

Why IPFS Still Matters for Decentralized Storage in 2025

The Quiet Persistence of IPFS

I first ran into IPFS back in 2016, at a small meetup in Berlin where someone was explaining content-addressed storage over bad coffee. The idea felt like a natural evolution of the web, but the tooling was rough. You had to compile from source, the gateway was slow, and nobody was quite sure what you'd do with it once you had it running. Nearly a decade later, I've watched a lot of decentralized projects come and go. IPFS has persisted, not because it's flashy, but because it solves a real problem that only gets bigger as the web centralizes further.

That problem is link rot. Every time a site goes down or a platform changes its API, millions of links break. IPFS addresses this by addressing content by its hash rather than its location. If you pin a file to the network, anyone with the right hash can retrieve it, regardless of which server originally hosted it. The system doesn't care about geography or hosting provider. It cares about the content itself.

What IPFS Actually Does

At its core, [IPFS](#) is a peer-to-peer hypermedia protocol. Think of it as a distributed file system that turns every participating node into a server. When you request a file, the network finds the closest peer that holds that content and delivers it. The hash of the file becomes its permanent address. If the file changes, the hash changes. This means you can verify that what you receive is exactly what the publisher intended, without relying on a certificate authority or a central domain.

For my own work running a small archival project, this has been transformative. I have a collection of old zines and community newsletters that were hosted on a free platform that shut down without warning. The originals were lost. But I had pinned them to IPFS months earlier. When I retrieved them, the hashes matched. I didn't need to beg a sysadmin for a backup or pay for a new server. The data was already distributed across a handful of peers who had chosen to host it.

Where IPFS Shines

The most obvious use case is for content that must survive its original host. Scientific datasets, public records, and creative works with no commercial backing benefit enormously. A university lab might publish a dataset to IPFS once. Years later, even if the lab moves or loses funding, the data remains accessible as long as at least one peer pins it.

Another strong use is for offline-first applications. Because IPFS treats content as the atomic unit, you can bundle an entire website into a single hash and share it over a local network without any internet connection. I've seen this used at conferences where organizers wanted attendees to access presentation files without hammering the venue's Wi-Fi. It worked better than any file-sharing service I've tried.

Trade-Offs You Need to Know

IPFS is not a magic bullet. The biggest friction point is that content availability depends on someone actively pinning it. If nobody hosts a file, it disappears from the network. This is the same problem that plagues all peer-to-peer systems. You can mitigate it by running your own pinning service or paying a provider, but that reintroduces a degree of centralization.

Performance is another consideration. Fetching a file from IPFS can be slower than a direct HTTP download, especially if the nearest peer is far away or has limited bandwidth. The network has improved a lot since the early days, but it still can't compete with a well-provisioned CDN for latency-sensitive content. For archival purposes, that trade-off is usually acceptable. For a real-time video stream, it is not.

Finally, the tooling ecosystem is fragmented. There are multiple implementations, several gateway services, and a handful of pinning providers. Choosing the right stack requires some research. The community is helpful, but you can't just install one package and expect everything to work out of the box.

How IPFS Fits Into a Broader Decentralized Strategy

IPFS is often paired with other technologies to fill its gaps. For example, a blockchain can record the hash of a dataset at a given point in time, providing a tamper-proof timestamp. A naming system like ENS (Ethereum Name Service) can map a human-readable name to an IPFS hash, so you don't have to remember a long string of characters. I've used this combination to create a simple, permanent link for a digital art project. The artwork itself lives on IPFS, the hash is recorded on a smart contract, and the ENS name resolves to that hash. Two years later, the link still works.

For site owners considering IPFS for their own content, the first step is to decide how much control you want to give up. If you run a gateway that serves your content over HTTP, your visitors never interact with the peer-to-peer network directly. That's fine for most use cases. But the real value of IPFS comes when you let your users become peers, too. When they access content, they automatically cache it and can serve it to others. This turns your audience into a distributed hosting network.

Practical Advice for Getting Started

If you want to try IPFS today, the easiest path is to use a public gateway like ipfs.io. Upload a file through the web interface, and you get back a hash. Share that hash with anyone, and they can retrieve the file through any IPFS gateway. This works for testing but doesn't guarantee persistence, because public gateways may evict unpinned content after a while.

For a more durable setup, install `go-ipfs` (the reference implementation) on a server or a Raspberry Pi. Initialize the repo, pin your files, and connect to the public swarm. The default configuration is safe enough for a personal node. If you want to pin content for others, you might look into IPFS Cluster, which coordinates multiple nodes to act as a single pinning service.

There are also managed services like Pinata or Eternum that handle the infrastructure for a fee. They are not free, but they save you the operational overhead. For a small project, a single VPS running `go-ipfs` with a few gigabytes of pinned content costs around five to ten dollars a month. That is cheaper than most object storage services, and you get the redundancy of the peer-to-peer network on top.

What I Wish Someone Had Told Me

Don't treat IPFS as a drop-in replacement for a web server. The mental model is different. You need to think in terms of content-addressed data, not URLs. Planning how to update content is also important. Because changing a file changes its hash, you need a strategy for pointing users to the latest version. A simple approach is to maintain a separate index file that lists the current hash for each piece of content. Update that index as you publish new versions.

Also, be aware that IPFS is not encrypted by default. The content is public to anyone who knows the hash. If you need privacy, encrypt the file before adding it to IPFS. The network does not provide access control. That's by design, but it means you must layer encryption on top if you need it.

The Long View

IPFS has matured a lot since those early days in Berlin. The protocol is stable, the implementations are reliable, and the community is pragmatic. It has not taken over the web, and it probably never will. But for specific niches, it is genuinely useful. If you care about preserving digital content, if you want to reduce dependency on a single hosting provider, or if you are building something that should outlast its original infrastructure, IPFS is worth the effort.

The quiet persistence of the protocol mirrors its own design. Content that is worth keeping finds a home somewhere in the network. The technology itself does not guarantee that, but it gives you the tools to make it happen. And in a time when so much of the web feels fragile, that is a meaningful thing to have.