Teetotality

Why build this blog - or anything - on IPFS?

I wrote here how this blog is built. The better question - thanks Aman! 🙏 - is **why**? Why not post these words on Medium, or put a static site on Github Pages or in an S3 bucket, or use Wordpress on a virtual private server, or even go super old-school and run a web server at home (I do have this sweet, sweet Sonic fiber after all). It's not as if we're short on options in 2020.

Is there really something wrong with all of them? Well, I guess it depends what you mean by "wrong"…

From a certain perspective, the internet and the web as we know them (including fundamental technologies such as DNS, TCP/IP, HTTP, SMTP, and even Javascript) are flawed in fundamental ways. Leaving the technicalities aside, how do these flaws manifest? It is hard to stop bad actors from doing bad things: sending too many emails, stealing sensitive data, flooding websites with traffic, spreading false facts and bifurcating the shared reality that allowed for a democratic global order. Paradoxically, the fact that there is no cost associated with basic communication actions - sending an email, tweeting, visiting a website - means that the system as a whole is more subject to manipulation. I.e., the qualities of the internet that make it open, accessible, and "free" are exactly those that make it vulnerable to capture. I won't try to make this argument here, but I will stipulate the following:

> The core technologies on which the internet is built embody some key design choices that
>
> 1. enable centralized control of software systems and data, whether by corporations or the state,
> 2. allow for the pervasive spread of disinformation that is very costly to identify and correct relative to the cost to produce and distribute it, and
> 3. make it quite difficult to implement solutions to those problems by building applications *on top of* them, rather than replacing them.[1]

I think that's the minimum decentralized-web proselytization required to set up the rest of this post, which is really a justification for me to use some nerd tech. And lest it needs be said: IPFS is, at best, a small part of the solution to these problems. It does not, on its own, cure the internet's ails - that's what bitcoin is for. 😜

Content Addressing

The main change that IPFS makes relative to the internet we're used to is in how web pages are named: addresses are tied to *content*, not a particular *server*. To understand the importance of this shift, let's go back to the old days of the internet. Let's say it's 1995, and I want to put a brand new hypertext document on the World Wide Web. First, I get all of my HTML written (including <blink> tags for emphasis!). Now I need to put that document somewhere where it can be found and retrieved. Unlike today, that probably means I'm going to put it in a specific directory on a specific hard drive on a specific computer with a specific IP address connected through a specific network connection. And unlike today, domain names are hard to come by, so I'm either using a folder on an existing web server, or, if I'm particularly well-resourced and adventurous, I'm running my own Apache web server on my very own internet-connected computer.

I go through all of these minutiae because all of these specifics were crucial to the quality, reliability, and speed of the connection, and whether the web page could be discovered by anyone I didn't directly tell about its existence. There were a lot of details to get right, and lots of things could and did go wrong.

In fact, all of these things *still matter*, greatly. They matter so much that, in the intervening years, billions of dollars have been invested, earned, lost, and re-earned in solving these problems better than any individual could do for themselves.[2] And because software automation allows for engineering costs to be spread over a very large number of customers, these services are all available for a tiny fraction of the cost of the web 1.0 era. From a certain perspective, these are solved problems.

Let's look at three reasonably modern approaches to publishing hyperlinked words and pictures on the World-Wide-Web.[3]

1. Wordpress

Wordpress is truly a dinosaur walking among us - an extremely successful dinosaur. It is the most successful example of what we might call the “traditional” dynamic web page model. Your content is stored in a MySQL database, and it is fetched, formatted with templates, and turned into HTML by the Wordpress server-side PHP code. That HTML is then served over HTTP by a traditional web server - either Apache or nginx.

As always, there are many ways to complexify this picture in the name of performance: running multiple servers behind a load balancer, caching pages on the server as HTML so they don't need to be re-generated from the database every time, or using a content delivery network so that pages can be served in many cases without needing to bother the “real” servers at all.

2. Github Pages

Whereas Wordpress relies on a server program to create HTML by pulling words and formatting from a database, a static site is even simpler: the HTML (and CSS, Javascript, image, etc.) files are just stored in a filesystem, and a web server like Apache, nginx, or something else is responsible for translating a URL received from a web browser into a file path, and returning that file to the client. Crucially, this requires much less work on the server side - just fetch the right file(s) from disk, and send them back over the network. No database lookups, no page construction.

While there are many ways of doing this, Github pages is a particularly modern and popular solution. In short, Github allows its users to populate a git repo with a web site (or, in many cases, source files that can be translated into a website in one go through a "static site generator"), and from that point forward they take responsibility for serving those files. While they do this with a lot of complex infrastructure in order to achieve performance, scalability, and reliability, the end result is just a very capable web server that finds files on disk (or in a cache) and sends them back to the client.

3. "Put it on Medium"

Even though Wordpress and Github Pages (along with a static site generator) take care of *a lot* of the work of creating and serving a website, they still require that you, like, *know* what a website is. Even with a lot of the details being taken care of, that's still a pretty high bar for someone whose main goal is to get their words and pictures into as many people's browser tabs as possible. Medium is the latest iteration of a fully-hosted blogging platform, a category which promises that the user doesn't need to know what HTML is, much less HTTP, a web server, load balancers, and DNS.

When it comes to sending words and pictures over the internet, this makes a lot of sense. Medium's engineers are much better at all of this than you will ever be. It's silly for everyone who wants to publish on the internet to have to know or care about the technologies that make it all go. But even though these engineering tasks are very much behind the curtain, they must still be made. Medium's engineers are working from the same set of options as anyone else; they're just getting paid to make them well enough that the service's users never have to worry about it.

Method	Browser Code	Server Rendering	Web Server	Web host	Naming	Routing
Wordpress	None[4]	PHP	nginx / Apache	A known server	DNS	TCP

Method	Browser Code	Server Rendering	Web Server	Web host	Naming	Routing
Github Pages	None[4]	None	¯_(ツ)_/¯[5]	¯_(ツ)_/¯[5]	DNS	TCP
Medium	Javascript	Javascript	¯_(ツ)_/¯[5]	¯_(ツ)_/¯[5]	DNS	TCP
IPFS	None[4]	None	IPFS daemon	Any IPFS node	IPNS	libp2p

So……why IPFS?

As I see it, the interesting arguments for IPFS boil down to these three.[6]

1. Ownership, control, censorship

   At the end of the day, Medium the company controls what happens on medium.com, subject to the laws of the jurisdictions where it operates and the private decisions of its leaders and owners. To a first approximation, Medium's interests are mostly aligned with authors who put their writing there. But that alignment is far from perfect, and an individual writer has almost no say in the decisions the company makes. In many ways, Medium has been better than some of its web predecessors about communicating its goals and values. But the fact remains that its users are wholly subject to its business decisions, without any right to appeal. Of course, Medium is beholden to its customers as a whole - but this is of no help to a user of the system whose needs diverge from the chosen path. In addition to, and intertwined with, this corporate control is the state's ability to dictate how Medium should act, including prohibiting certain content and demanding that certain information be turned over when demanded.
2. Resilience

   As I attempted to communicate in the discussion above, even the most sophisticated modern web services still operate within the fundamental limitations of HTTP and DNS. And while it is *possible* to create systems using these technologies that are robust to many perturbations, outages, and catastrophic events, doing so is difficult, expensive, and harmful to the quality of the user experience. It does not make sense for any profit-seeking entity to prioritize this kind of resilience - certainly not for consumer services based on ad or subscription revenue. From their point of view, the existing tools work well enough.
3. Elegance

   What role should aesthetics play in these decisions? I don't know, and I think I'm pretty middle-of-the-road in the endless war between programmers who favor purity and formal elegance and those who prioritize outcomes and practical considerations. But I will say that content

addressing strikes me, and many software people who come across it, as *obviously* superior to host-based addressing along certain dimensions. But this beauty can certainly feel like a slender reed on which to place the design of a system that needs to get real work done. So weight this as you will.

These advantages are, to varying degrees, abstract and aspirational. Building anything on IPFS in 2020 represents a bet on a future in which priorities have changed, and engineering decisions are therefore made differently. Put another way: IPFS is a technology that belongs to a certain class of futures - a technology that will thrive in those futures (but not others), and that may help to bring them about.

Plus, it's super cool. You should try it!

Resources:

- A beginner's guide to IPFS - Hacker Noon
- Awesome IPFS Resources
- Official IPFS Docs

---

1. To be clear, these are all statements about how these technologies operate through human psychology, and with our sociopolitical context. They are statements about technosocial systems. ↩
2. True story: my software engineering professor at Berkeley in 2000, Eric Brewer, was at the time a co-founder of Inktomi - basically the first of what we would now call a CDN. This was absolutely cutting edge technology at the time. My main memory of him is that he seemed very smug; around that time, he was a paper billionaire. ↩
3. These examples are brutally simplified, and each one represents just one possible way of designing the system. For example, I describe self-hosted Wordpress, but many people use Wordpress' SaaS offering. ↩
4. I.e., no client-side code is *required* - any webpage can include Javascript as a static asset. ↩
5. Not that we *can't* or *don't* know, but that we don't *need* to. ↩
6. You'll see people tout that, with IPFS, you get CDN functionality "for free" - because of content addressing, files can live and be served from anywhere. But, 1), CDNs exist and work well, and 2) just because content *can* be served from any IPFS node doesn't mean that most content will be available from many nodes. ↩

dweb ipfs