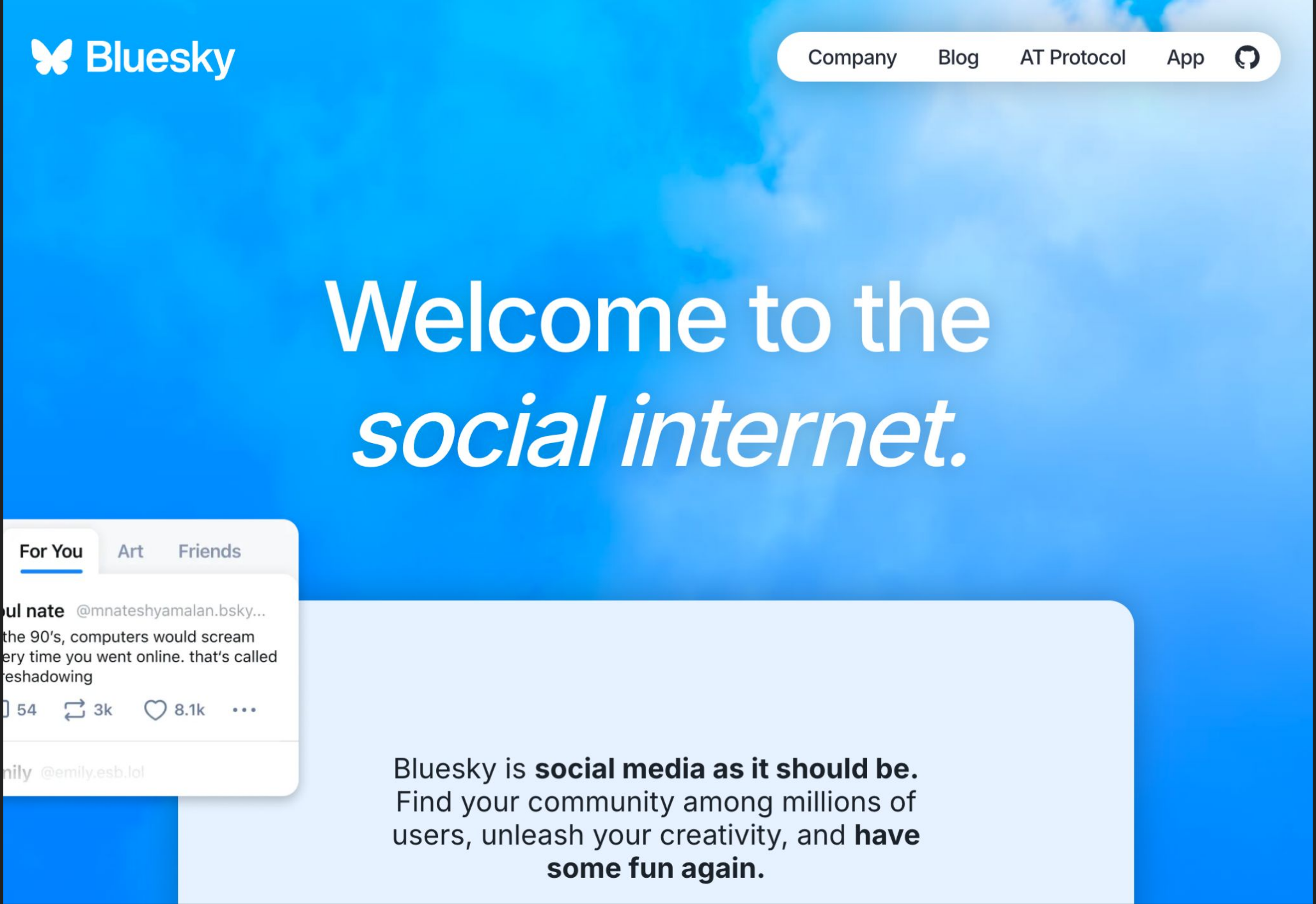
AT: The Billion-Edge Social Graph

Alex Garnett
Bluesky Social

SCaLE 2026

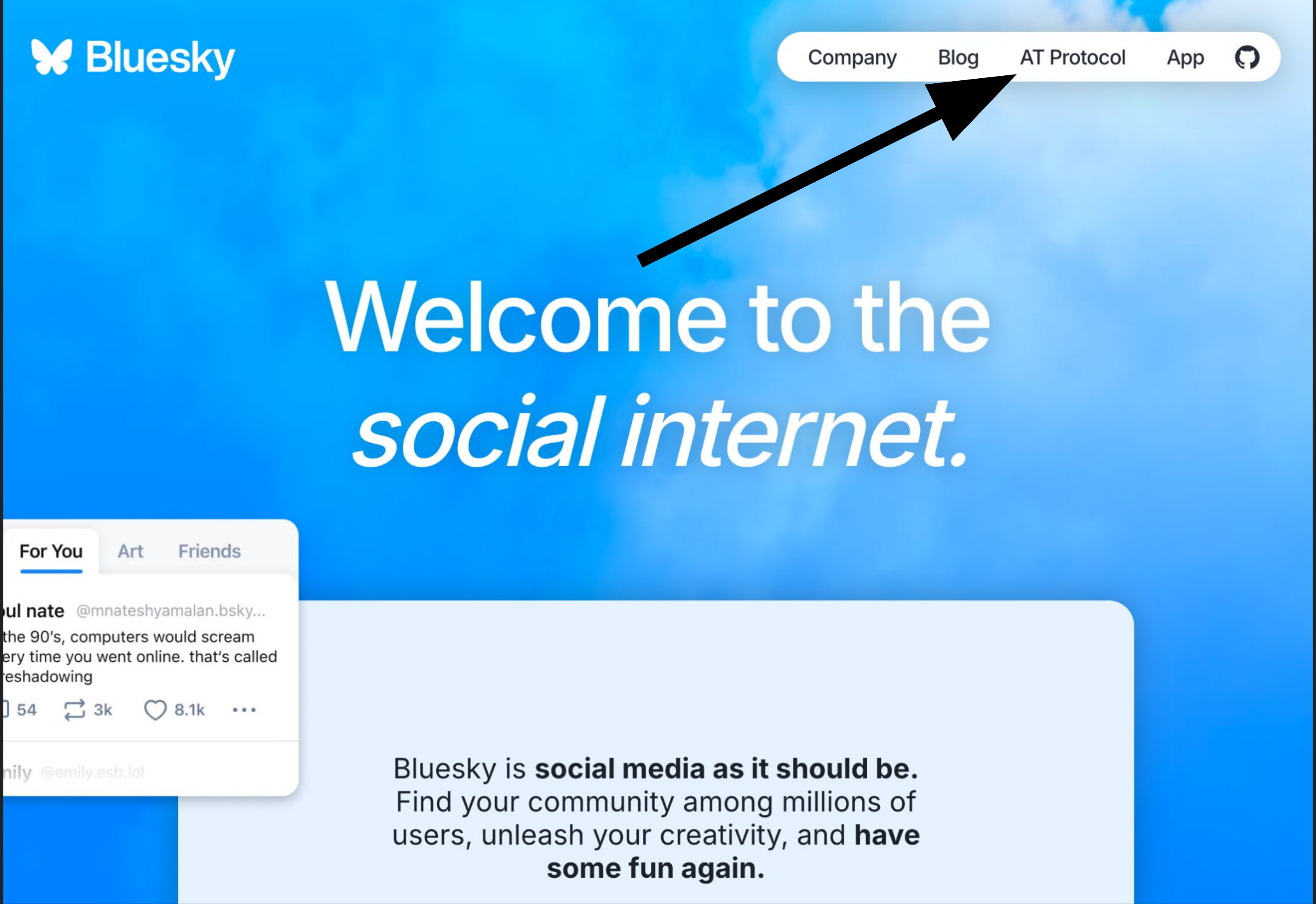


Welcome to the *social internet.*

ul nate @mnateshyamalan.bsky...

the 90's, computers would scream ery time you went online. that's called eshadowing

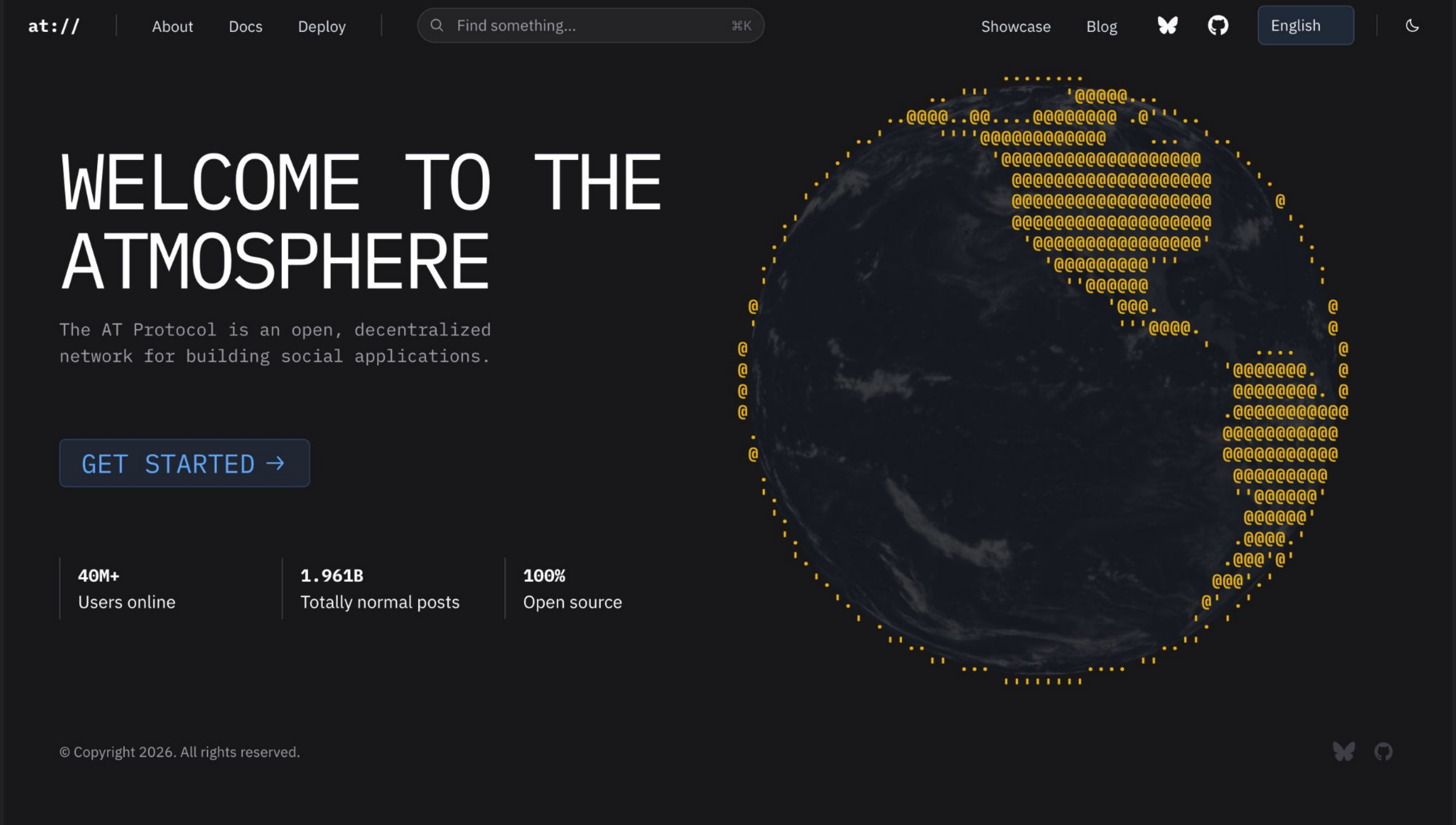
Bluesky is **social media as it should be.** Find your community among millions of users, unleash your creativity, and **have some fun again.**



Welcome to the *social internet.*

ul nate @mnateshyamalan.bsky...
the 90's, computers would scream
ery time you went online. that's called
eshadowing

Bluesky is **social media as it should be.** Find your community among millions of users, unleash your creativity, and **have some fun again.**



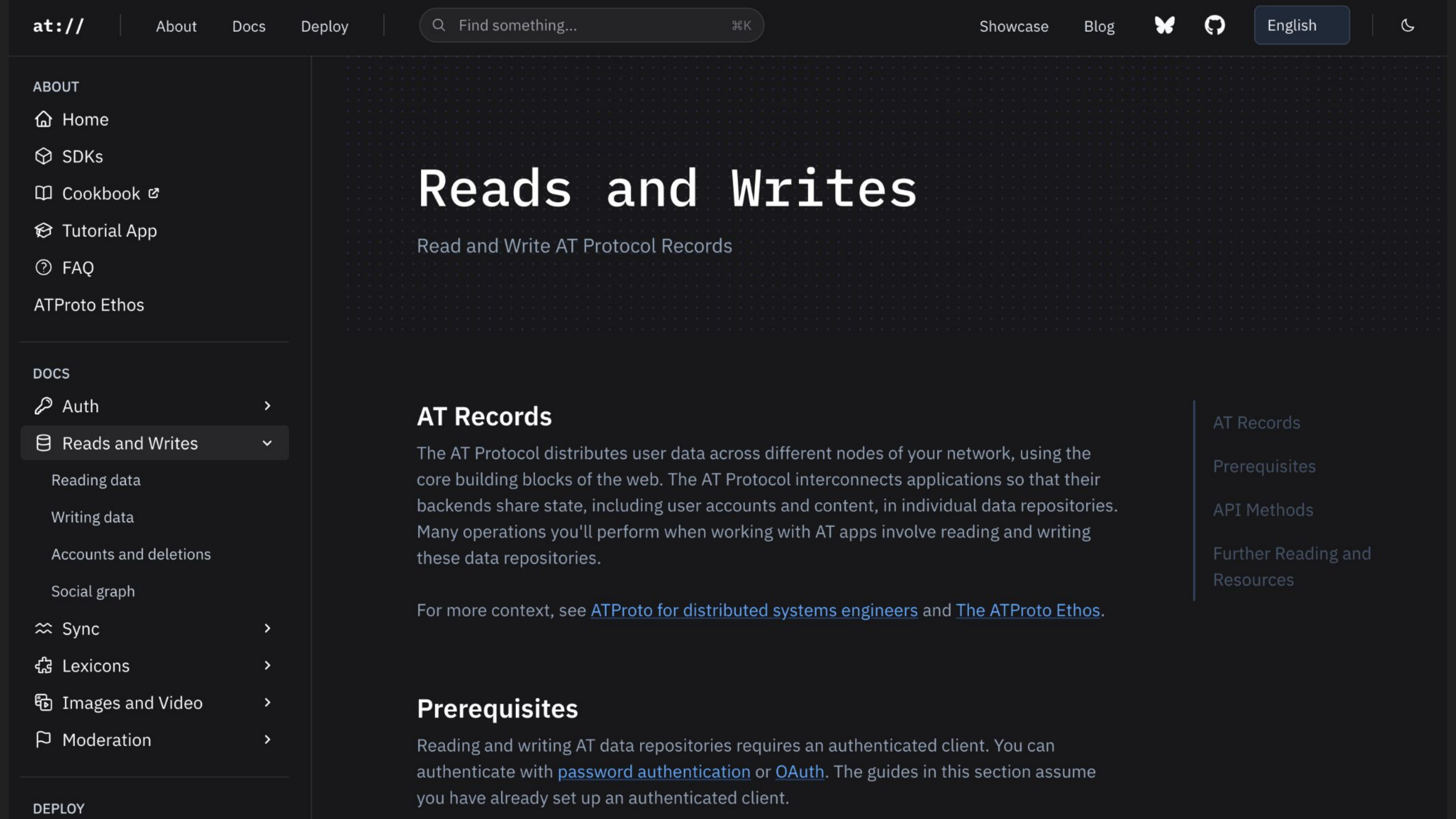
WELCOME TO THE ATMOSPHERE

The AT Protocol is an open, decentralized network for building social applications.

40M+
Users online

1.961B
Totally normal posts

100%
Open source



Reads and Writes

Read and Write AT Protocol Records

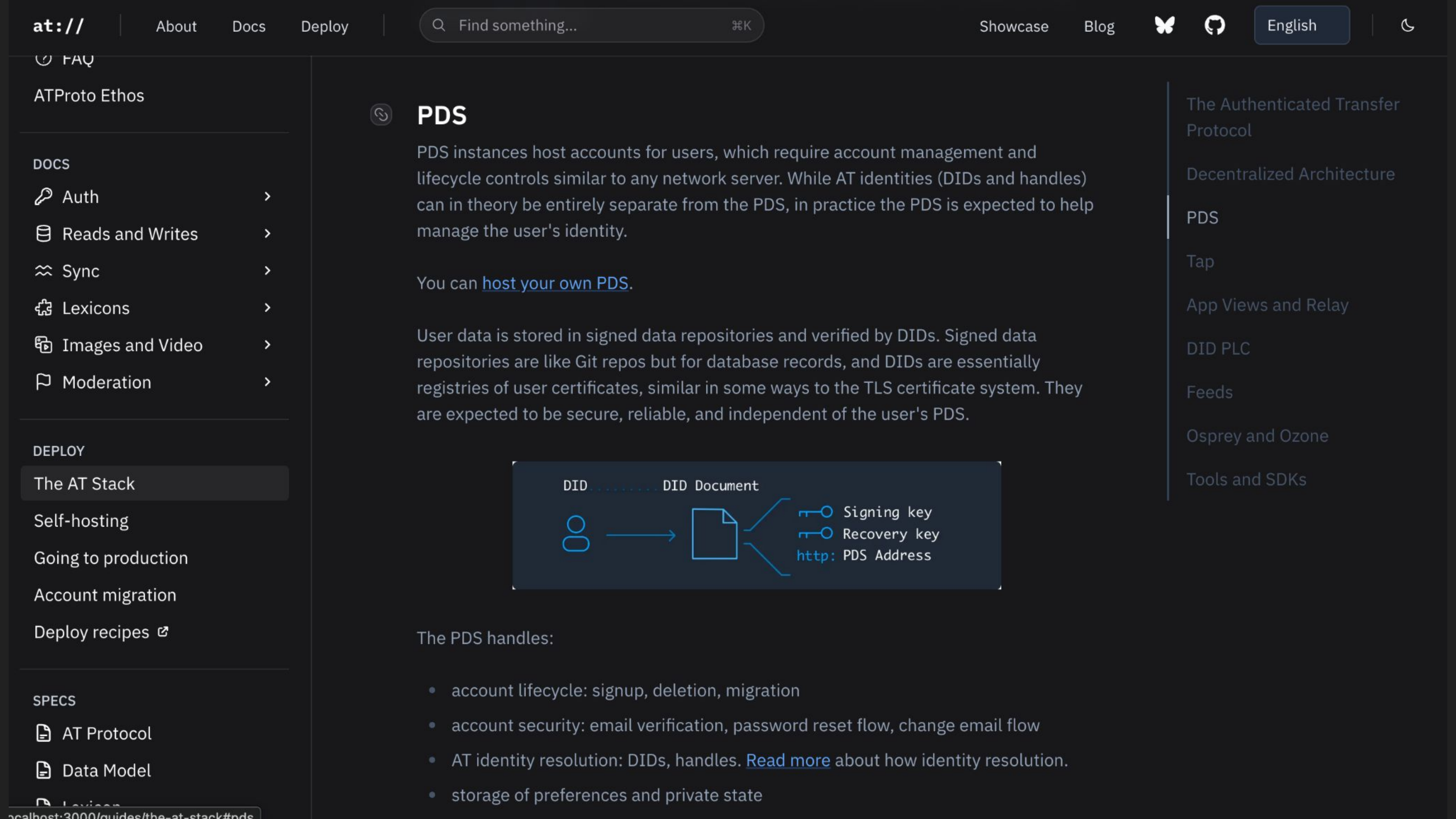
AT Records

The AT Protocol distributes user data across different nodes of your network, using the core building blocks of the web. The AT Protocol interconnects applications so that their backends share state, including user accounts and content, in individual data repositories. Many operations you'll perform when working with AT apps involve reading and writing these data repositories.

For more context, see ATProto for distributed systems engineers and The ATProto Ethos.

Prerequisites

Reading and writing AT data repositories requires an authenticated client. You can authenticate with password authentication or OAuth. The guides in this section assume you have already set up an authenticated client.



FAQ
ATProto Ethos

DOCS
Auth
Reads and Writes
Sync
Lexicons
Images and Video
Moderation

DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes

SPECS
AT Protocol
Data Model
Lexicon

PDS

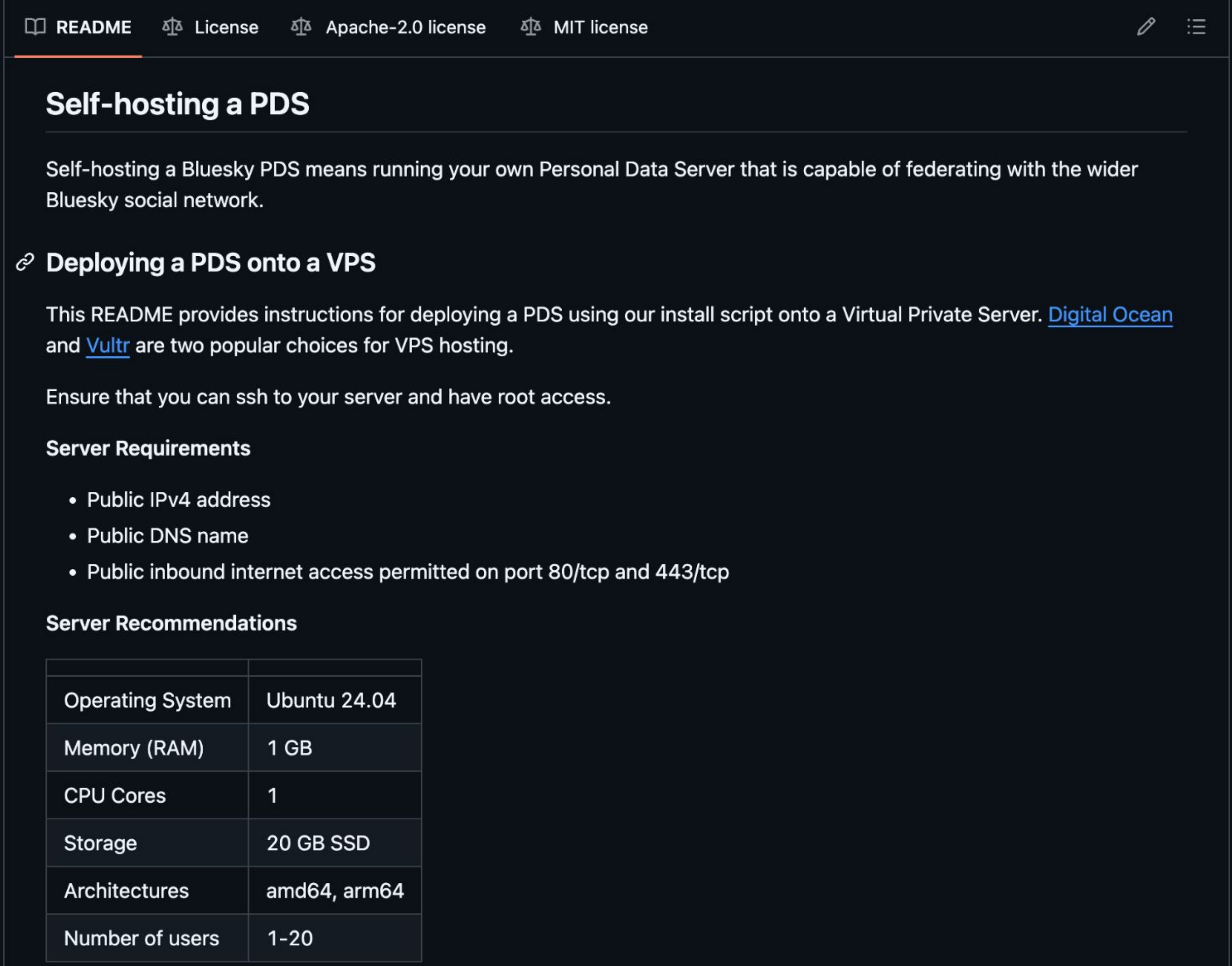
PDS instances host accounts for users, which require account management and lifecycle controls similar to any network server. While AT identities (DIDs and handles) can in theory be entirely separate from the PDS, in practice the PDS is expected to help manage the user's identity.

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:

- account lifecycle: signup, deletion, migration
- account security: email verification, password reset flow, change email flow
- AT identity resolution: DIDs, handles. Read more about how identity resolution.
- storage of preferences and private state



Self-hosting a PDS

Self-hosting a Bluesky PDS means running your own Personal Data Server that is capable of federating with the wider Bluesky social network.

Deploying a PDS onto a VPS

This README provides instructions for deploying a PDS using our install script onto a Virtual Private Server. Digital Ocean and Vultr are two popular choices for VPS hosting.

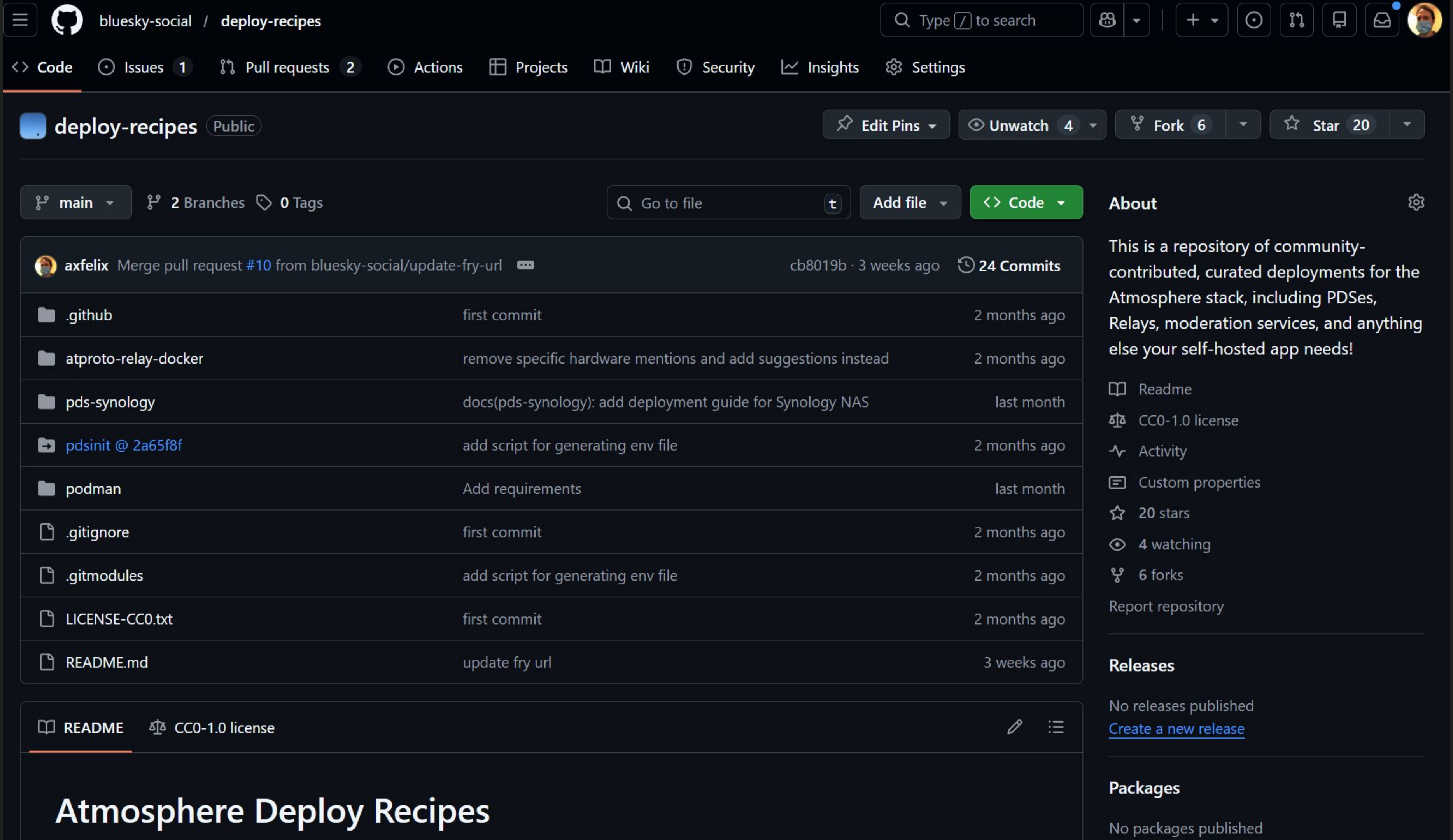
Ensure that you can ssh to your server and have root access.

Server Requirements

- Public IPv4 address
- Public DNS name
- Public inbound internet access permitted on port 80/tcp and 443/tcp

Server Recommendations

Operating System	Ubuntu 24.04
Memory (RAM)	1 GB
CPU Cores	1
Storage	20 GB SSD
Architectures	amd64, arm64
Number of users	1-20



axfelix Merge pull request #10 from bluesky-social/update-fry-url cb8019b · 3 weeks ago 24 Commits

.github	first commit	2 months ago
atproto-relay-docker	remove specific hardware mentions and add suggestions instead	2 months ago
pds-synology	docs(pds-synology): add deployment guide for Synology NAS	last month
pdsinit @ 2a65f8f	add script for generating env file	2 months ago
podman	Add requirements	last month
.gitignore	first commit	2 months ago
.gitmodules	add script for generating env file	2 months ago
LICENSE-CC0.txt	first commit	2 months ago
README.md	update fry url	3 weeks ago

Atmosphere Deploy Recipes

About

This is a repository of community-contributed, curated deployments for the Atmosphere stack, including PDSes, Relays, moderation services, and anything else your self-hosted app needs!

- Readme
- CC0-1.0 license
- Activity
- Custom properties
- 20 stars
- 4 watching
- 6 forks

Report repository

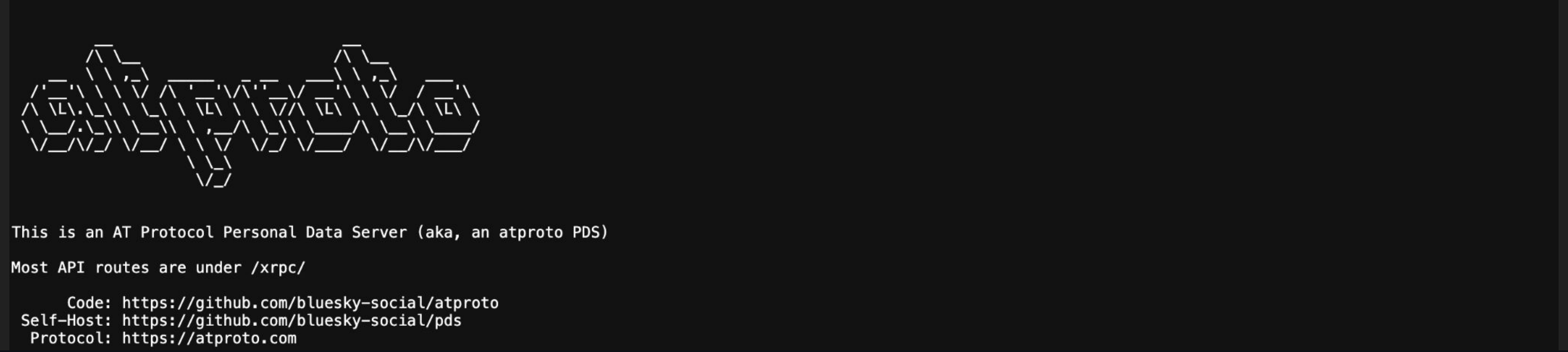
Releases

No releases published

Create a new release

Packages

No packages published

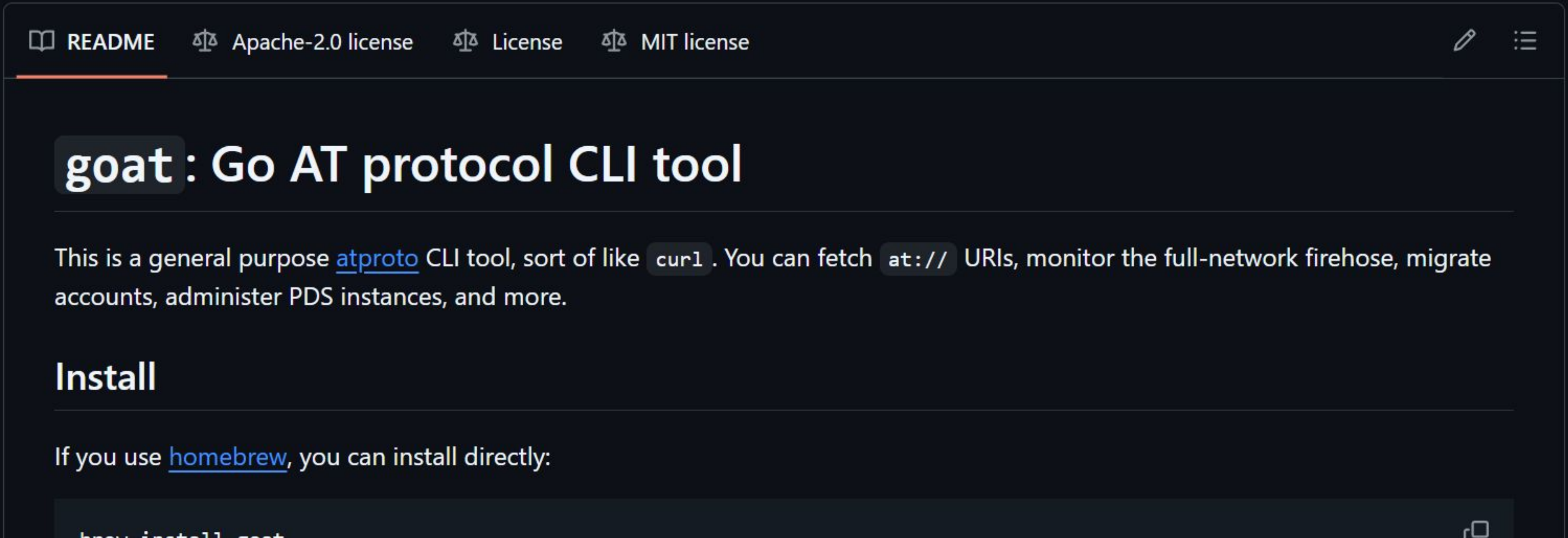


```
         __                         __
        /\ \__                     /\ \__
   __   \ \ ,_\  _____   _ __   ___\ \ ,_\   ___
 /'__'\  \ \ \/ /\ '__'\/\''__\/ __'\ \ \/  / __'\
/\ \L\.\_ \ \ \_\ \ \L\ \ \ \//\ \L\ \ \ \_/\ \L\ \
\ \__/.\_\ \ \__\\ \ ,__/\ \_\\ \____/\ \__\ \____/
 \/__/\/_/  \/__/ \ \ \/  \/_/ \/___/  \/__/\/___/
                   \ \_\
                    \/_/


This is an AT Protocol Personal Data Server (aka, an atproto PDS)

Most API routes are under /xrpc/

      Code: https://github.com/bluesky-social/atproto
 Self-Host: https://github.com/bluesky-social/pds
  Protocol: https://atproto.com
```



`goat`: Go AT protocol CLI tool

This is a general purpose atproto CLI tool, sort of like `curl`. You can fetch `at://` URIs, monitor the full-network firehose, migrate accounts, administer PDS instances, and more.

Install

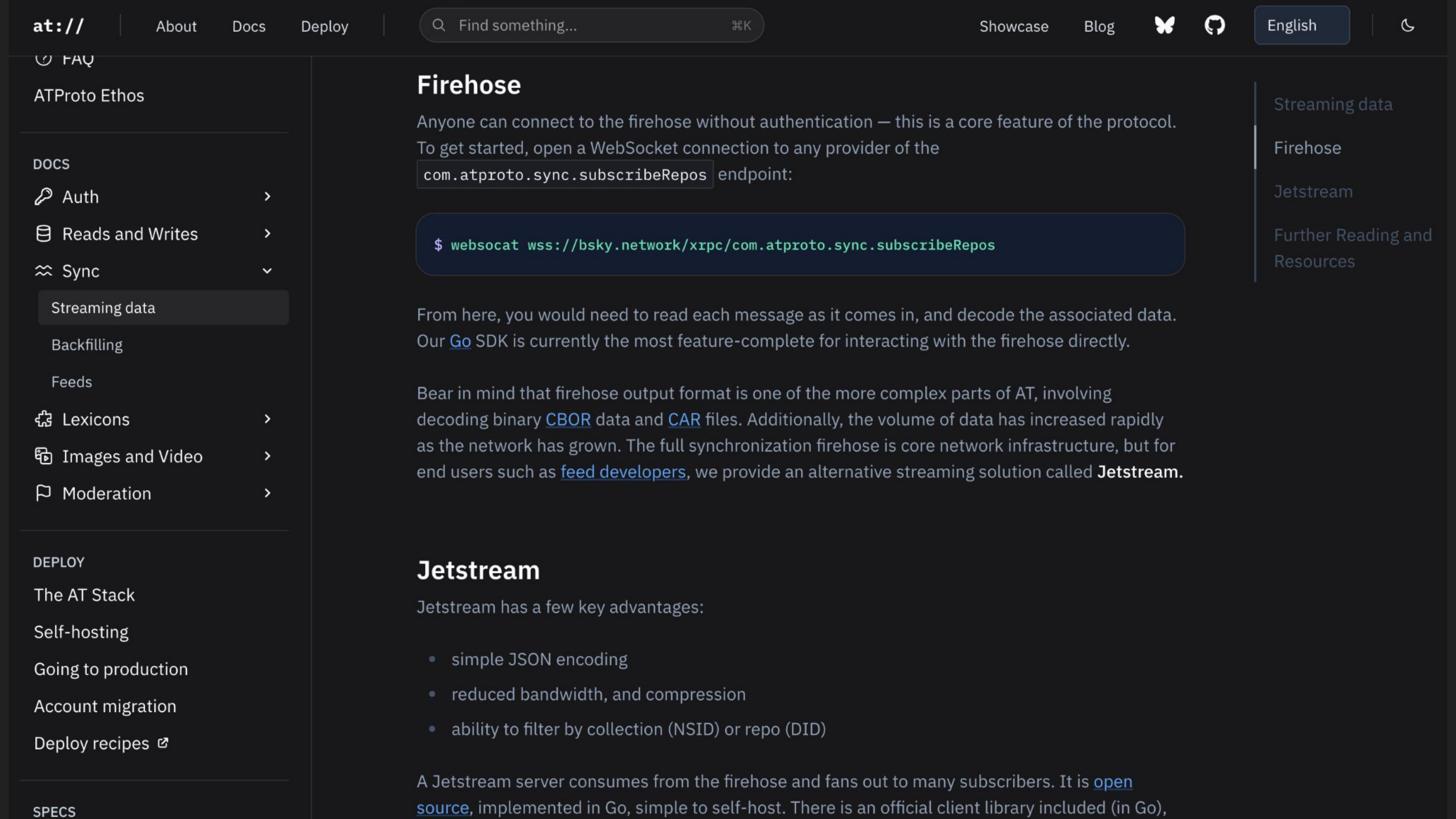
If you use homebrew, you can install directly:

```
brew install goat
```

Sipping the Firehose

Lexicon Authoring

Social Graphs



FAQ
ATProto Ethos
DOCS
Auth
Reads and Writes
Sync
Streaming data
Backfilling
Feeds
Lexicons
Images and Video
Moderation
DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes
SPECS

Firehose

Anyone can connect to the firehose without authentication — this is a core feature of the protocol. To get started, open a WebSocket connection to any provider of the `com.atproto.sync.subscribeRepos` endpoint:

```
$ websocat wss://bsky.network/xrpc/com.atproto.sync.subscribeRepos
```

From here, you would need to read each message as it comes in, and decode the associated data. Our Go SDK is currently the most feature-complete for interacting with the firehose directly.

Bear in mind that firehose output format is one of the more complex parts of AT, involving decoding binary CBOR data and CAR files. Additionally, the volume of data has increased rapidly as the network has grown. The full synchronization firehose is core network infrastructure, but for end users such as feed developers, we provide an alternative streaming solution called **Jetstream.**

Jetstream

Jetstream has a few key advantages:

- simple JSON encoding
- reduced bandwidth, and compression
- ability to filter by collection (NSID) or repo (DID)

A Jetstream server consumes from the firehose and fans out to many subscribers. It is open source, implemented in Go, simple to self-host. There is an official client library included (in Go),

Let’s try it!

```
$ brew install websocat
```

```
$ websocat \
'wss://jetstream2.us-east.bsky.network/subscribe?wantedCollections=app.bsky.feed.post'
```

FAQ
ATProto Ethos

DOCS
Auth ›
Reads and Writes ›
Sync ⌄
Streaming data
Backfilling
Feeds
Lexicons ›
Images and Video ›
Moderation ›

DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes

SPECS

Backfilling

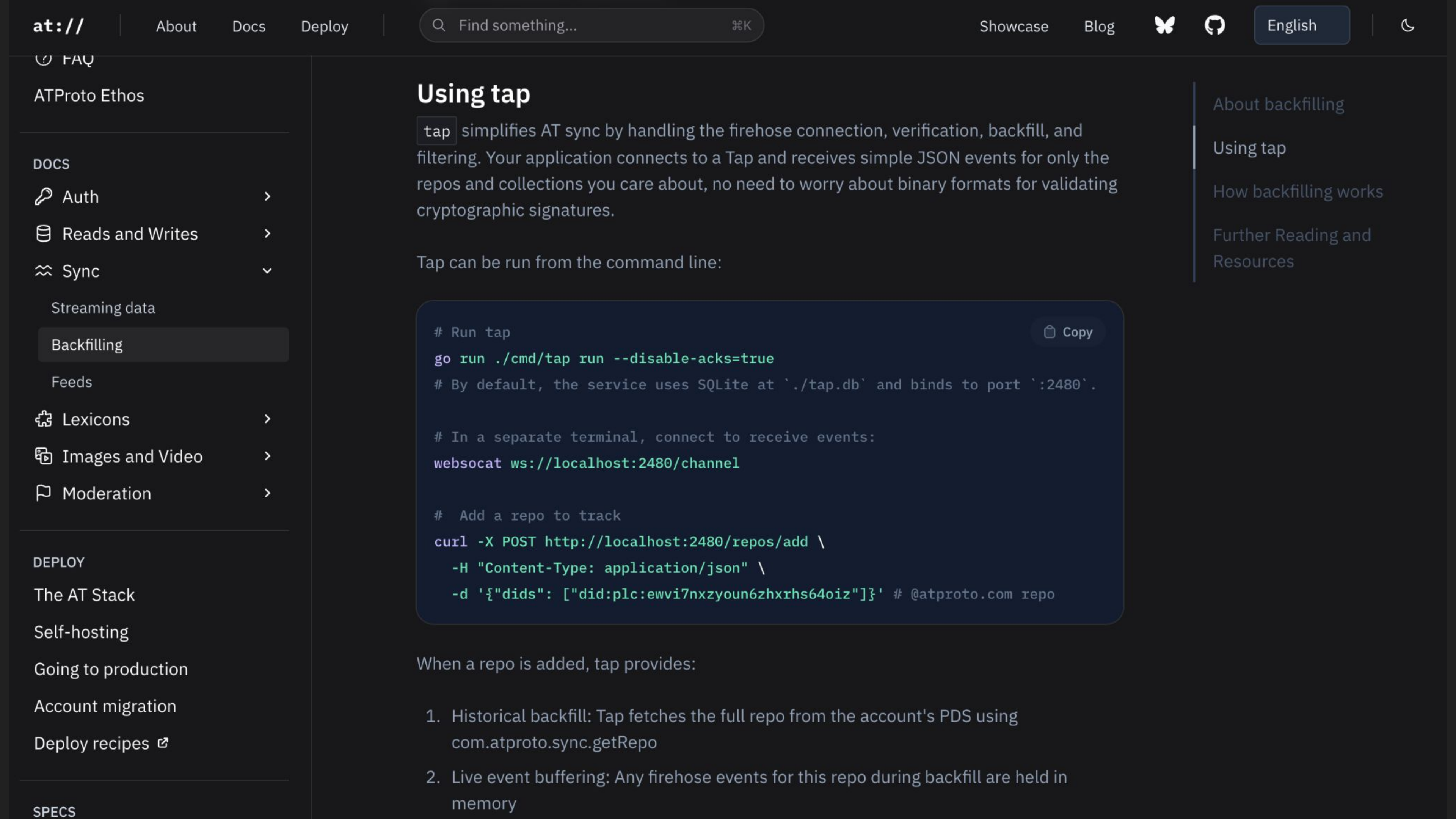
Replicating the network

About backfilling

Backfilling is the process of syncing all the data in the network from scratch. You may want to do this if you're running a service that requires a complete copy of the data in the network. This is not generally necessary for running feed generators, labelers, or bots, as most of the time they are fine handling live data off of the firehose. However, backfilling may be of interest if you want to perform large-scale data analysis.

For the entire network to be backfillable by third parties at all is a novel concept for AT. Other, monolithic social networks generally only offer an large-scale event stream (like our firehose) from the current date and time, making it difficult to perform longitudinal data analysis without additional data vendors. With the AT Protocol and adequate resources, you can *always* backfill the entire network on your own. This, in turn, benefits researchers and other forms of data analysis — if you can provision enough storage, you can have your own local copy of the entire Atmosphere.

When backfilling, you generally need to maintain 'up to date' replica of the data, which requires a cutover to streaming firehose data once the backfill is complete. We created `tap` to streamline this process.



FAQ
ATProto Ethos

DOCS
Auth
Reads and Writes
Sync
Streaming data
Backfilling
Feeds
Lexicons
Images and Video
Moderation

DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes

SPECS

Using tap

`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 can be run from the command line:

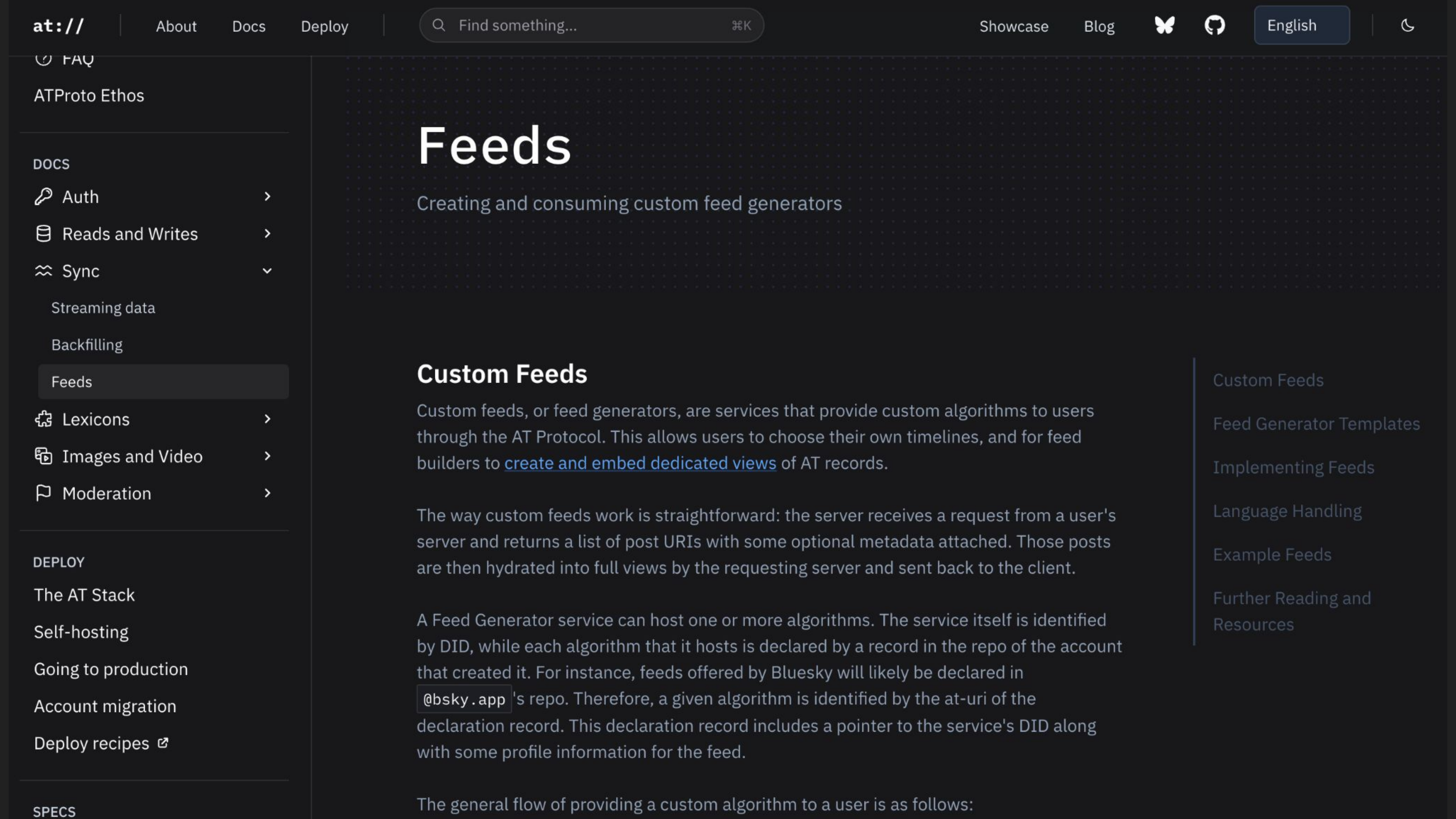
```
# Run tap
go run ./cmd/tap run --disable-acks=true
# By default, the service uses SQLite at `./tap.db` and binds to port `:2480`.

# In a separate terminal, connect to receive events:
websocat ws://localhost:2480/channel

#  Add a repo to track
curl -X POST http://localhost:2480/repos/add \
  -H "Content-Type: application/json" \
  -d '{"dids": ["did:plc:ewvi7nxzyoun6zhxrhs64oiz"]}' # @atproto.com repo
```

When a repo is added, tap provides:

1. Historical backfill: Tap fetches the full repo from the account's PDS using com.atproto.sync.getRepo
2. Live event buffering: Any firehose events for this repo during backfill are held in memory



Feeds

Creating and consuming custom feed generators

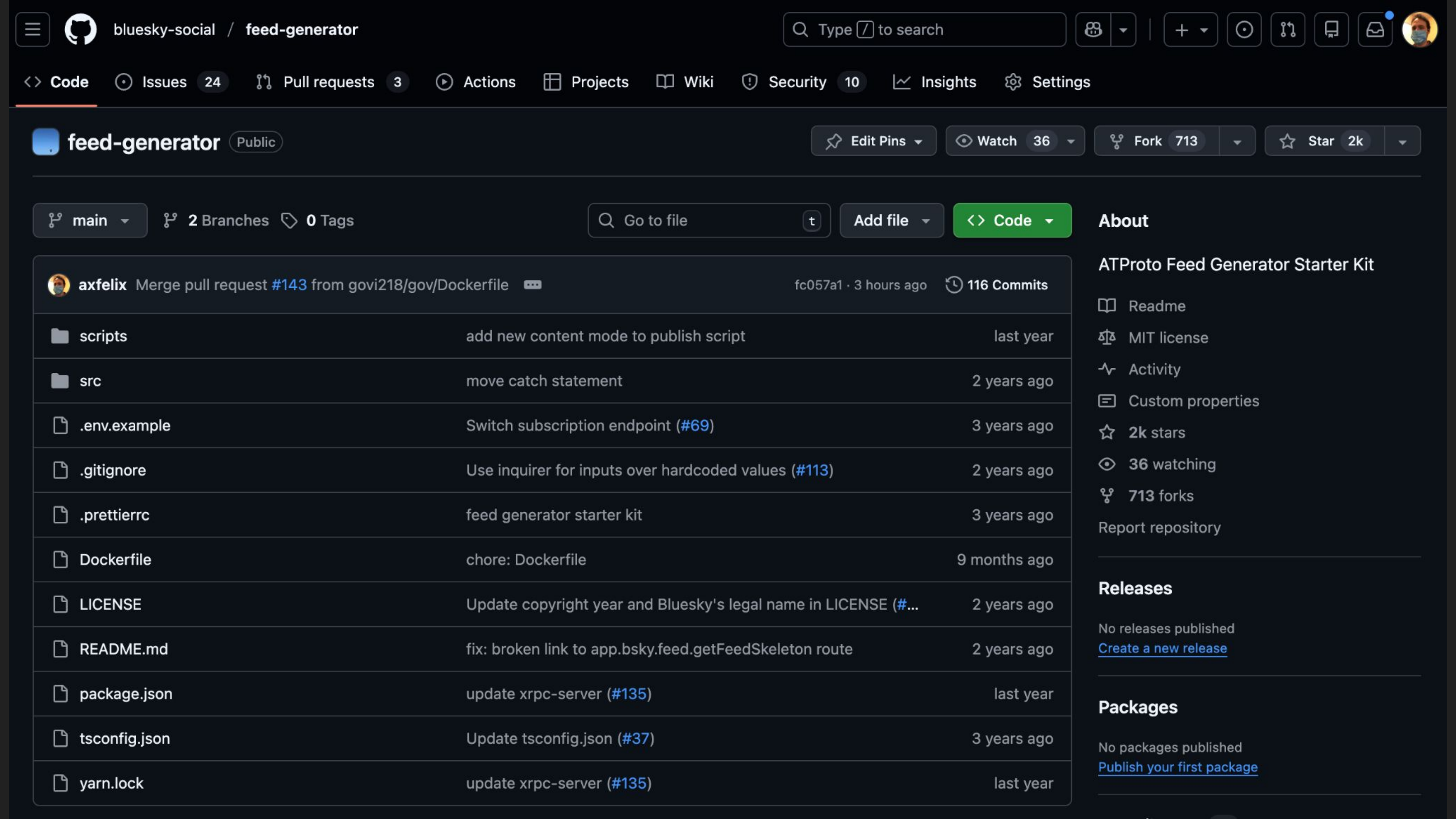
Custom Feeds

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, and for feed builders to create and embed dedicated views of AT records.

The way custom feeds work is straightforward: 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.

The general flow of providing a custom algorithm to a user is as follows:



feed-generator Public

2 Branches 0 Tags

axfelix Merge pull request #143 from govi218/gov/Dockerfile fc057a1 · 3 hours ago 116 Commits

scripts	add new content mode to publish script	last year
src	move catch statement	2 years ago
.env.example	Switch subscription endpoint (#69)	3 years ago
.gitignore	Use inquirer for inputs over hardcoded values (#113)	2 years ago
.prettierrc	feed generator starter kit	3 years ago
Dockerfile	chore: Dockerfile	9 months ago
LICENSE	Update copyright year and Bluesky's legal name in LICENSE (#...	2 years ago
README.md	fix: broken link to app.bsky.feed.getFeedSkeleton route	2 years ago
package.json	update xrpc-server (#135)	last year
tsconfig.json	Update tsconfig.json (#37)	3 years ago
yarn.lock	update xrpc-server (#135)	last year

About

ATProto Feed Generator Starter Kit

- Readme
- MIT license
- Activity
- Custom properties
- 2k stars
- 36 watching
- 713 forks

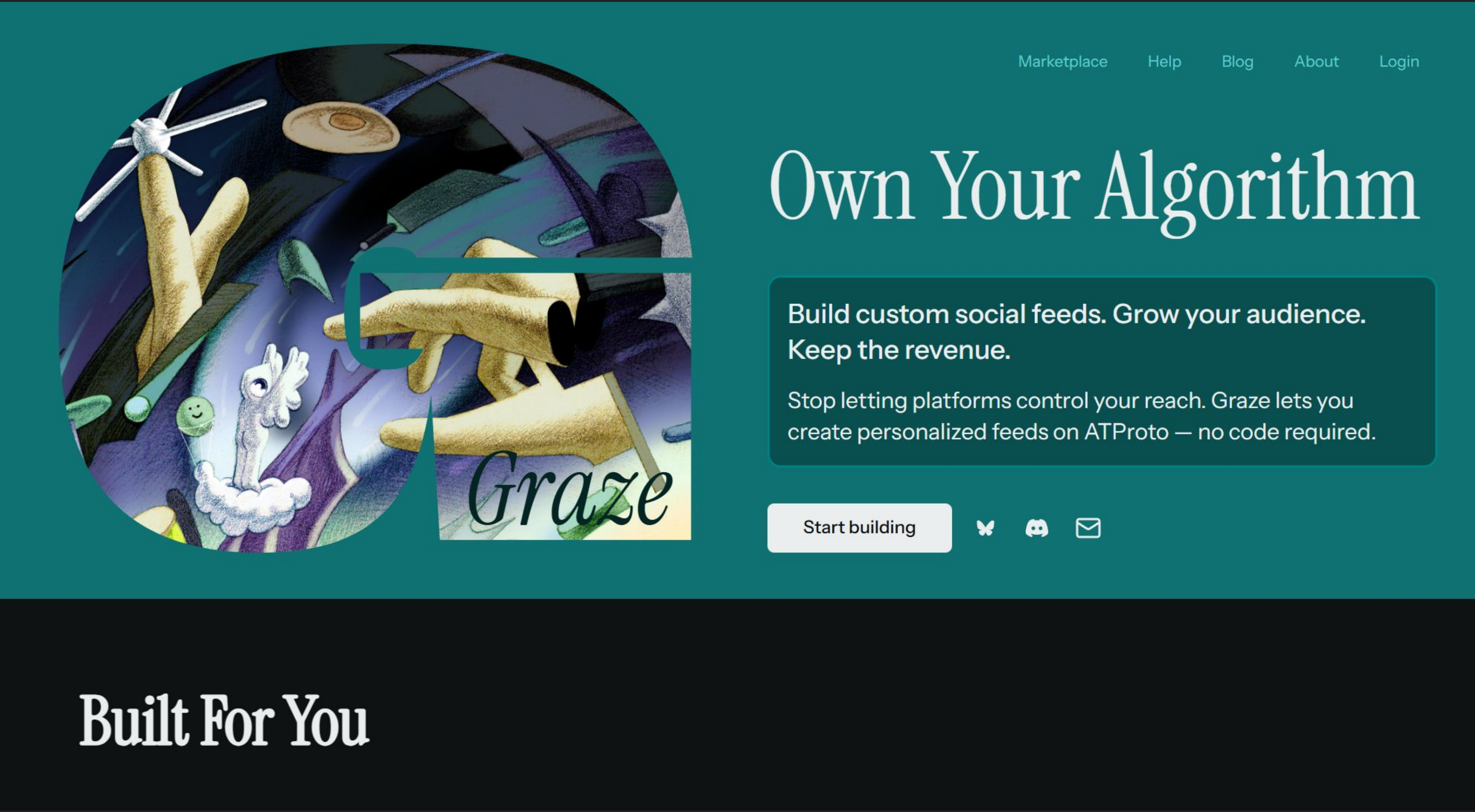
Report repository

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

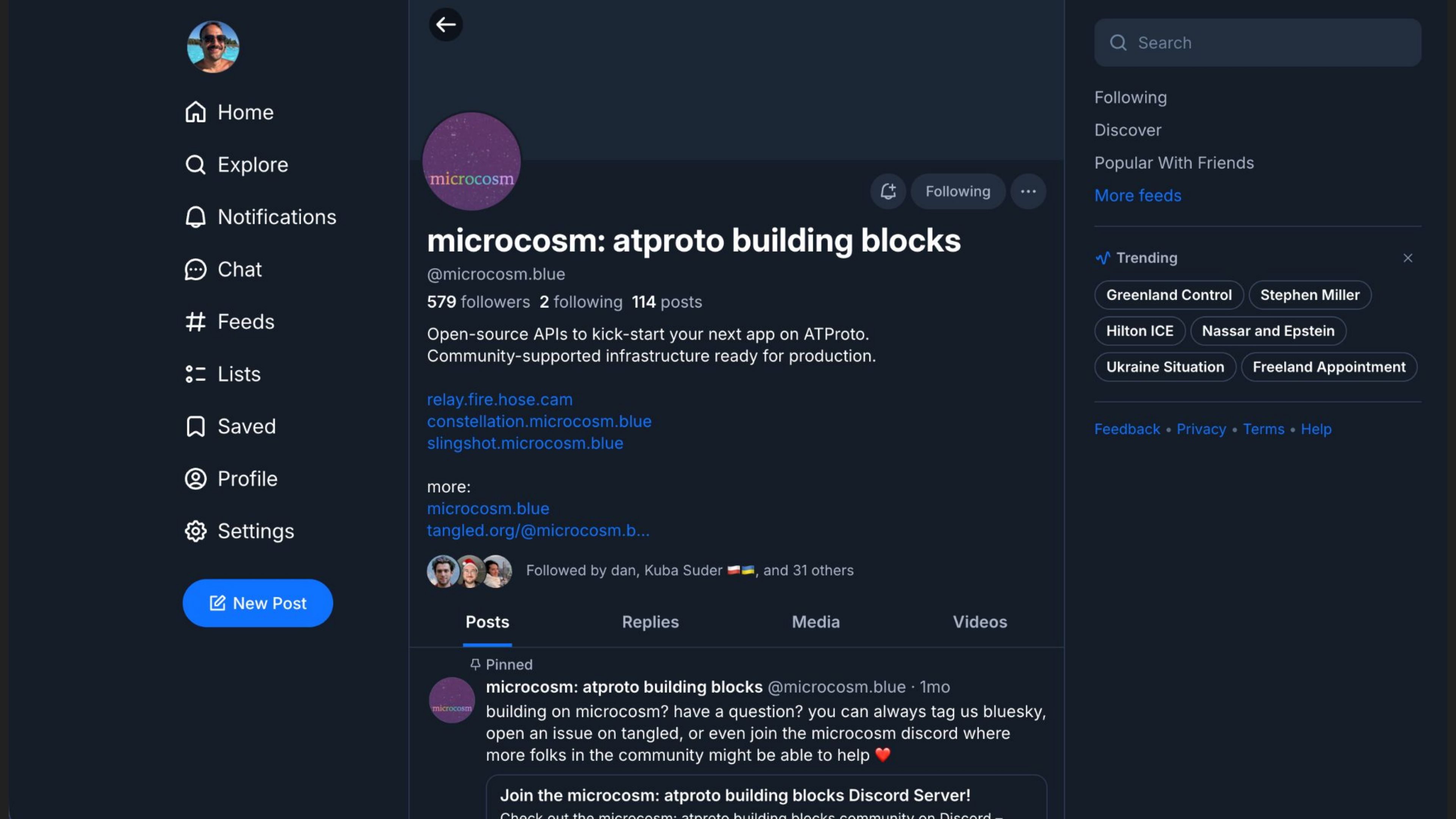


Own Your Algorithm

Build custom social feeds. Grow your audience. Keep the revenue.

Stop letting platforms control your reach. Graze lets you create personalized feeds on ATProto — no code required.

Built For You



microcosm: atproto building blocks

@microcosm.blue

579 followers 2 following 114 posts

Open-source APIs to kick-start your next app on ATProto.
Community-supported infrastructure ready for production.

relay.fire.hose.cam
constellation.microcosm.blue
slingshot.microcosm.blue

more:
microcosm.blue
tangled.org/@microcosm.b...

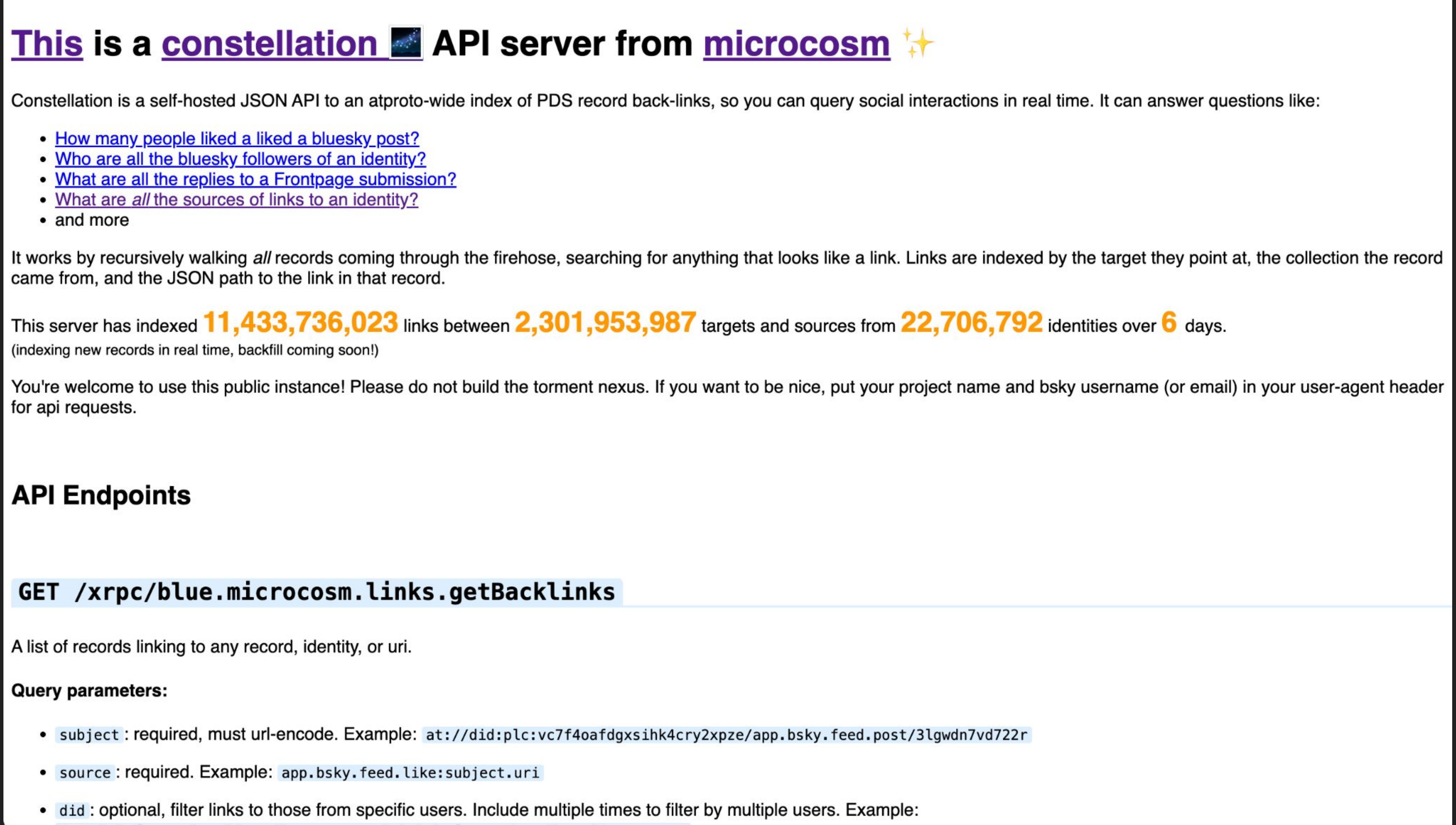
Followed by dan, Kuba Suder 🇵🇱🇺🇦, and 31 others

microcosm: atproto building blocks @microcosm.blue · 1mo

building on microcosm? have a question? you can always tag us bluesky, open an issue on tangled, or even join the microcosm discord where more folks in the community might be able to help ❤️

Join the microcosm: atproto building blocks Discord Server!

Check out the microcosm: atproto building blocks community on Discord –



This is a constellation 🌌 API server from microcosm ✨

Constellation is a self-hosted JSON API to an atproto-wide index of PDS record back-links, so you can query social interactions in real time. It can answer questions like:

- How many people liked a liked a bluesky post?
- Who are all the bluesky followers of an identity?
- What are all the replies to a Frontpage submission?
- What are *all* the sources of links to an identity?
- and more

It works by recursively walking *all* records coming through the firehose, searching for anything that looks like a link. Links are indexed by the target they point at, the collection the record came from, and the JSON path to the link in that record.

This server has indexed **11,433,736,023** links between **2,301,953,987** targets and sources from **22,706,792** identities over **6** days.
(indexing new records in real time, backfill coming soon!)

You're welcome to use this public instance! Please do not build the torment nexus. If you want to be nice, put your project name and bsky username (or email) in your user-agent header for api requests.

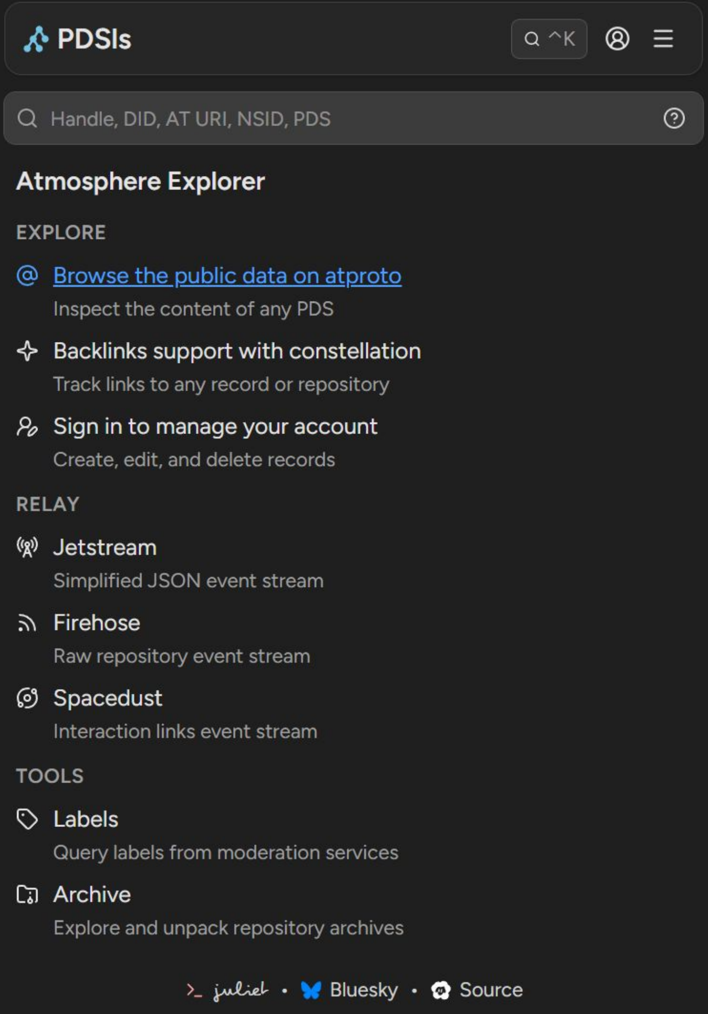
API Endpoints

`GET /xrpc/blue.microcosm.links.getBacklinks`

A list of records linking to any record, identity, or uri.

Query parameters:

- `subject`: required, must url-encode. Example: `at://did:plc:vc7f4oafdgxsihk4cry2xpze/app.bsky.feed.post/3lgwdn7vd722r`
- `source`: required. Example: `app.bsky.feed.like:subject.uri`
- `did`: optional, filter links to those from specific users. Include multiple times to filter by multiple users. Example:



Atmosphere Explorer

EXPLORE

@ Browse the public data on atproto
Inspect the content of any PDS

Backlinks support with constellation
Track links to any record or repository

Sign in to manage your account
Create, edit, and delete records

RELAY

Jetstream
Simplified JSON event stream

Firehose
Raw repository event stream

Spacedust
Interaction links event stream

TOOLS

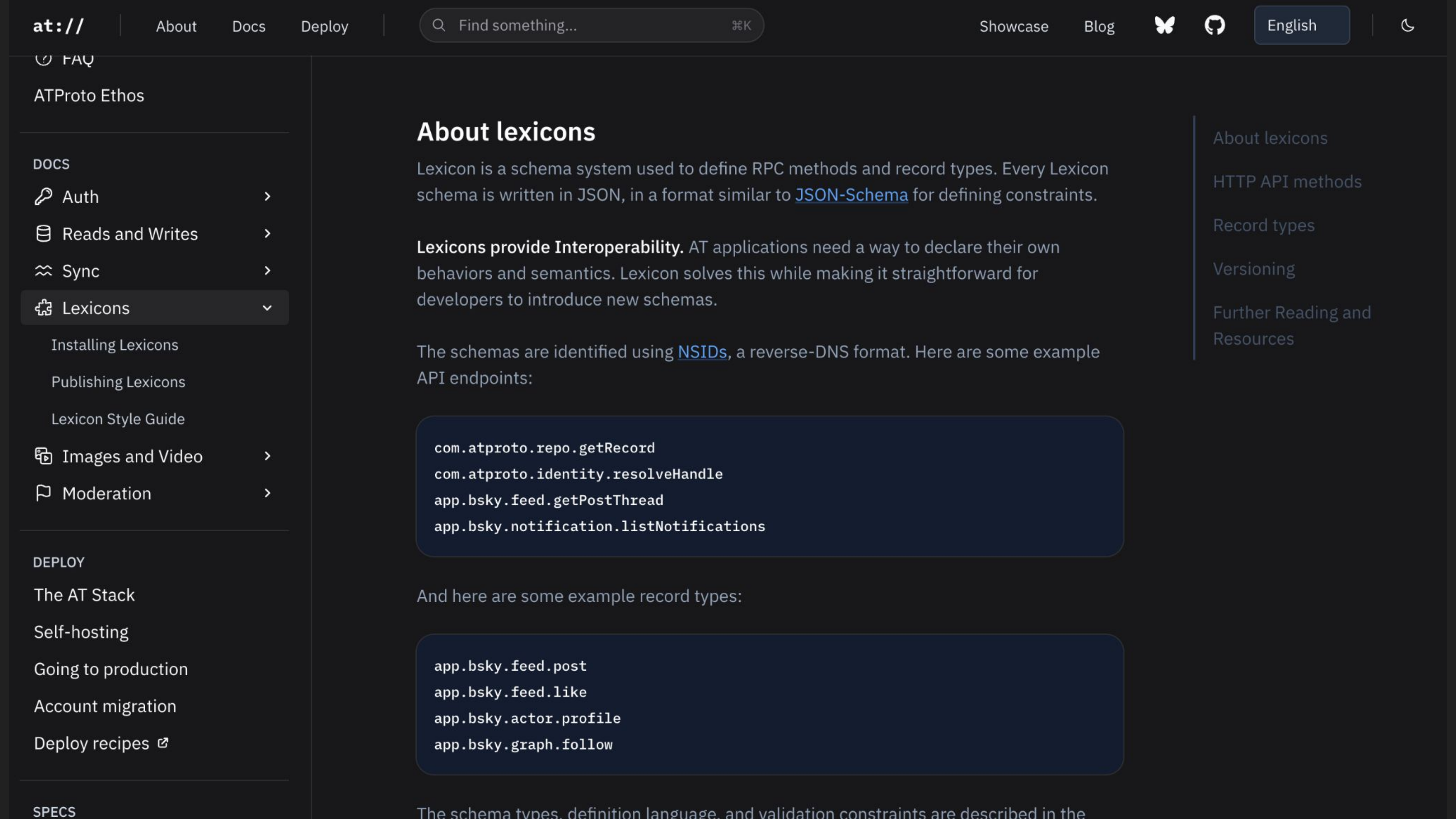
Labels
Query labels from moderation services

Archive
Explore and unpack repository archives

Sipping the Firehose

Lexicon Authoring

Social Graphs



FAQ
ATProto Ethos
DOCS
Auth
Reads and Writes
Sync
Lexicons
Installing Lexicons
Publishing Lexicons
Lexicon Style Guide
Images and Video
Moderation
DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes
SPECS

About lexicons

Lexicon is a schema system used to define RPC methods and record types. Every Lexicon schema is written in JSON, in a format similar to JSON-Schema for defining constraints.

Lexicons provide Interoperability. AT applications need a way to declare their own behaviors and semantics. Lexicon solves this while making it straightforward for developers to introduce new schemas.

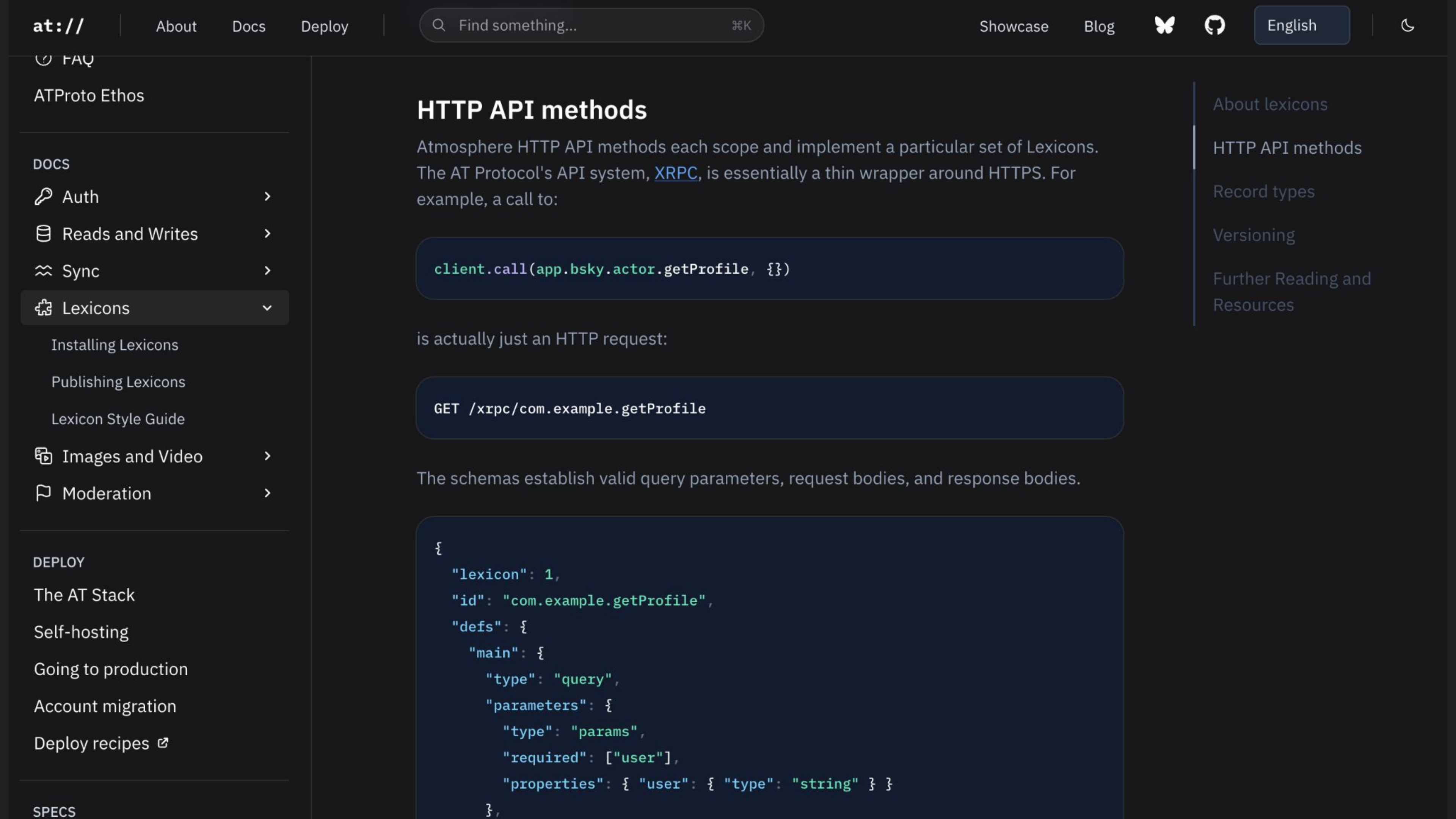
The schemas are identified using NSIDs, a reverse-DNS format. Here are some example API endpoints:

```
com.atproto.repo.getRecord
com.atproto.identity.resolveHandle
app.bsky.feed.getPostThread
app.bsky.notification.listNotifications
```

And here are some example record types:

```
app.bsky.feed.post
app.bsky.feed.like
app.bsky.actor.profile
app.bsky.graph.follow
```

The schema types, definition language, and validation constraints are described in the



FAQ
ATProto Ethos
DOCS
Auth
Reads and Writes
Sync
Lexicons
Installing Lexicons
Publishing Lexicons
Lexicon Style Guide
Images and Video
Moderation
DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes
SPECS

HTTP API methods

Atmosphere HTTP API methods each scope and implement a particular set of Lexicons. The AT Protocol's API system, XRPC, is essentially a thin wrapper around HTTPS. For example, a call to:

```
client.call(app.bsky.actor.getProfile, {})
```

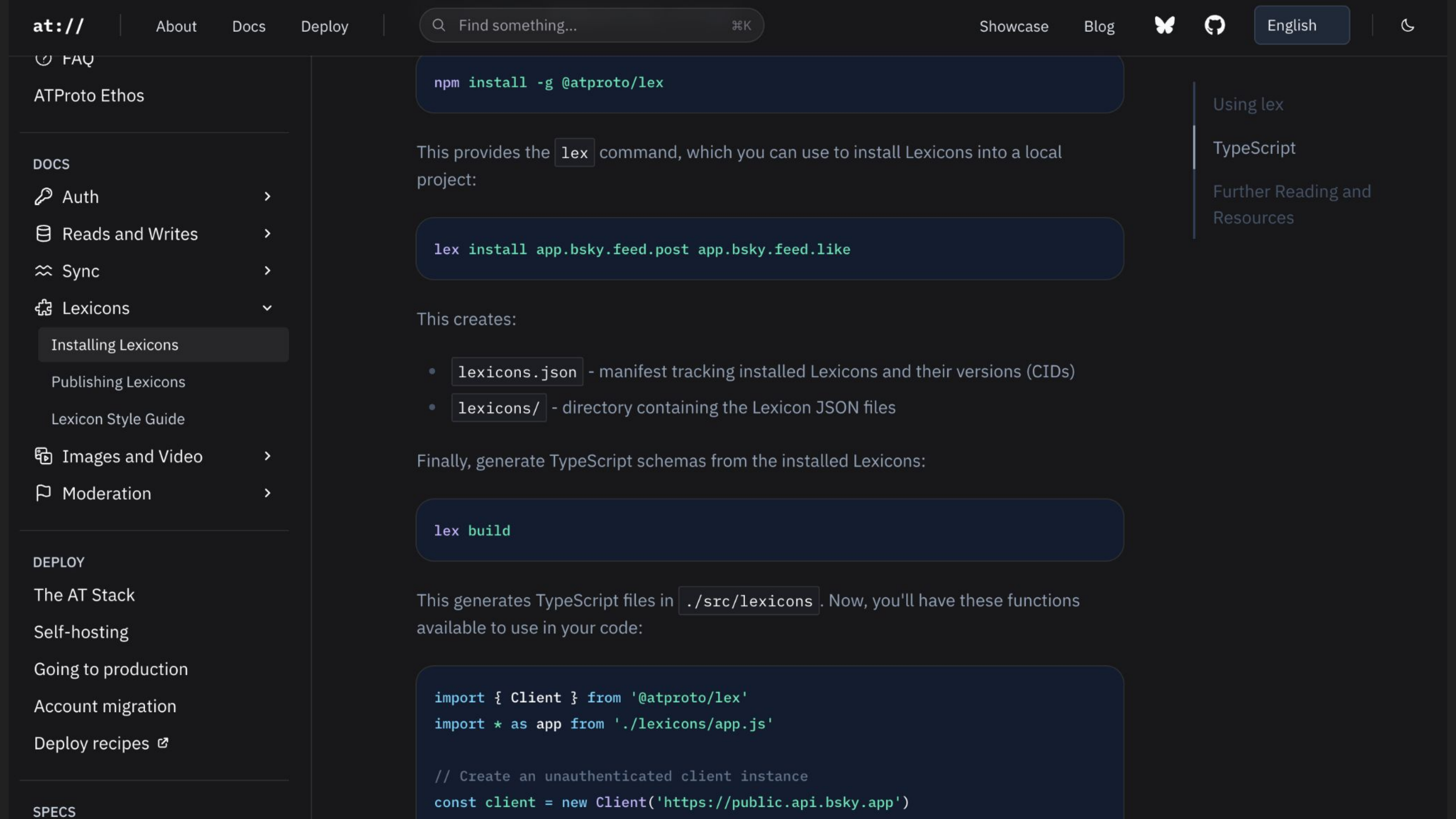
is actually just an HTTP request:

```
GET /xrpc/com.example.getProfile
```

The schemas establish valid query parameters, request bodies, and response bodies.

```
{
  "lexicon": 1,
  "id": "com.example.getProfile",
  "defs": {
    "main": {
      "type": "query",
      "parameters": {
        "type": "params",
        "required": ["user"],
        "properties": { "user": { "type": "string" } }
      },
```

About lexicons
HTTP API methods
Record types
Versioning
Further Reading and Resources



FAQ
ATProto Ethos

DOCS
Auth
Reads and Writes
Sync
Lexicons
Installing Lexicons
Publishing Lexicons
Lexicon Style Guide
Images and Video
Moderation

DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes

SPECS

```
npm install -g @atproto/lex
```

This provides the `lex` command, which you can use to install Lexicons into a local project:

```
lex install app.bsky.feed.post app.bsky.feed.like
```

This creates:

- `lexicons.json` - manifest tracking installed Lexicons and their versions (CIDs)
- `lexicons/` - directory containing the Lexicon JSON files

Finally, generate TypeScript schemas from the installed Lexicons:

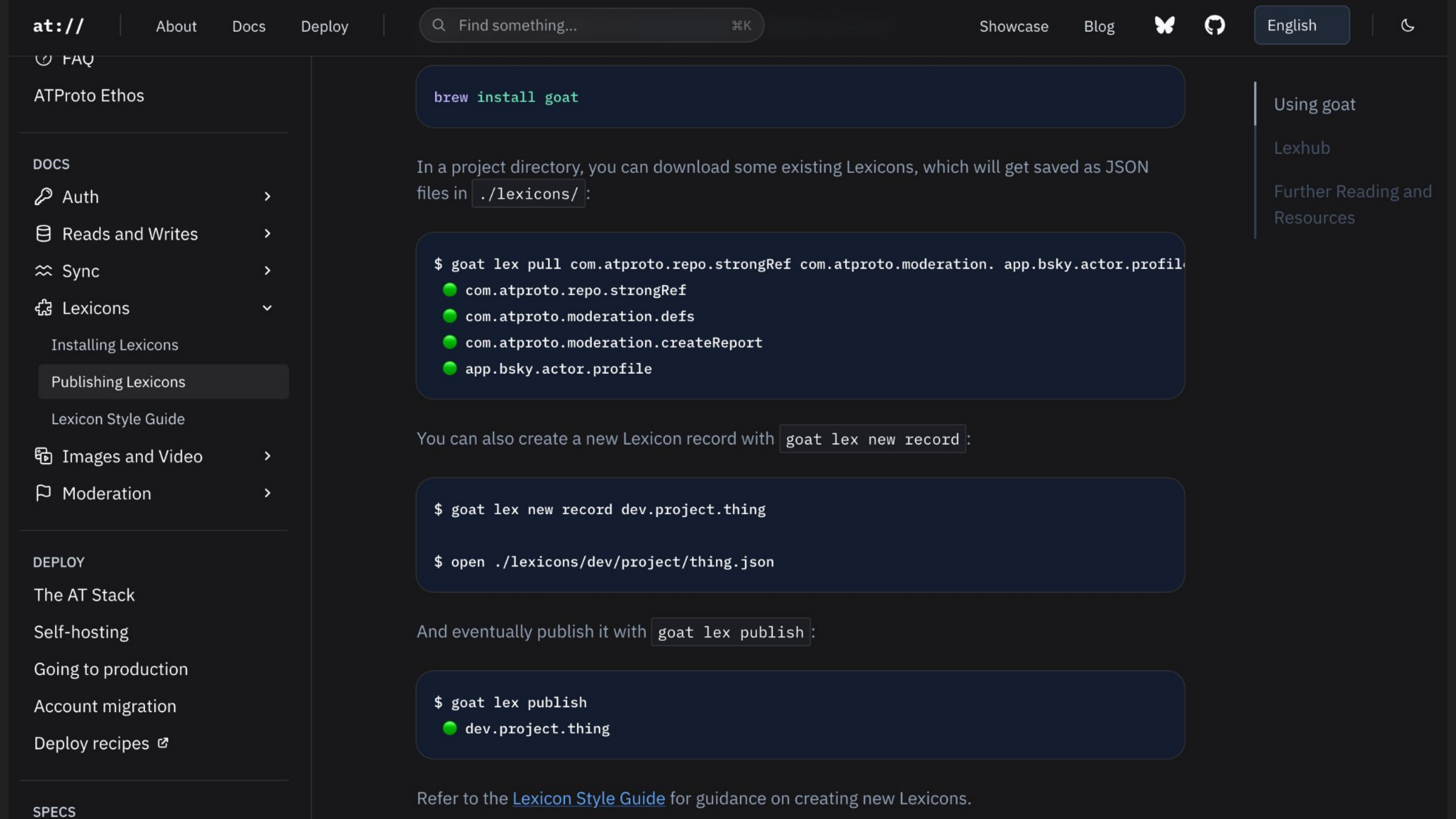
```
lex build
```

This generates TypeScript files in `./src/lexicons`. Now, you'll have these functions available to use in your code:

```
import { Client } from '@atproto/lex'
import * as app from './lexicons/app.js'

// Create an unauthenticated client instance
const client = new Client('https://public.api.bsky.app')
```

Using lex
TypeScript
Further Reading and Resources



FAQ
ATProto Ethos

DOCS
Auth
Reads and Writes
Sync
Lexicons
Installing Lexicons
Publishing Lexicons
Lexicon Style Guide
Images and Video
Moderation

DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes

SPECS

```
brew install goat
```

In a project directory, you can download some existing Lexicons, which will get saved as JSON files in `./lexicons/`:

```
$ goat lex pull com.atproto.repo.strongRef com.atproto.moderation. app.bsky.actor.profil
 🟢 com.atproto.repo.strongRef
 🟢 com.atproto.moderation.defs
 🟢 com.atproto.moderation.createReport
 🟢 app.bsky.actor.profile
```

You can also create a new Lexicon record with `goat lex new record`:

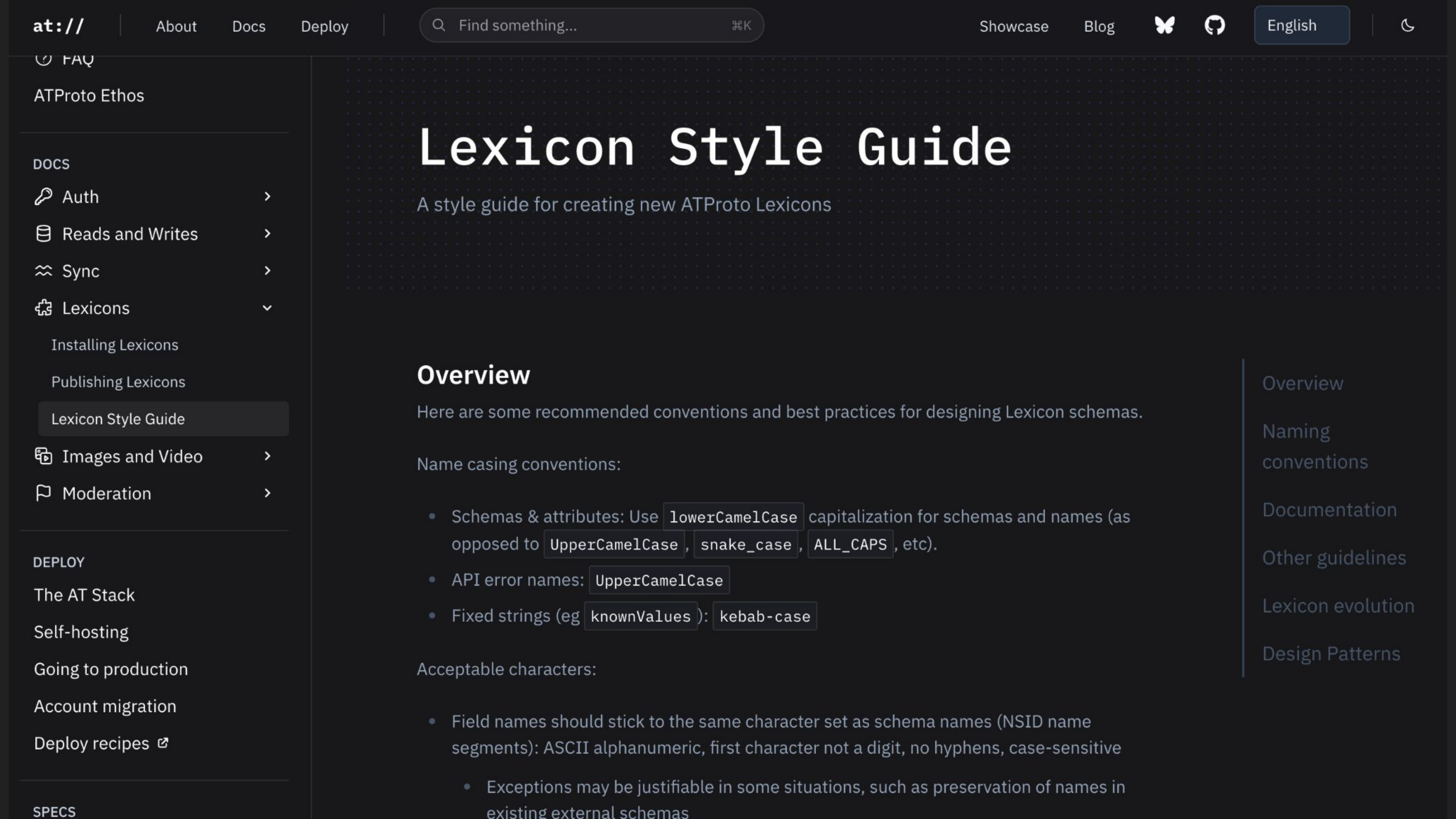
```
$ goat lex new record dev.project.thing


$ open ./lexicons/dev/project/thing.json
```

And eventually publish it with `goat lex publish`:

```
$ goat lex publish
 🟢 dev.project.thing
```

Refer to the Lexicon Style Guide for guidance on creating new Lexicons.



FAQ
ATProto Ethos

DOCS
Auth
Reads and Writes
Sync
Lexicons
Installing Lexicons
Publishing Lexicons
Lexicon Style Guide
Images and Video
Moderation

DEPLOY
The AT Stack
Self-hosting
Going to production
Account migration
Deploy recipes

SPECS

Lexicon Style Guide

A style guide for creating new ATProto Lexicons

Overview

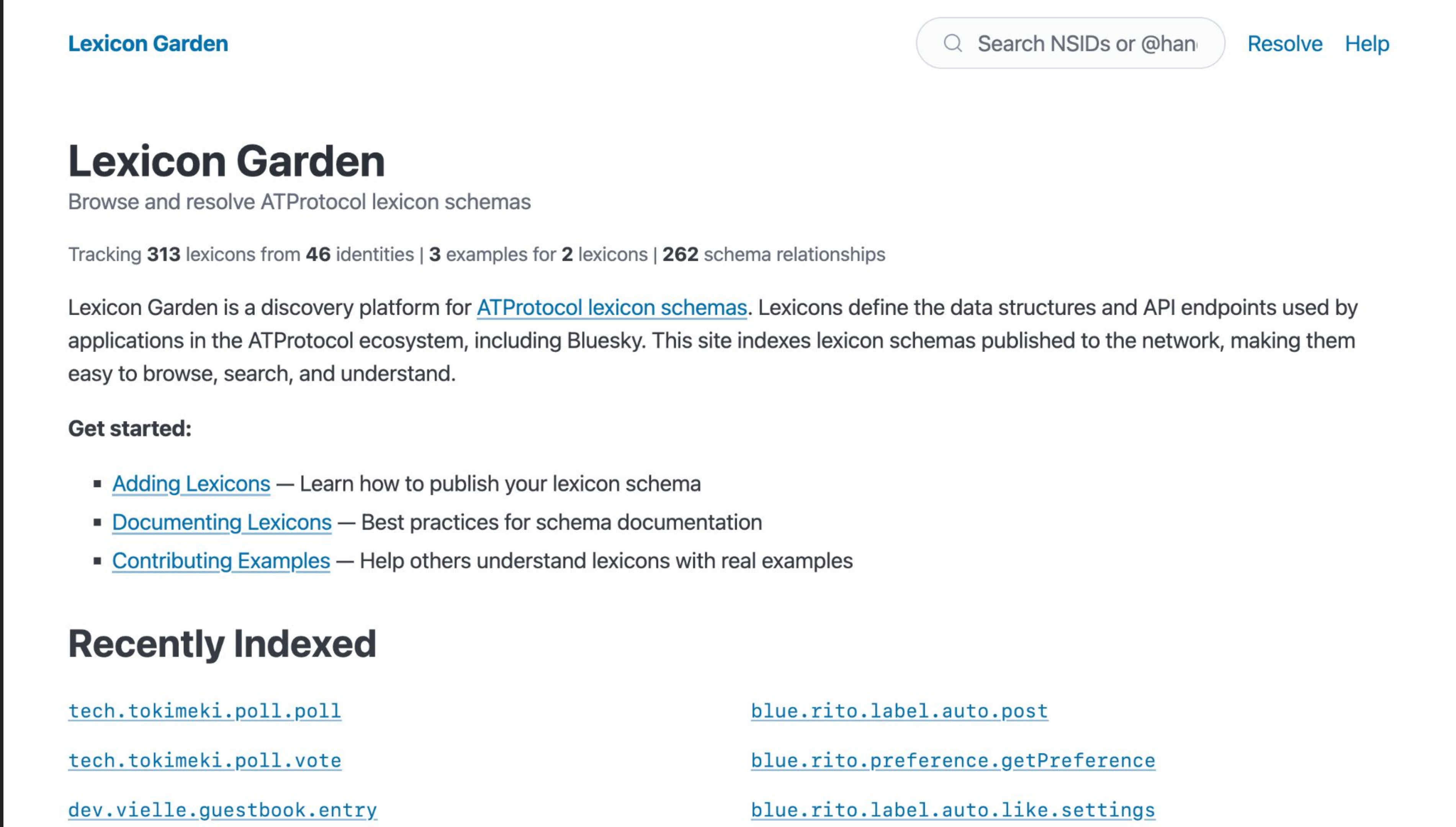
Here are some recommended conventions and best practices for designing Lexicon schemas.

Name casing conventions:

- Schemas & attributes: Use `lowerCamelCase` capitalization for schemas and names (as opposed to `UpperCamelCase`, `snake_case`, `ALL_CAPS`, etc).
- API error names: `UpperCamelCase`
- Fixed strings (eg `knownValues`): `kebab-case`

Acceptable characters:

- Field names should stick to the same character set as schema names (NSID name segments): ASCII alphanumeric, first character not a digit, no hyphens, case-sensitive
  - Exceptions may be justifiable in some situations, such as preservation of names in existing external schemas



Lexicon Garden

Browse and resolve ATProtocol lexicon schemas

Tracking **313** lexicons from **46** identities | **3** examples for **2** lexicons | **262** schema relationships

Lexicon Garden is a discovery platform for ATProtocol lexicon schemas. Lexicons define the data structures and API endpoints used by applications in the ATProtocol ecosystem, including Bluesky. This site indexes lexicon schemas published to the network, making them easy to browse, search, and understand.

Get started:

- Adding Lexicons — Learn how to publish your lexicon schema
- Documenting Lexicons — Best practices for schema documentation
- Contributing Examples — Help others understand lexicons with real examples

Recently Indexed

`tech.tokimeki.poll.poll`

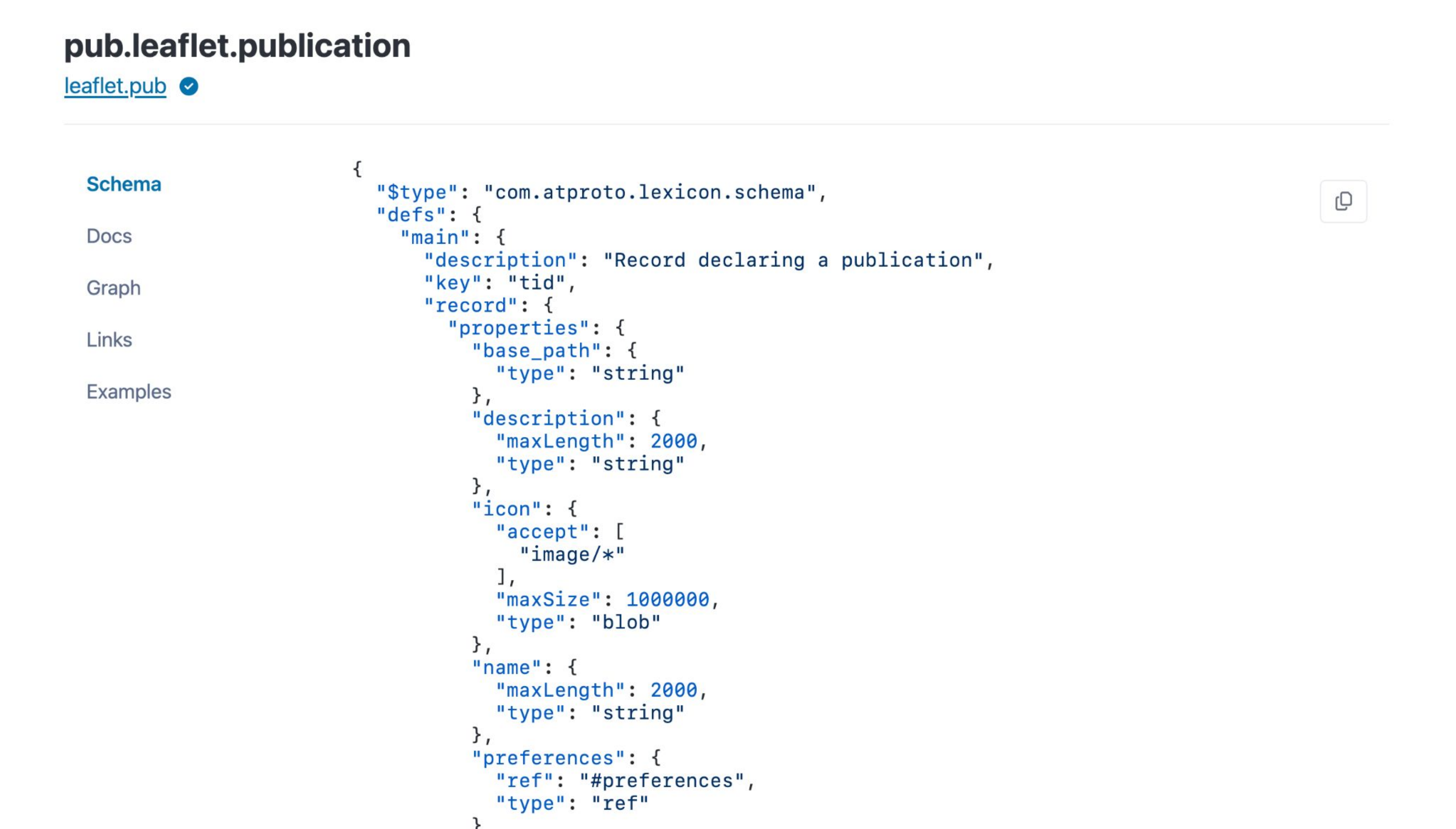
`tech.tokimeki.poll.vote`

`dev.vielle.guestbook.entry`

`blue.rito.label.auto.post`

`blue.rito.preference.getPreference`

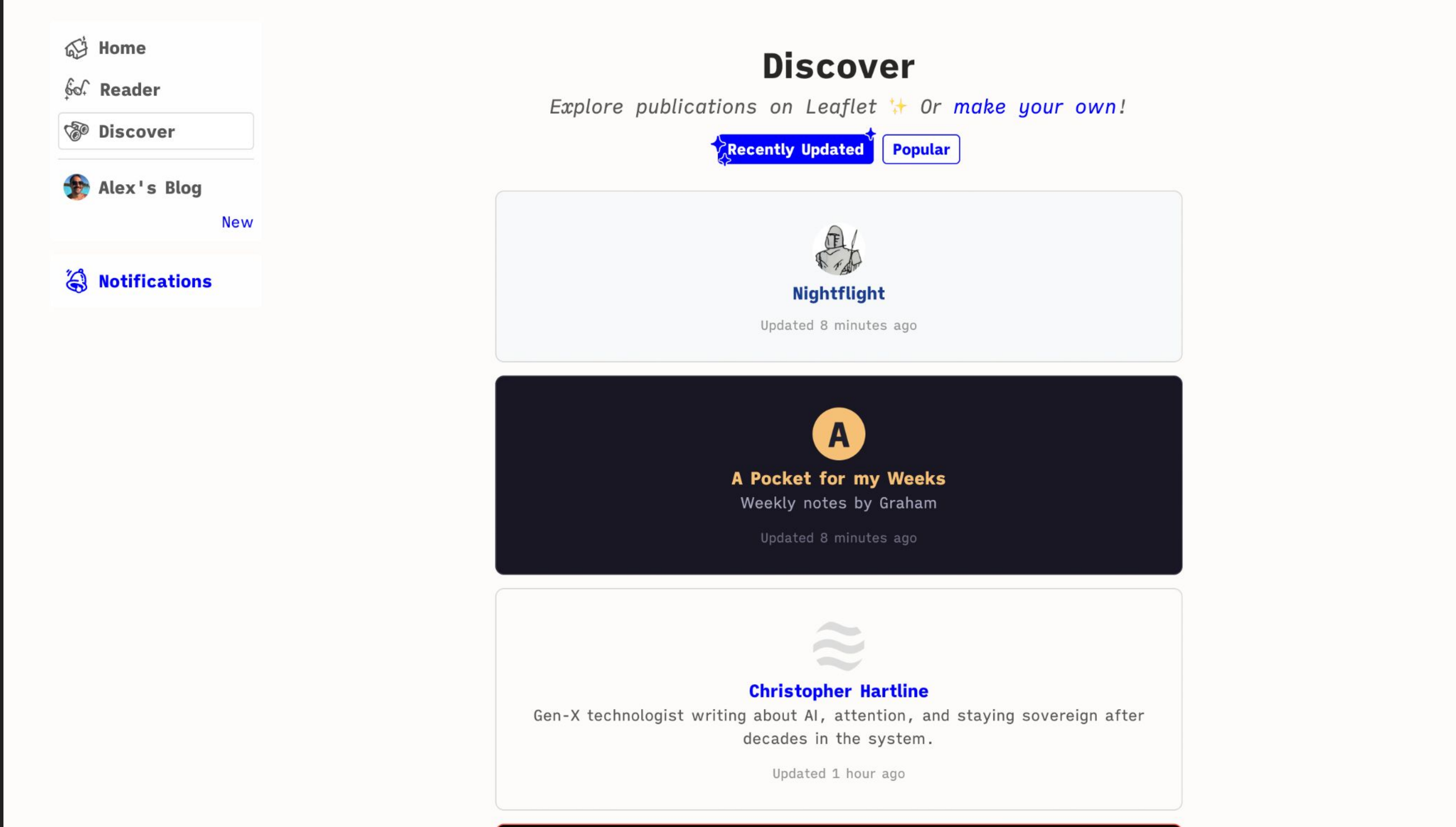
`blue.rito.label.auto.like.settings`



pub.leaflet.publication

leaflet.pub

```
{
  "$type": "com.atproto.lexicon.schema",
  "defs": {
    "main": {
      "description": "Record declaring a publication",
      "key": "tid",
      "record": {
        "properties": {
          "base_path": {
            "type": "string"
          },
          "description": {
            "maxLength": 2000,
            "type": "string"
          },
          "icon": {
            "accept": [
              "image/*"
            ],
            "maxSize": 1000000,
            "type": "blob"
          },
          "name": {
            "maxLength": 2000,
            "type": "string"
          },
          "preferences": {
            "ref": "#preferences",
            "type": "ref"
          }
```



Discover

Explore publications on Leaflet ✨ Or make your own!

Nightflight

Updated 8 minutes ago

A Pocket for my Weeks

Weekly notes by Graham

Updated 8 minutes ago

Christopher Hartline

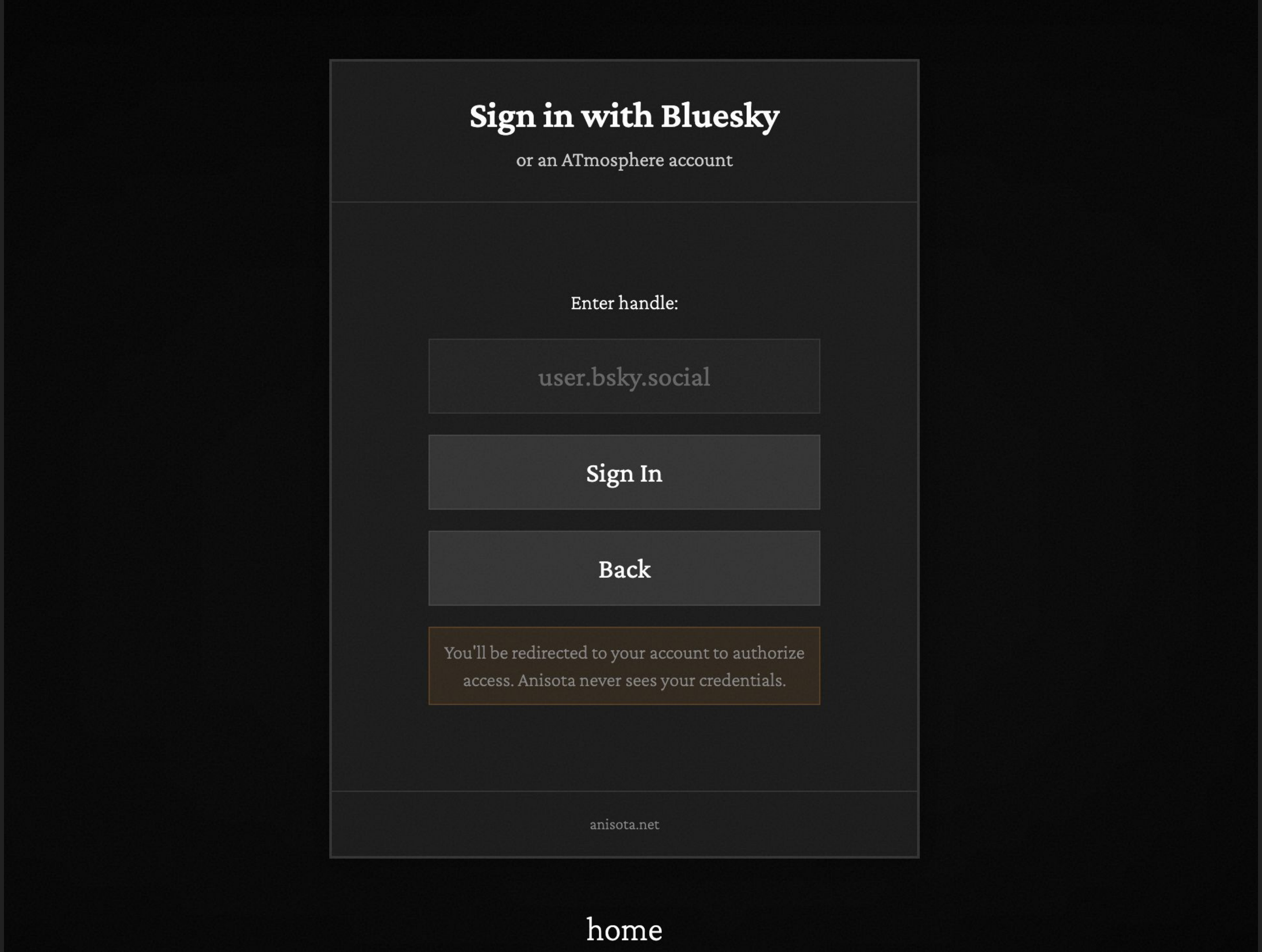
Gen-X technologist writing about AI, attention, and staying sovereign after decades in the system.

Updated 1 hour ago

Week 9: Euro Truck Simulator

I can tell DevRel season is starting back up again, because I counted the number of flights I currently have booked in the next 6 months, and came back with **18**. Granted, some of these are vacations, and I'm a bit of an over-preparer when it comes to vacations (PLEASE do not start me on a conversation about min-maxing Chase points, I will embarrass us both), but a big part of that is the job! I'm usually on the road a little more than once a month from January-May, and while I think that's a mostly sustainable cadence (and appreciate getting face time with AT heads from all across the world), it does come on pretty strong every year.

Anyway, this is all a roundabout way of complaining/bragging that I've so far had 2 talks accepted since starting in this role, which is a nice little streak! One isn't for many months from now — it'll be at the first North American event of WeAreDevelopers, which is co-presented with Docker, and is I think intended to be a successor to DockerCon. WeAreDevelopers' Berlin events have always been good, so I'm looking forward to this when it happens next September. The other talk is barely more than a month away — at FOSDEM's Decentralised Communication developer room in Brussels at the end of January! This will be basically my first proper AT Protocol talk, and figuring out what points that I personally like to hit is one of the most fun parts of a new job. I know how I *write* about this project, but that's never quite the same thing as how to *talk* about a



Sign in with Bluesky

or an ATmosphere account

Enter handle:

user.bsky.social

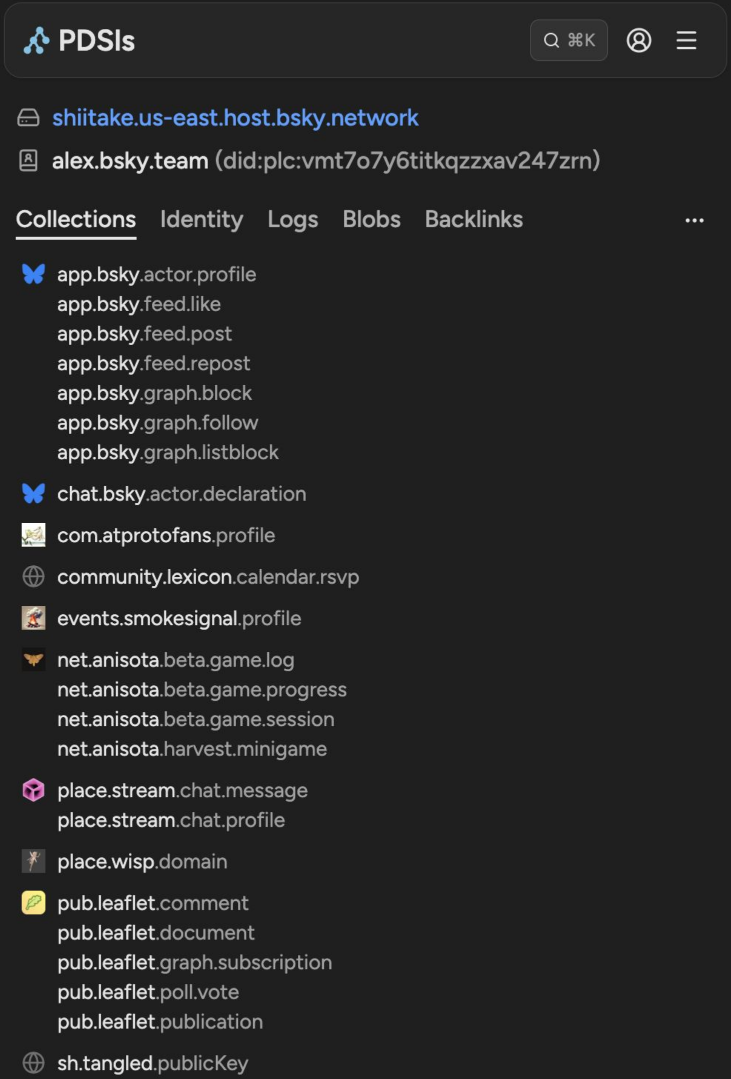
Sign In

Back

You'll be redirected to your account to authorize access. Anisota never sees your credentials.

anisota.net

home



shiitake.us-east.host.bsky.network

alex.bsky.team (did:plc:vmt7o7y6titkqzzxav247zrn)

app.bsky.actor.profile
app.bsky.feed.like
app.bsky.feed.post
app.bsky.feed.repost
app.bsky.graph.block
app.bsky.graph.follow
app.bsky.graph.listblock

chat.bsky.actor.declaration

com.atprotofans.profile

community.lexicon.calendar.rsvp

events.smokesignal.profile

net.anisota.beta.game.log
net.anisota.beta.game.progress
net.anisota.beta.game.session
net.anisota.harvest.minigame

place.stream.chat.message
place.stream.chat.profile

place.wisp.domain

pub.leaflet.comment
pub.leaflet.document
pub.leaflet.graph.subscription
pub.leaflet.poll.vote
pub.leaflet.publication

sh.tangled.publicKey

Sipping the Firehose

Lexicon Authoring

Social Graphs

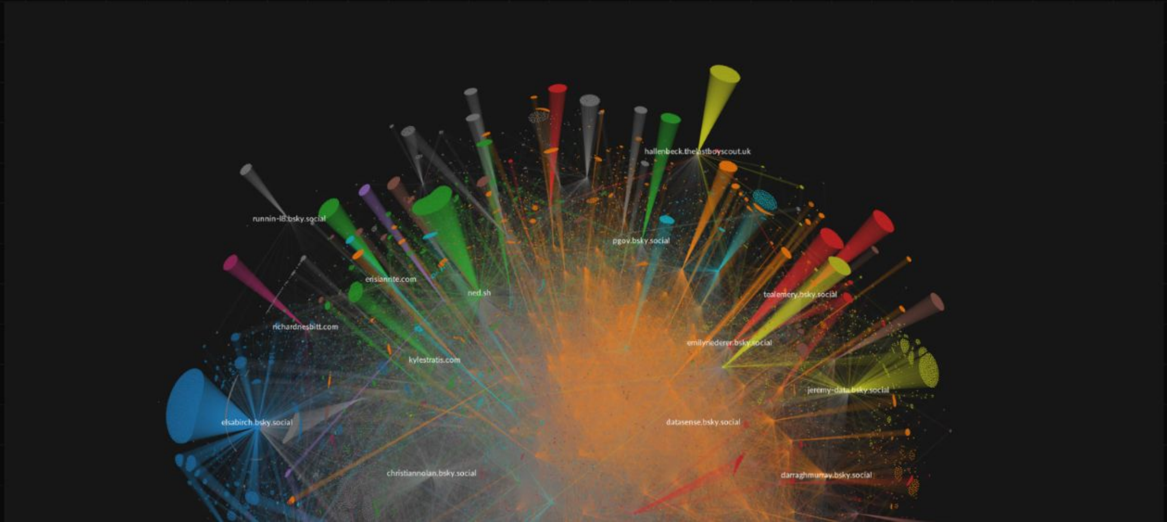
Exploring AT Protocol with Python

Thu Nov 14 2024

In the last few weeks, there has been a lot of activity on Bluesky. Bluesky is a social network built on open standards. Specifically, it is built on top of the AT Protocol. Most (if not all) data is exposed via XRPC endpoints.

This post is a quick glance at the AT Protocol and its Python SDK. To do that we'll create a script to download all the `#dataBS` posters and create a graph with the connections around that community.

You can explore the final interactive graph online!



Getting the social graph

Each post has a `post.author.handle` property that can be used in our next XRPC call to `app.bsky.graph.getFollows` endpoint. This endpoint returns all the actors that the given actor follows (also using the `cursor` property to paginate through the results).

We can write a quick function to get all the follows for a given actor:

```
def get_all_follows(author):
    cursor = None
    follows = []
    while True:
        fetched = client.app.bsky.graph.get_follows(params={'actor': author,
'cursor': cursor})
        follows = follows + fetched.follows
        if not fetched.cursor:
            break
        cursor = fetched.cursor
    return follows
```

And then we can use that function to get all the follows for all the `#dataBS` authors. Since we don’t know how big the graph is, we’ll be dumping the results into a CSV file.

Before writing the results, let’s get the unique authors:

overreacted

by

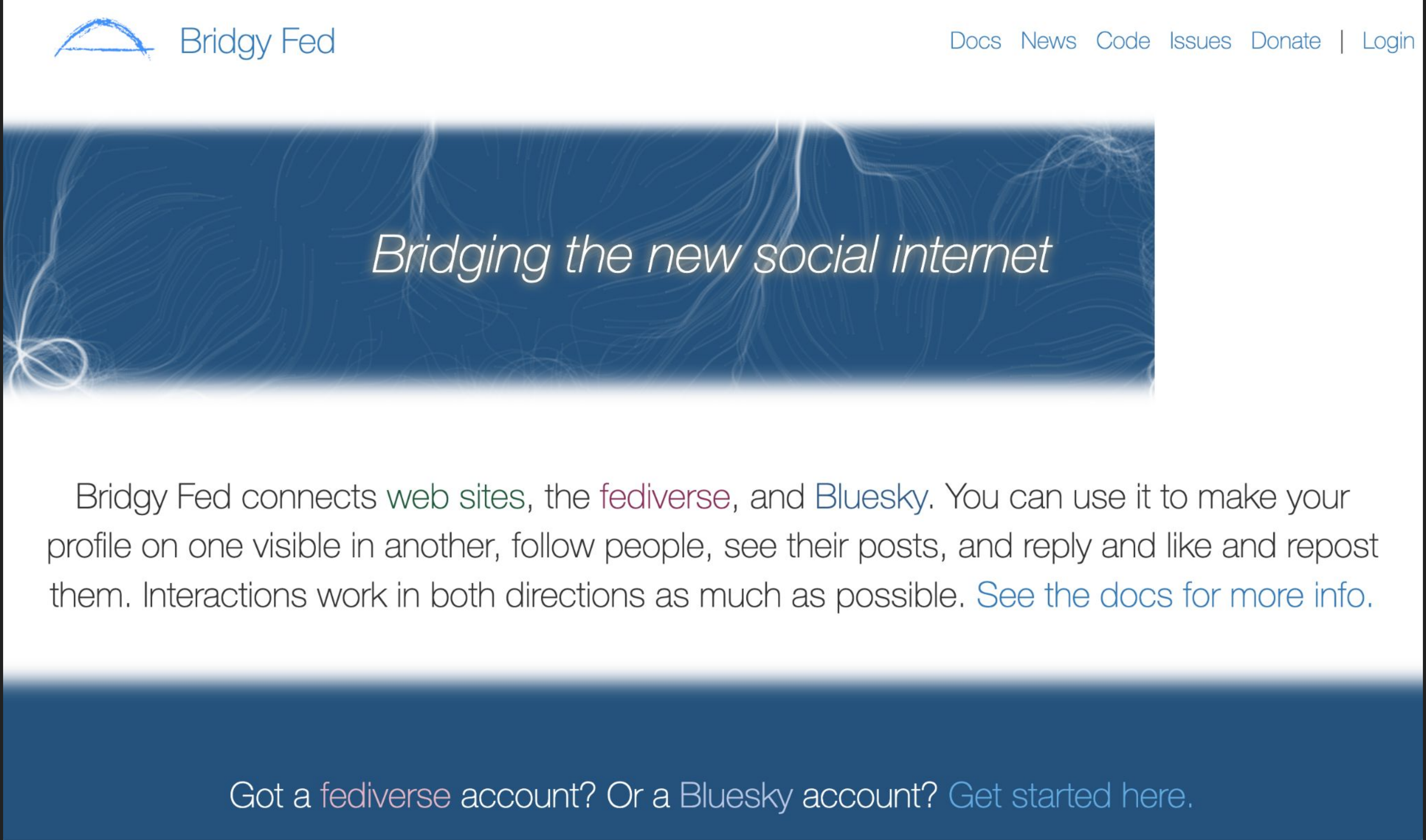
A Social Filesystem

January 18, 2026

Pay what you like

Remember files?

You write a document, hit save, and the file is on your computer. It’s yours. You can inspect it, you can send it to a friend, and you can open it with other apps.

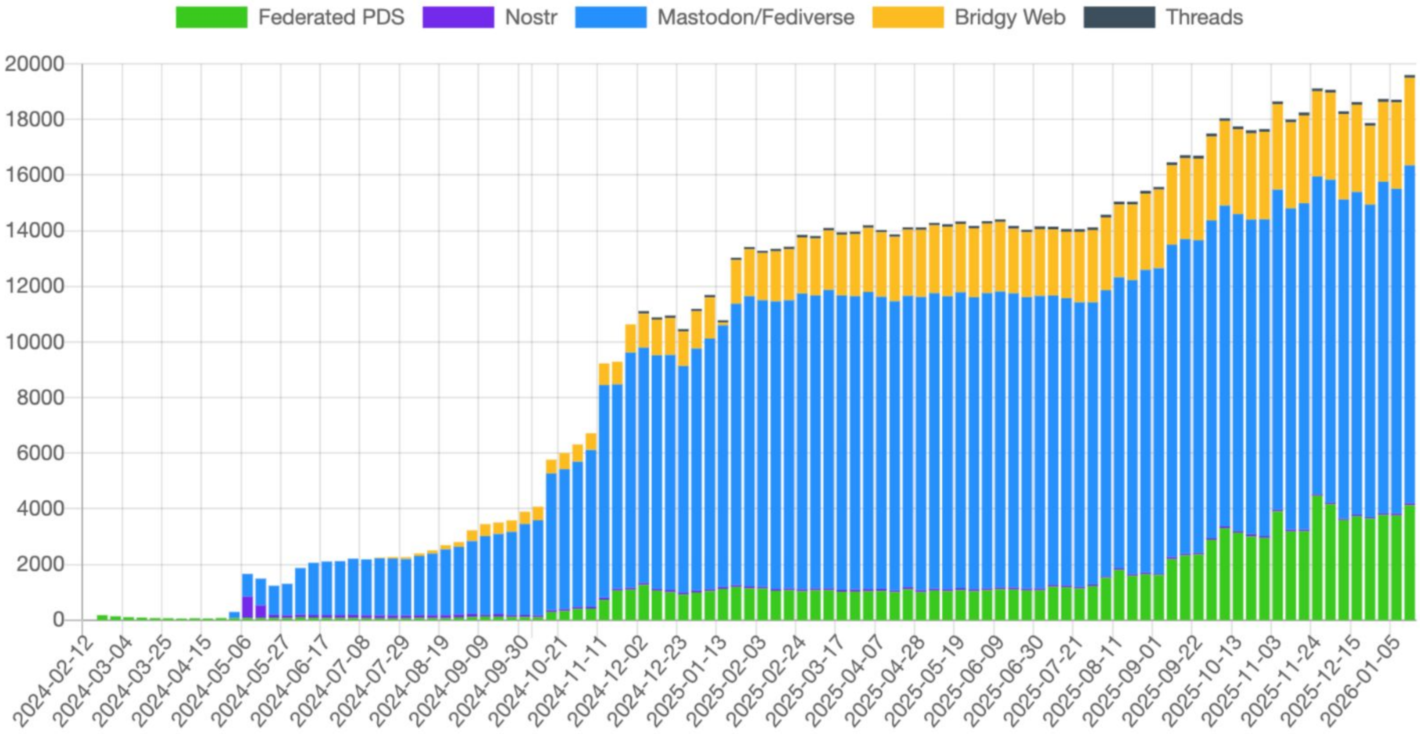


Bridging the new social internet

Bridgy Fed connects web sites, the fediverse, and Bluesky. You can use it to make your profile on one visible in another, follow people, see their posts, and reply and like and repost them. Interactions work in both directions as much as possible. See the docs for more info.

Got a fediverse account? Or a Bluesky account? Get started here.

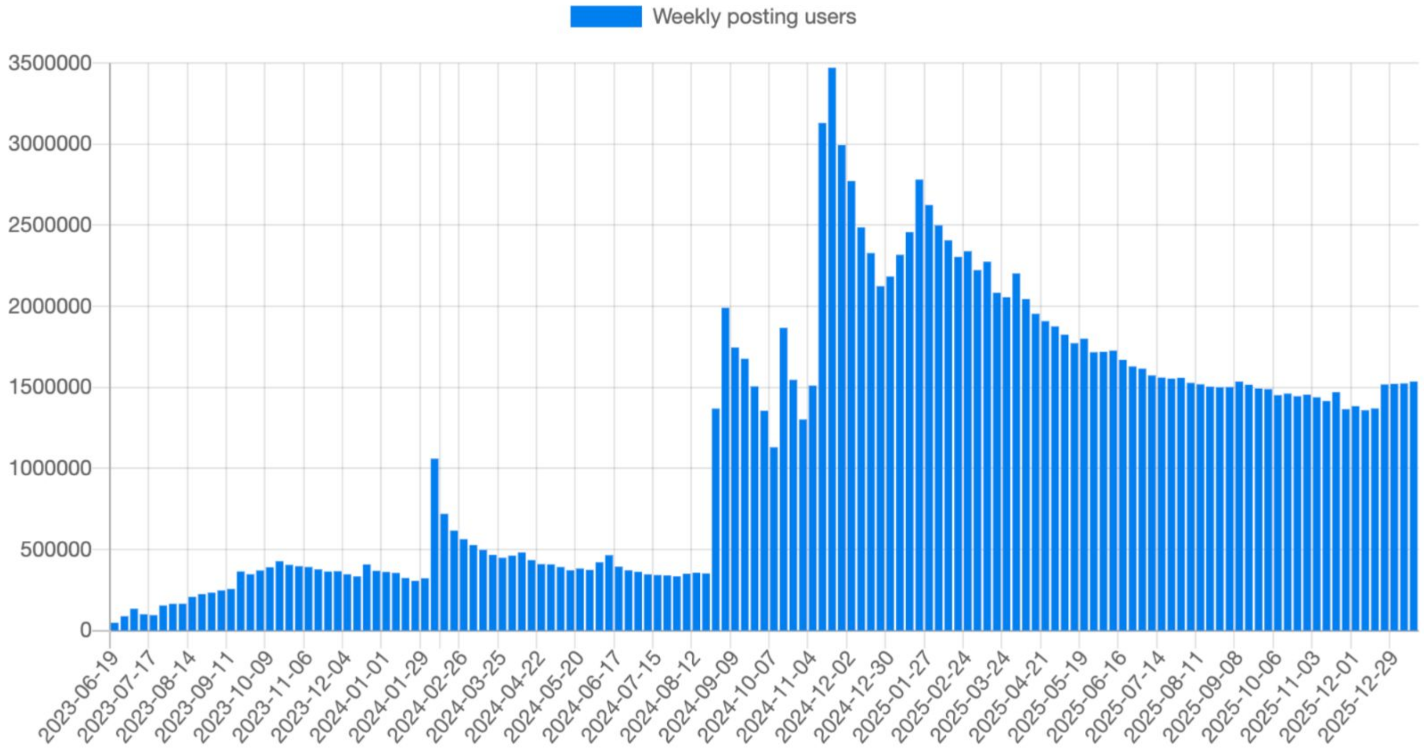
Weekly users posting from non-Bluesky PDSes:



(click a series in the legend to toggle it on/off)

ⓘ Note: “Federated PDS” here means any non-Bluesky managed PDS except Bridgy; the other four groups are all users bridged via Bridgy, grouped by where the original posts are from – technically both “Nostr” and “Threads” also come through Fediverse.

Unique weekly posting users (excluding bridged):



Weekly users posting from non-Bluesky PDSes:

So… why build on ATProto?

- No need to architect your own data models
- SDK code generation provides first-class support for all Lexicons
- Build on top of existing social graph
- All records are basically HTTP GETs or POSTs
- Hyper-engaged developer community
- Self-host as much or as little of the stack as you want
- Low-code, no-code, or lotsa-code? Solve for your use case
- Protocol interoperability. And big-world scale.

Thank you!

You can find me at @alex.bsky.team (or @axfelix@digipres.club Fedi)

The 2026 Atmosphere conference will be in Vancouver, March 26 - 29

Tickets are still available! Lots of exciting talks! Meet the community!

2026 will be the “Atmos-year” (sorry)

Questions?