



POSSIBILITY SPACE • FULL USE CASE

01

Decentralized Frontend

Decentralized hosting — no server to run, no middleman to depend on.

BUILD STATE

Claimed

COMPLEXITY

●●○○○

Vision

A website stays online only for as long as everything underneath it holds. Any link in that chain can break. A cloud provider can suspend an account over a billing dispute or an automated abuse flag, without a person ever reviewing the case. A registrar can seize a domain under pressure from one jurisdiction; a government can order a site blocked or restricted. And none of it has to be deliberate: when a major CDN or cloud region goes down, millions of sites vanish with it for hours. A policy change, a takedown order, or an outage on a Tuesday is enough to take a project offline by Wednesday.

Most builders never think about these dependencies until one breaks. A side project, a community tool, a small business's storefront: all of it sits on infrastructure owned by someone else, reachable only for as long as that someone else stays willing — and stays up.

Swarm removes the middleman from the equation. A site published to Swarm isn't installed on a server. It's split into pieces and distributed across a network of independent participants, none of whom individually decides whether the site stays up. There's no single account to suspend, no single machine whose failure takes the site down, and no central chokepoint where pressure can be applied. Censorship becomes structurally hard rather than one phone call away, and downtime becomes a local event the network routes around — resilience that grows with every participant that joins.

The address stays usable, too. A site can be updated, a new build, a fixed typo, a redesign, without anyone needing to notify visitors of a new link or re-point a domain. The reference people already have keeps resolving to whatever the latest version is.

This isn't hypothetical. justdeploy.eth, a guide to deploying on Swarm, is itself published this way: it teaches the exact mechanism it depends on to stay online.

Architecture

A site on Swarm doesn't live on a server. Every mechanism below works toward the same end: making the site's availability a property of the network rather than a property of any one company or machine.

Content addressing and chunks (shipped). Uploading a site splits its files into fixed-size chunks, each addressed by the hash of its own contents, and scatters them across Swarm's shared storage layer, the same layer every other piece of content on the network uses. No single node holds the whole site, and no single node needs to be reachable for someone else to load it; a chunk is served by whichever nodes happen to be holding it.

Manifests (shipped). A root manifest ties the chunks back together: it's the map from a file path like `/index.html` or `/assets/logo.svg` to the chunks that hold it, so a single reference is enough to reconstruct the entire site. Uploaded once, that manifest reference would normally be fixed forever, a new version would mean a new, different reference.

Feeds and Single Owner Chunks (shipped). A feed makes it possible to update. A feed is built from Single Owner Chunks, signed by an Ethereum keypair, giving a static address for content that changes over time. Publishing a new build means writing a new manifest reference to the feed; the feed's own address doesn't change, so a link shared once keeps resolving to whatever the latest version is.

Postage stamps (shipped). Keeping all of this available costs postage stamps, purchased as a batch (depth sets capacity, amount funds duration) and paid upfront by whoever uploads the site. This

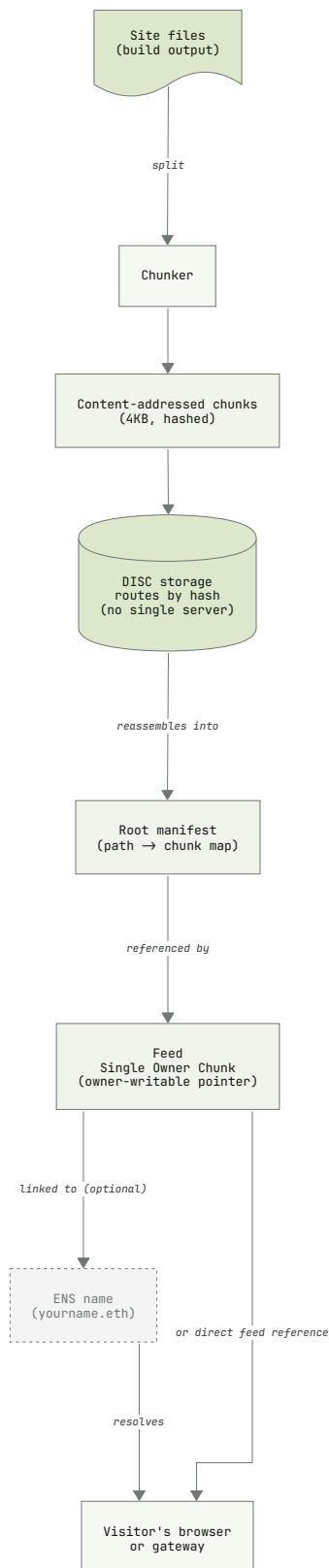
replaces a hosting bill: instead of paying a company every month to keep a server running, a stamp pays the network directly, for a duration chosen at upload time. Batches default to immutable, meaning earlier chunks in the batch aren't silently overwritten as new ones are added, which matters for a site whose older manifest versions should stay retrievable even after a redeploy.

ENS (external). For a human-readable address, the feed's manifest reference can be set as the content hash on an ENS domain's resolver. Once set, Bee resolves that ENS name straight to the feed's latest content, no further ENS transaction needed on future updates: the feed changes, the ENS record doesn't have to.

PRIMITIVES

Content addressing DISC storage Manifests Feeds / Single Owner Chunks Postage stamps

EXTERNAL DEPENDENCIES ENS



Data flow. Site files are split by a chunker into content-addressed chunks and written to DISC storage, no single server involved. A root manifest maps file paths back to those chunks; a feed (an owner-writable pointer) references the current manifest, so the same feed address survives every future update. An ENS name can optionally resolve straight to the feed.

Development

Three tools cover this at every skill level, and all three do the same thing underneath: buy a postage stamp — Swarm’s prepaid storage, capacity plus duration bought upfront instead of a monthly hosting bill — upload the build output, point a name at the result. This covers any site that builds to static files; a server-rendered app needs a static build step first, with backend logic living in smart contracts instead of a server.

Easy — Beeport. Uploads a build folder straight from the browser and buys the postage stamp as part of the flow. No install, no local node; a wallet is enough to get started.

Intermediate — Swarm Desktop. The same upload-and-stamp flow as Beeport, wrapped in a desktop GUI backed by a local node, better suited to repeat deploys from the same machine.

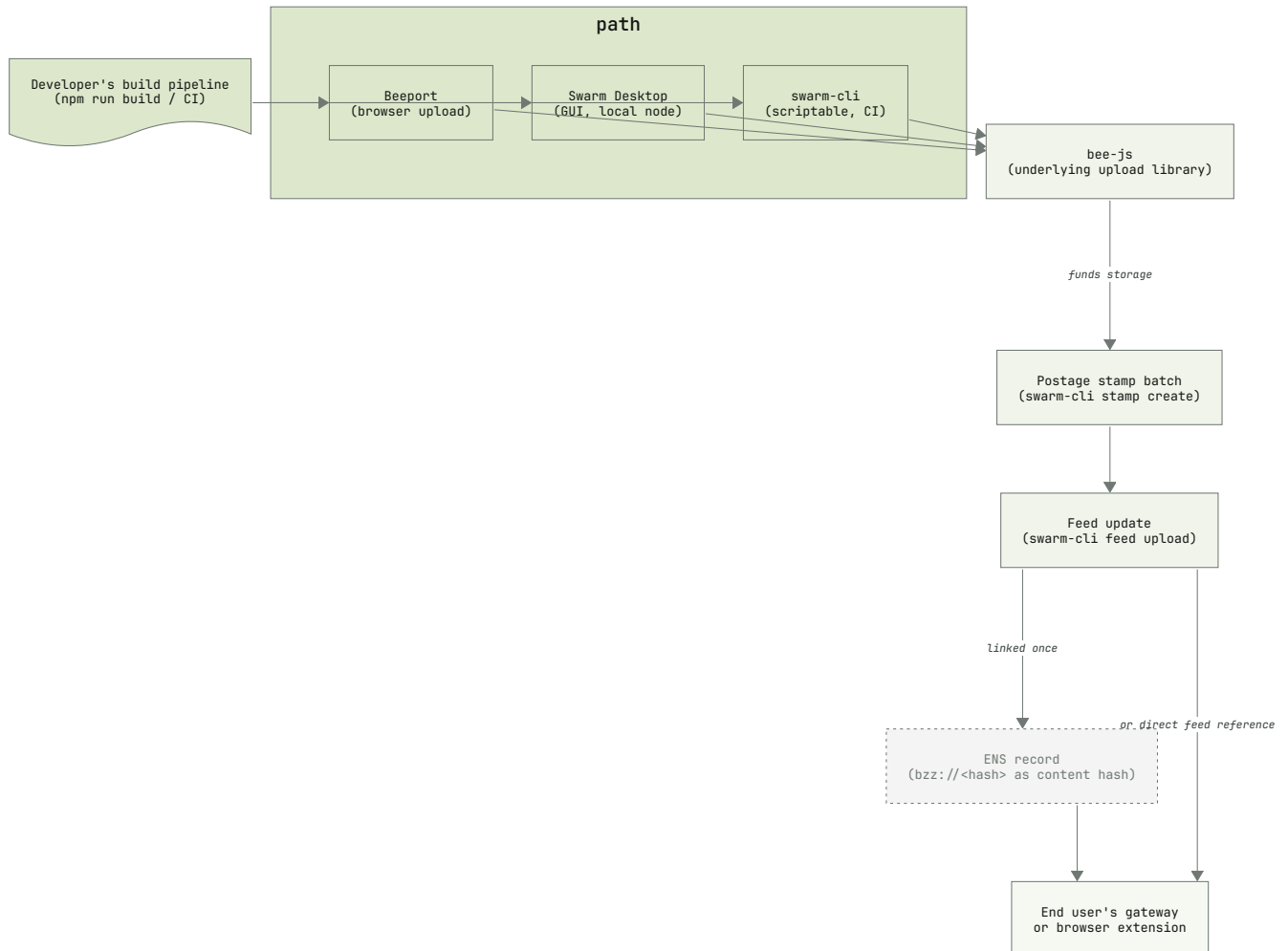
Advanced — Bee + swarm-cli. For CI/CD or anything scripted: `swarm-cli quickstart` sets up a Bee light node, `swarm-cli stamp create --capacity 680MB --ttl 3months` buys storage sized to the site (it accepts human units like 1GB or 1w/3months; immutable by default, which stops the batch from overwriting the site’s own earlier chunks as new ones are written), and `swarm-cli feed upload ./dist` uploads the build folder, prompting for a stamp and an identity if neither exists yet.

Pointing a name at it. Once the first upload is live, take the feed’s manifest reference and set it as the ENS content hash for the domain, a one-time step. Every future deploy reuses the same feed and the same ENS record; only the manifest the feed points to changes, so nothing downstream needs updating.

Repeat deploys / CI. `swarm-cli stamp list` checks what’s already funded, and `swarm-cli stamp topup` or `swarm-cli stamp dilute` extend an existing batch’s duration or capacity rather than buying a new one. `swarm-cli feed upload` then pushes each new build non-interactively, as long as a stamp and identity are already set up.

TOOLS & REFERENCES

- Bee documentation — <https://docs.ethswarm.org> (Feeds, postage stamps, manifests)
- bee-js library — <https://bee-js.ethswarm.org>
- swarm-cli — <https://github.com/ethersphere/swarm-cli>
- Beeport — <https://beeport.ethswarm.org>
- Swarm Desktop — <https://desktop.ethswarm.org>
- justdeploy.eth — full walkthrough of all three deploy paths, hosted on Swarm itself
- Workshop video: How to decentralise any frontend — the same three paths, live (youtube.com/live/ofskcDHRgv0)
- **External dependency (not a Swarm primitive):** ENS (ens.domains)



Component assembly. A developer's build pipeline goes through one of three paths, Beeport, Swarm Desktop, or swarm-cli, all driving bee-js underneath. bee-js buys a postage stamp batch and writes a feed update; an ENS record is set once and then keeps resolving to whatever the feed points to. The end user reaches the site through a gateway or browser extension.

Reference implementations

Claimed. justdeploy.eth is a real, live guide to deploying a site on Swarm, itself hosted the way it teaches: split into chunks, referenced by a feed, reachable at an ENS name. It covers all three paths in this document (Beeport, Swarm Desktop, Bee + swarm-cli) with working commands and a live postage-stamp cost calculator. Built and maintained by the Swarm Foundation DevRel team.