advantage source control and feeding the Asciidoctor and Jekyll rendering

engines. YAML is arguably the cleanest and lightest of the semi-structured data formats, like JSON with no extraneous markup.

Liquid

A good templating format and engine is the unsung hero of any fully customizable content system. For this we choose Liquid, because it is delightfully extensible and widely used by major commercial and open-source platforms, including Jekyll. Liquid empowers us to prebuild content like lists, tables, and references from our YAML small-data, as well as generate site-navigation elements and perform all manner of other text-transformation challenges.

AJYL: Greater than the Sum of its Parts

The power and potential of this last component, the Liquid templating format/engine, would be difficult to overstate. While YAML and AsciiDoctor provide reliable storage formats for static data and content, respectively, they are frustratingly limited when it comes to truly dynamic or programmatic capabilities, especially when it comes to variable text substitution, conditionals, and iterative looping.

Liquid comes to the rescue, closing these gaps with elegantly filtered text transformations, logical conditional flows, and intricately configurable looping over data. Once you are able to identify challenges that are solvable with Liquid, you will see why AJYL is flexible enough to solve nearly any documentation problem you can dream up.

Tying all of these technologies together is an additional utility and a burgeoning set of conventions for implementing AJYL in production docs across numerous applications.

LiquiDoc CMF: AJYL's Flagship Application

LiquiDoc is a build tool designed to make pulling the four component technologies of AJYL together. It provides a command-line interface and a Ruby Gem so you can integrate it into your product build system or deploy it independently.

LiquiDoc performs routines scripted in Liquid-enhanced YAML. During a preconfigured build procedure, LiquiDoc

- copies digital assets into place as needed;
- prebuilds new source files from small data;
- renders HTML sites using Jekyll and Asciidoctor; and
- renders PDF documents using Asciidoctor and Prawn.

LiquiDoc CMF (LDCMF) is the content-management *framework* designed to standardize complex documentation projects in order to better instruct and coordinate AJYL projects. LiquiDoc—the utility—can be used without this framework. It excells at dynamically reformatting source matter using templates, and also at ingesting lots of external data into Asciidoctor operations. Meanwhile, LDCMF—the framework—is designed precisely to take full advantage of AJYL and associated

technologies.

Alternatives to LiquiDoc CMF

Still a highly experimental and nacent framework we expect to (im)prove over time, we recognize there may be better frameworks out there. If you're using these four technologies together with other tooling, or even using LiquiDoc differently, we want to know about it and point people your way.

For now, even in their infancy, AJYL and LDCMF form a reliable wheel you do not need to reinvent. The platform can solve a huge range of complex docs challenges with a minimal array of lightweight technologies to learn, all of them completely free and open source.

A Note About Ruby

While the Ruby programming language is central all of AJYL's tooling, *you do not have to like or even know any Ruby programming* to take terrific advantage of AJYL and LDCMF. Asciidoctor, Jekyll, Liquid, and LiquiDoc are all sourced in Ruby, and YAML is very heavily associated with these and many other Ruby projects. This is all advantageous to the stack and anyone looking to develop more tooling for it, but most AJYL users will never touch the underlying Ruby source.

However, you will have to have the **Ruby runtime environment** installed on every machine that needs to build your docs. This includes every documentation conributor's personal workstation as well as integration and deployment servers.

Ruby plays extremely nicely with Linux and MacOS, but Windows users will find it frustrating at first. Since Windows is dominant in the technical writing profession, we are hard at work to bridge this gap. We believe the extra effort is worthwhile to escape the trap of proprietary tools.

The good news is *Ruby makes installing and maintaining the entire AJYL toolchain trivial* once it is configured. Because Ruby provides a cross-platform runtime enironment, builds will work the same across operating systems.

Friends of AJYL

No platform does *everything*, and wise developers know better than to expect the impossible. Do what you do best, and for everything else: extension and integration, which is why *JAMstack* is included in *AJYL* in the first place. Just to show AJYL has you covered for all your docs needs, here are our answers to how to solve various elements of a complete docs-as-code toolset.

- **Source control**
  Spolier alert! You're going to want to use Git unless you have a damn good reason (and already know it).
- **Continuous deployment**
  Since AJYL is strictly docs-as-code using open-source tools, it will work with pretty much any build/deploy system. If you have one you love, use it. If you don't know where to start, we can help.

- **Search**
  Cloud search providers are the state of the art here, though solutions range from self-contained JavaScript to cloud-hosted SaaS options all the way to highly customizable self-hosted search platforms.
- **Content Management**
  Perhaps the weakest aspect of the docs-as-code approach, including AJYL, is the the tooling for creation, editing, and coordination of dynamic content. It is also where most established proprietary solutions fall down.

Undisclaimer

The maintainers of this document are in no way rewarded or reimbursed for endorsements or recommendations made herein. All opinions are freely given by people who truly hope every project discussed below succeeds beyond its contributors' wildest dreams.

Alternatives and Extensions

While we believe the four core components of AJYL are the ideal combination of technologies, there are of course suitable alternatives to each. Moreover, numerous complementary platforms round out a complete documentation toolchain. What follows is a list of badass projects — themselves either fully open source or extremely FOSS-friendly — which we recommend instead of or in addition to AJYL's core technologies.

Alternatives to AsciiDoc for Content Source Markup

We love and believe in the growing AsciiDoc community, but you may have good reasons to try another markup format. The two most popular alternatives to AsciiDoc are Markdown and reStructuredText. Markdown is not a proper technical documentation language, as it lacks dynamic and semantic features, but shops that are already using it heavily in other arenas (in API docs, on GitHub, etc) may have cause to use it for their core docs source as well. If you are not already dependent on Markdown, for goodness sake do not get started now.

ReStructuredText is reputedly as powerful as AsciiDoc, and should be a very serious contender if the product you're documenting is largely sourced in Python.

Truthfully, the range of legitimate lightweight markup formats is at least as big as those supported by the excellent text-conversion application Pandoc. Pandoc handles Markdown (including CommonMark and GitHub-flavored Markdown), reStructuredText, AsciiDoc, Emacs Org-Mode, Emacs Muse, Textile, txt2tags, MediaWiki markup, DokuWiki markup, TikiWiki markup, TWiki markup, Vimwiki markup, and ZimWiki markup.

Alternatives to YAML for Small Data

A modern docs-as-code platform must make use of a data source that is accessible to the product developers and documentarians alike. This means open-source tooling and lightweight interfaces: not exactly the domain of XML- or SQL-based systems. JSON is not a terrible option, but it is unnecessarily complicated. YAML makes sense for us because it is in widespread use, including in

several of the components core or akin to AJYL: Jekyll, asciidoctor-pdf, and LiquiDoc.

The suitable options include any of the various structured and semi-structured serialization formats — anything that works in a flat file (and is therefore Git-managable).

Alternatives to Liquid for Templating

This is perhaps the richest category. Templating formats are a dime a dozen, and many of them are excellent. This Wikipedia master list is perhaps the most-definitive list. Those formats which work with Tilt might be the best place to find a Liquid alternative. If your chosen static-site generator is not Jekyll, it may require a different templating for its layout control. For instance, Hugo uses the Go language's native templating libraries, which can be deceptively and frustratingly similar to Liquid. Therefore, if a particular format/engine is necessitated by your SSG, that might be a better choice for all your content templating (prebuilding), as well. (LiquiDoc can be extended to accommodate any templating language handled by Tilt.)

GitHub for Source Collaboration Platform

If you have a favorite solution to this category, or if you use a non-Git source-control platform, you probably know what you're doing. For the rest, GitHub is a slight preference over GitLab, which are also doing great stuff. Most of our code and examples will be in GitHub, but both integrate somewhat nicely with AsciiDoc, so there's no killer reason GH is better. We strongly hope GitLab and all the others succeed.

Netlify for Continuous Deployment

The best way to integrate AJYL is to hook it into your continuous-deployment operations. You'll want to do this to automate at least three aspects of building docs:

1. The production delivery of the docs at your chosen interval, up to and including a deployment triggered by merging a branch to master.
2. Immediate staging of a draft edition of your docs with every merge/pull request to the repository, as well as every subsequent commit, so reviewers can review content changes without having to generate the docs locally.
3. Perform integration tests to ensure against merging docs that will break any build they're involved in.

Netlify can handle so much of this so well out of the box, you need to be sitting down when you start to explore their offerings.

Alternative CI/CD Options

Build and deploy tools are robust. If you have one you like and it can incorporate Ruby gems or perform command-line executions, you will be able to integrate all AJYL tooling. If you are just setting out to manage a documentation build, LDCMF and Netlify are the stress-free means to getting started with AJYL, from scratch to production deployment.

Other tools in this category often used to integrate docs with the product build include Jenkins, Travis CI, CircleCI, and CodeShip. Travis CI, CodeShip, and CircleCI all use YAML for configuration.

GitHub & Atom for Content Management

Content management is aided by web interfaces and coding clients. **GitHub**'s Web interface allows editing and managing text files as well as handling commits. GitHub also provides a client-side coding tool called Atom, widely praised by developers and growing in popularity with technical writers. Advanced as they are, these general-purpose tools have many AsciiDoc-aware enhancements, but their content-management features are too general to provide more than rudimentary assistance.

The eventual solution to this problem will be a combination of cloud-based Git and JAMstack tooling such as **Netlify CMS** and Jekyll Admin, which also provide browser interfaces but handle files locally and interact with both local and remote repositories. These tools do not yet play nice with AsciiDoc files, but they offer tremendous potential for integration with highly customized workflows, which would make docs contributions far more accessible to non-technical users.

Truthfully, file management in AJYL is primarily handled with a combination of code editor (such as Atom) and a terminal for command-line interfaces.

OpenAPI/Swagger for REST API Docs

Developer documentation is a special beast with dozens of great FOSS solutions in the category. If your product includes native APIs, you'll want to use specialized tooling designed to output docs for code written in that language: for instance, Javadoc for Java or Sphinx or PyDoc for Python.

REST APIs are a little different, as they are more generic for the end user (the developer integrating something with your API). The OpenAPI specification is a sourcing schema that allows developers to organize REST API information as data, in JSON or YAML format. OpenAPI-sourced docs can be rendered and delivered using Swagger. OpenAPI lets you define *actual product (REST API) functionality* and configuration as well as serve as source for documentation output. Swagger can even generate *interactive REST API test interfaces* within the documentation! If you want to push elements of your API into your AsciiDoc content, there's a plugin for that.

Alternatives to OpenAPI

As noted, most source languages have an associated API-documentation system for native extension. Wikipedia maintains excellent comparison lists of the top contenders in this category. If you are getting started with Native API docs, check out Tom Johnson's guide to just that.

For RESTful API documentation, the main alternative to OpenAPI

Algolia for Search

Every documentation site needs to be searchable. Most static-site generators use weak, burdensome JavaScipt-based frontend search that is not up to the job. Algolia provides an awesome cloud service that makes indexing easy and cheap (if not free).

Other Cloud Search Options

- Swiftype
- Amazon AWS CloudSearch

These options, like Algolia, are highly customizable and will likely handle the vast majority of applications, including for complicated enterprise-scale docsets. If you need more power, however, therea are excellent roll-your-own options.

Self-hosted Search Options

- Elastic
- Solr

Prawn for PDF

Prawn is an excellent PDF rendering engine that integrates almost seamlessly with Asciidoctor (it's another Ruby native), its renderings are somewhat limited. AsciiDoc is the recommended source language for O'Reilly books, so it can convert to excellent formats. That said, Prawn does not provide for post-processing edits. If you are particular about your PDF output, you realize any solution will require manual fine-tuning of output. Asciidoc's Prawn integration is quite customizable, but AsciiDoc can travel to PDF via two other routes.

Alternative PDF Generators

If you are generating PDF and HTML from the same source, do not wander from AsciiDoc. These options might round out your toolchain.

- Asciidoctor's DocBook backend
- Asciidoctor's LaTeX backend

Cloudflare for SSL/TLS, DNS, and CDN

If you don't know what some of these terms mean, you need Cloudflare all the more! Cloudflare provides free HTTPS encryption, domain name services, and an always-on content-delivery network that will serve your site for you if your server goes down.

More to Come

As we take on new challenges in the world of technical documentation, this document will expand, as will the number of links detailing how to implement the various associated solutions. For now, we hope the point is made that the first elaborate FOSS technical documentation stack can handle some pretty gnarly jobs and offers excellent extensibility.

To see an active implementation of AJYL, check out Codewriting, a site about developing docs as code.

When you’re ready to start building, use the other resources here for all your AJYL quickstart tools.