

The AT Stack

Components of the AT Protocol Stack

The Authenticated Transfer Protocol

The AT Protocol (Authenticated Transfer Protocol) is a standard for public conversation and an open-source framework for building social apps.

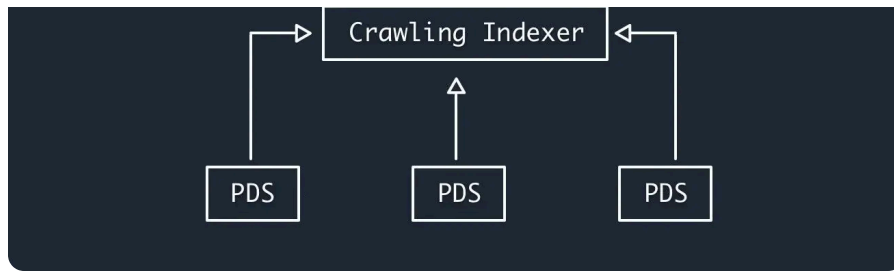
It creates a standard format for user identity, follows, and data on social apps, allowing apps to interoperate and users to move across them freely. It is a federated network with account portability.

Decentralized Architecture

AT syncs repositories in a federated networking model. Federation ensures the network is convenient to use and reliably available. Repository data is synchronized between servers over standard web technologies (HTTP and WebSockets).

Personal data servers, or PDSes, are your home in the cloud. They host your data, distribute it, manage your identity, and orchestrate requests to other services to give you your views.

Relays handle all of your events, like retrieving large-scale metrics (likes, reposts, followers), content discovery (algorithms), and user search.



The AT Protocol is architected in a “big world with small world fallbacks” way, modeled after the open web itself. With the web, individual computers upload content to the network, and then all of that content is then broadcasted back to other computers. Similarly, with the AT Protocol, we're sending messages to a much smaller number of big aggregators, which then broadcast that data to personal data servers across the network.

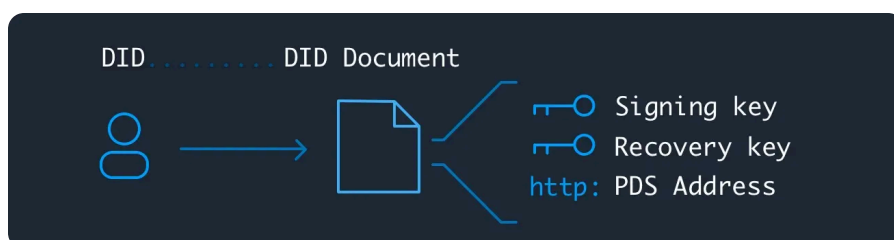
Technically, you can implement an AT application with just a single PDS — Relays are not a necessary part of the specification. They enable the ecosystem to scale for big-world social networks.

PDS

PDS instances host accounts for users, which require account management and lifecycle controls similar to any network server. While AT identities (DIDs and handles) can in theory be entirely separate from the PDS, in practice the PDS is expected to help manage the user's identity.

You can [host your own PDS](#).

User data is stored in signed data repositories and verified by DIDs. Signed data repositories are like Git repos but for database records, and DIDs are essentially registries of user certificates, similar in some ways to the TLS certificate system. They are expected to be secure, reliable, and independent of the user's PDS.



The PDS handles:



- AT identity resolution: DIDs, handles. [Read more](#) about how identity resolution.
- storage of preferences and private state
- ability to takedown or suspend accounts, and handle moderation contacts
- secret key management: AT signing key, and PLC rotation key
- [PLC](#) identity operations and endpoints: change handle; validate, sign, and submit PLC ops
- email delivery: for reset flows, as well as forwarding moderation mail
- data synchronisation using repository event streams
- output `#identity` and `#account` (coming soon) events on repo event stream
- usually the ability to manage a default handle namespace (base domain)
- OAuth client flows, including web sign-in pages, and optional MFA

You can migrate an individual account's data repository freely from one PDS to another. This involves the import and export of repositories as CAR files. There are some community tools that make this easier:

- <https://atproto.at/>, which lets you browse all the data stored in your own repository (and what PDS it's currently hosted on)
- <https://pdsmoveover.com/>, a collection of tools for PDS migration
- <https://boat.kelinci.net/>, a set of interfaces for repository import and export
- <https://github.com/bluesky-social/goat>, our Go AT command-line tool.

PDSs implement many AT endpoints, and by default will proxy requests to other services. Caching, CDNs, and so on are typically implemented at the [App View](#) layer. Most other components of the AT stack are designed to direct load *away* from a PDS, so that hosting user data remains economical.

Relays

Relays handle all of your events in real-time. The Relay also handles rate limiting and network abuse of individual PDSes to ensure that availability from across the network. Importantly, relays themselves do not interpret records in repositories; they simply store them temporarily and forward them to consumers. Relays produce a [firehose](#) of events, which combines the [event streams](#) from multiple repositories and PDSes.

Relays can be operated at different scales. Full-network relays track all repositories across the entire network, providing complete coverage. Partial-network relays might



monitoring, or archive preservation.

A full network relay isn't particularly expensive to operate, and doesn't require specialized hardware, though it does require significant ingress and egress network bandwidth, which is the main cost in operating a relay.

There are several relays available, each may have their own rate limits and network policies. Some of the available relays are:

- `relay.fire.hose.cam` and `relay3.fr.hose.cam` from [microcosm](#).
- `atproto.africa` from [Blacksky](#)
- `relay.upcloud.world` from [Upcloud](#)
- The two from [Bluesky PBC](#).
- The three from [firehose.network](#)

The relays operated by [Bluesky PBC](#) are probably the most commonly used since this is the default relay for the reference PDS server implementation which many PDS hosts use. They have documented their [rate limits](#) on their website.

Tap

Tap is used for syncing existing records from the AT network from PDS repositories and then continuing to receive new records from the firehose via a relay.

Tap simplifies AT sync by handling the firehose connection, verification, backfill, and filtering. Your application connects to a Tap and receives simple JSON events for only the repos and collections you care about, no need to worry about binary formats for validating cryptographic signatures. Tap features:

- verifies repo structure, MST integrity, and identity signatures
- automatic backfill: fetches full repo history from PDS when adding new repos
- filtered output: by DID list, by collection, or full network mode
- ordering guarantees: live events wait for historical backfill to complete
- delivery modes: WebSocket with acks, fire-and-forget, or webhook
- single Go binary
- SQLite or Postgres backend
- designed for moderate scale (millions of repos, 30k+ events/sec)



App Views

App Views handle application specific logic like retrieving large-scale metrics (likes, reposts, followers), content discovery (algorithms), and user search.

An App View is the piece that actually assembles your feed and all the other data you see in the app, and is downstream from a Relay's firehose of data. This is a semantically-aware service that produces aggregations across the network and views over some subset of the network. For example, the Relay might crawl to grab data such as a certain post's likes and reposts, and the app view will output the count of those metrics.

There can be an ecosystem of App Views for each lexicon deployed on the network. For example, *Bluesky* currently supports a micro-blogging mode: the `app.bsky` Lexicon. Developers who create new lexicons could deploy a corresponding App View that understands their Lexicon to service their users. Other lexicons could include [video](#), or [long-form blogging](#), or [collaborative coding](#). By bootstrapping off of an existing Relay, data collation will already be taken care of for these new applications. They can benefit from the critical mass of users already posting on Bluesky and populating their repositories with other Lexicons; they only need to provide the indexing behaviors necessary for their application.

App Views are application-level tooling, rather than inherent parts of the AT protocol. App Views respond to [XRPC](#) calls over HTTP.

You can find a well-documented example of other App View implementations in [The Blacksky Architecture](#). Different Atmosphere applications may make different implementation choices.

DID PLC

By default, new atproto identities are created with as `did:plc` documents.

`did:plc` is a self-authenticating [Decentralized Identifier](#) which is strongly-consistent, recoverable, and allows for key rotation. Each identity is made up of a series of signed operations that each reference the preceding operation in a chain back to the genesis operation. Any observer can verify the chain of operations in order to verify the current state of the identity.

PLC operations are stored in a directory (currently run at [plc.directory](#)) which is a centralized service that validates operations, stores them, and serves those operations



operation on their behalf.

For guidance on PLC mirroring, refer to [this developer blog](#).

Feeds

As with Web search engines, users are free to select their aggregators. Feeds, search indices, and entire app views can be provided by independent third parties, with requests routed by the PDS based on user configuration.

Custom feeds, or feed generators, are services that provide custom algorithms to users through the AT Protocol. This allows users to choose their own timelines, whether it's an algorithmic For You page or a feed of entirely cat photos.

Custom feeds work straightforwardly: the server receives a request from a user's server and returns a list of post URIs with some optional metadata attached. Those posts are then hydrated into full views by the requesting server and sent back to the client.

A Feed Generator service can host one or more algorithms. The service itself is identified by DID, while each algorithm that it hosts is declared by a record in the repo of the account that created it. For instance, feeds offered by Bluesky will likely be declared in `@bsky.app`'s repo. Therefore, a given algorithm is identified by the at-uri of the declaration record. This declaration record includes a pointer to the service's DID along with some profile information for the feed.

[Browse](#) custom feeds.

Osprey and Ozone

Moderation is handled by two parts of our stack: *Osprey*, an event stream decisions engine and analysis UI designed to investigate and take automatic action; and *Ozone*, a labeling service and web frontend for making moderation decisions.

Tools and SDKs



- The [Indigo](#) repository contains most of our Go tooling, including the code for the [Relay](#) and the Go SDK.
- There's also <https://github.com/bluesky-social/goat>, our Go AT command-line tool.
- The <https://github.com/bluesky-social/atproto> repository contains most of our TypeScript tooling, including the code for the [PDS](#) and the TypeScript SDK. The [atcute](#) community project provides alternative TypeScript packages.
- There are many other community tools that we use every day. For example, the [Microcosm](#) tooling, written in Rust, provides a set of APIs to help build query layers without requiring dedicated AppViews! Refer to <https://constellation.microcosm.blue/> for details.

[← PREVIOUS](#)[NEXT →](#)[Using Ozone](#)[Self-hosting](#)