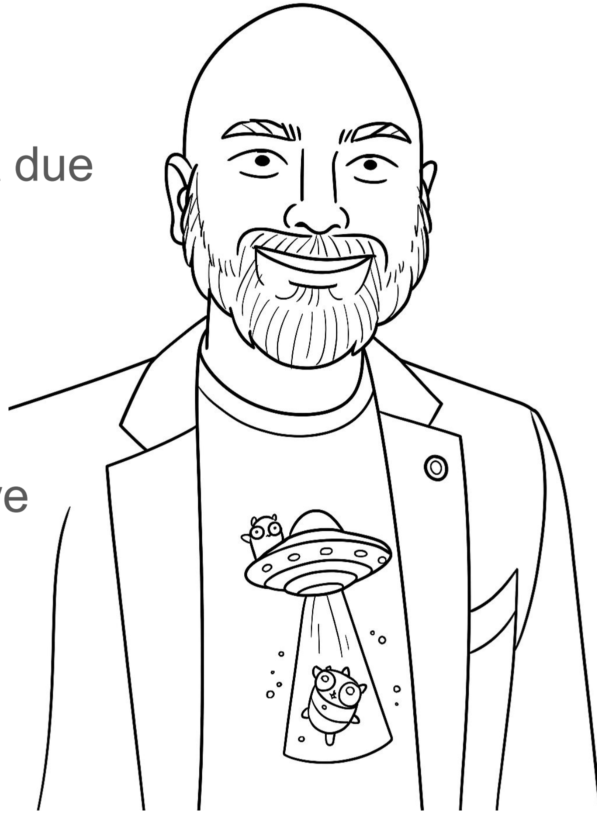
Building Decentralized Social Apps with Go and @Proto

bit.ly/4mAQOjC

Who Am I?

- Developing in Go since 2012
- Work with go full time at Scoutred (A land development due diligence search company)
- Core developer of Tegola Vector tile server (tegola.io)
- Heard about Bluesky around the start of 2024, and got excited about the Architecture, and it’s possibilities.
- Want more people to know about what is possible so we can have more and nicer things!



bit.ly/4mAQOjC

What to expect from this Talk

The objective of this talk is to show you how easy it is to build an Application that makes use of @Proto, and the variety of applications that can be built on top of it.

In order to Accomplish this we will go through the following steps:

1. Quick History
2. A Brief Look at Various Components of the Architecture
3. Focus in On Identity
4. Build out a simple AppView
5. Showcase other apps built on the @Proto

bit.ly/4mAQOjC

Quick History of @Proto

- Started out as a internal project in Twitter
- Spun out as independent Public Benefit Corporation
- BlueSky is an AppView run by PBC who's CEO is Jay (@jay.bsky.team)
- ATProto is Developed and maintained by BlueSky and independent developers
- Not all services can be fully run independently of BlueSky just yet.
- Architecture very different from Mastodon, closer to the Web. Less Federated and more **Distributed**

The Protocol is capable of much more.

bit.ly/4mAQOjC

What is it Authenticated Protocol (@Proto)?

It is a protocol that can be used use build social interoperable applications; like BlueSky.

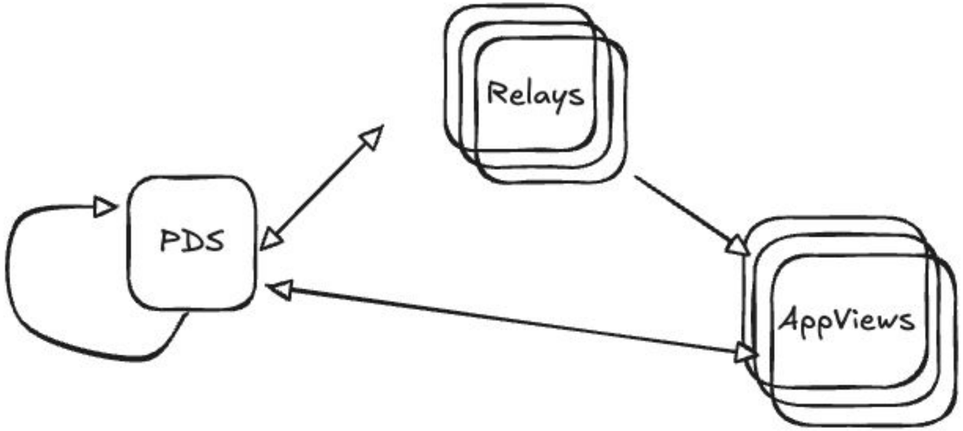
bit.ly/4mAQOjC

What is it Authenticated Protocol (@Proto)?

It is a protocol that can be used use build social interoperable applications; like BlueSky.

Made up of three components:

1. Personal Data Server (PDS)



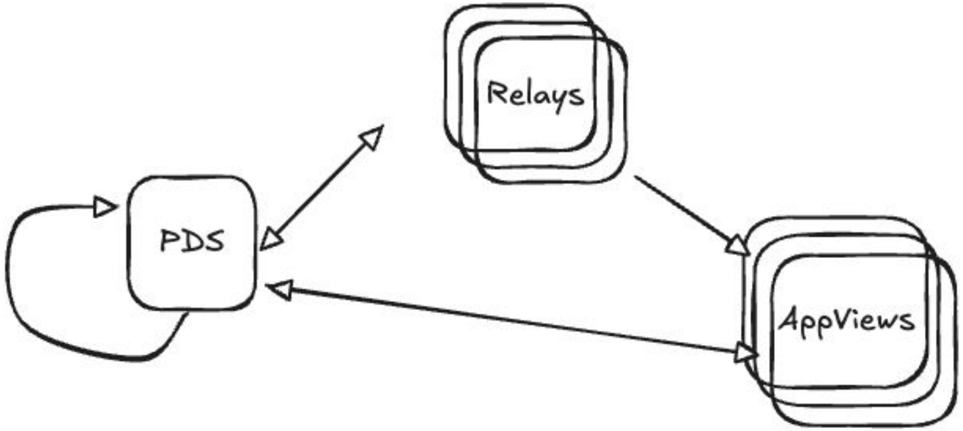
bit.ly/4mAQOjC

What is it Authenticated Protocol (@Proto)?

It is a protocol that can be used use build social interoperable applications; like BlueSky.

Made up of three components:

1. Personal Data Server (PDS)
2. Relays



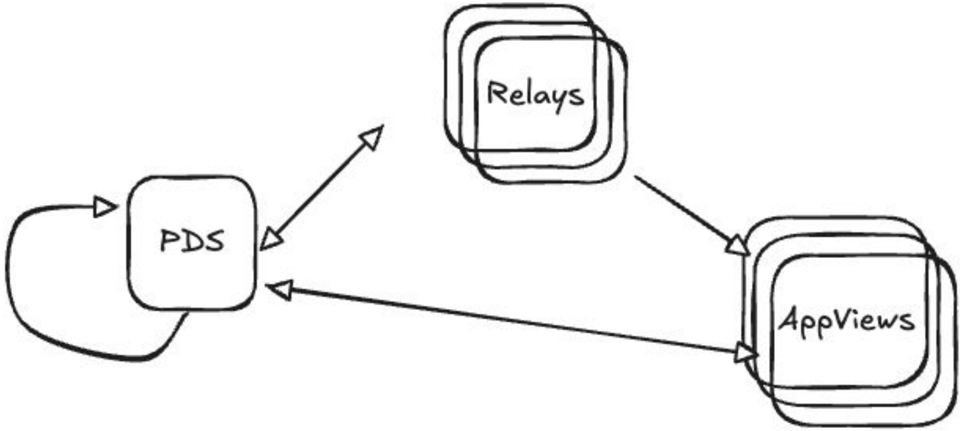
bit.ly/4mAQOjC

What is it Authenticated Protocol (@Proto)?

It is a protocol that can be used use build social interoperable applications; like BlueSky.

Made up of three components:

1. Personal Data Server (PDS)
2. Relays
3. AppViews
   a. Filters
   b. Labelers



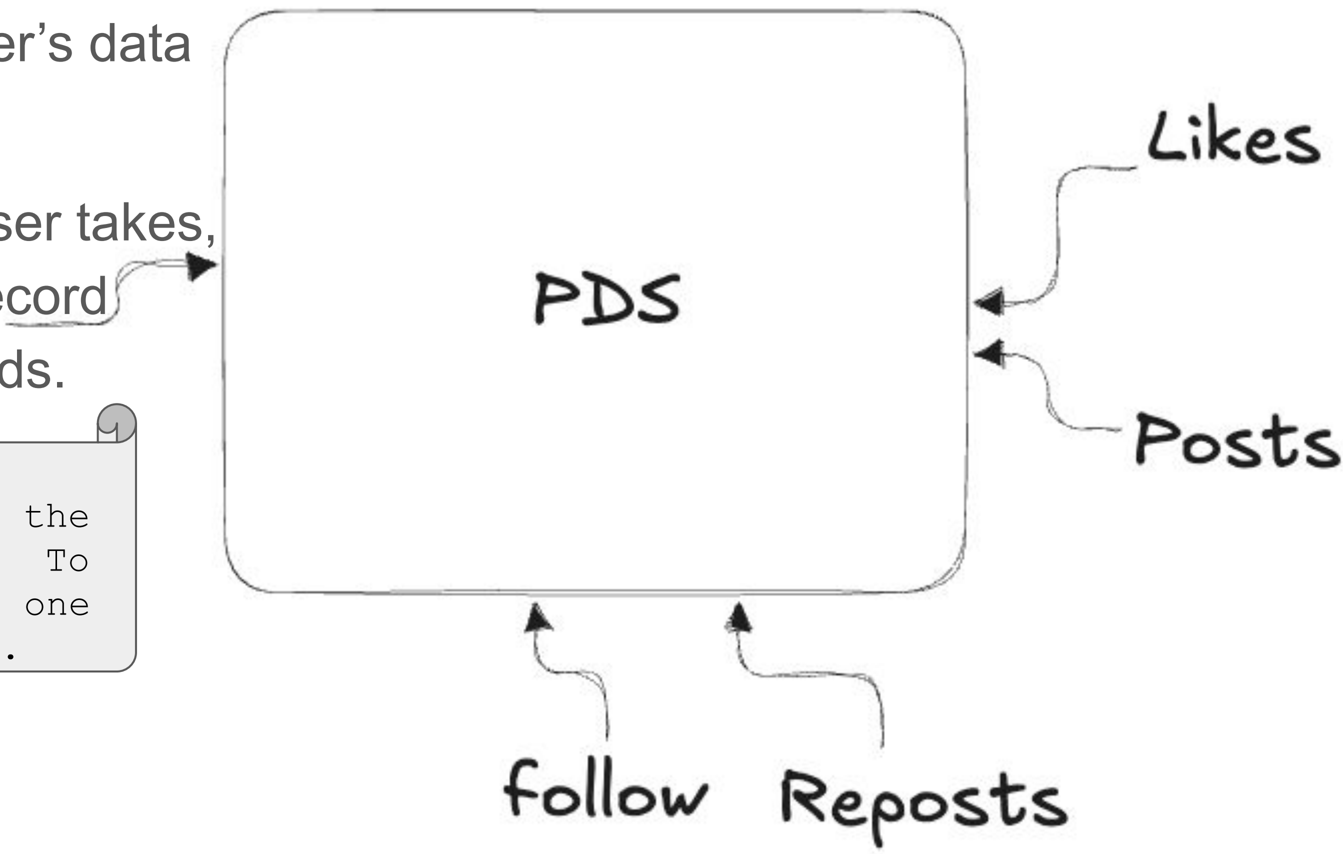
bit.ly/4mAQOjC

PDS (Personal Data Server)

A PDS is where a user's data is stored.

Every action that a user takes, should generate a record that is stored in the pds.

Note:
All data store on the PDS is public read. To write to the PDS one must be Authenticated.



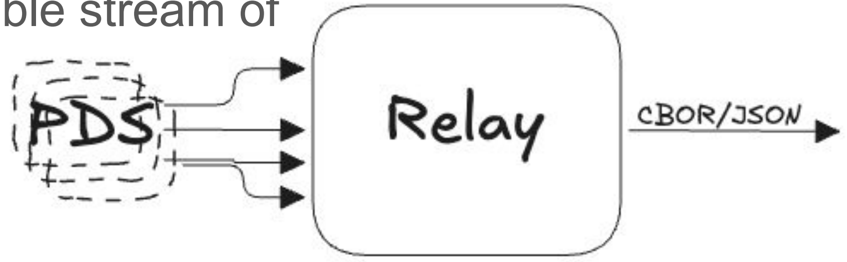
bit.ly/4mAQOjC

Relays

Relays are the heart of the system, they gather the changes across all the PDS and create a data stream for others to ingest.

There are two relays:

1. Firehouse is an unfiltered stream of encoded data
2. Jetstream (build onto of Firehose) a filterable stream of JSON data



bit.ly/4mAQOjC

Demo: Firehose

```
go run github.com/CharlesDardaman/blueskyfirehose firehose
```

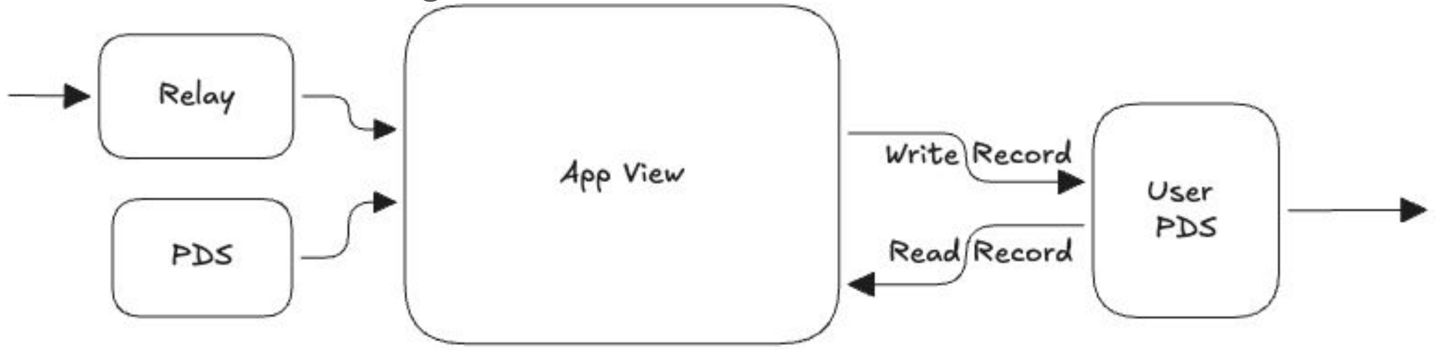
bit.ly/4mAQOjC

App Views

App views are the applications that allow users to interact with the data produced by the PDS and to generate new data to be written to the User's PDS.

Two Types of "special" App Views

- Labelers — Enable moderation and content classification
- Feed Generators — Custom feed generation



bit.ly/4mAQOjC

Labelers

- Produce judgements about user-generated content, such as identifying spam, inappropriate material, pronouns, etc…
- AppView and PDS can incorporate these labels to find strategies for displaying labeled content; such as blurring, hiding or enhance the view in other ways.
- Bluesky provides “OZone” as tool for moderation, to enable users to create their own moderation strategies.

bit.ly/4mAQOjC

Feed Generators

Process the posts from the firehose for inclusion of customer feeds.

Users can build custom feeds using tools like Graze (https://www.graze.social/)

bit.ly/4mAQOjC

Handles and Identity

- Handles are DNS based names
- Handles resolve to a Decentralized Identifiers (DID)
- DID are used to get PDS server of the User

We are going to resolve a Handle to the PDS server the “hard way” at first.

Handle -> DID

There two ways to resolve this, and both need to be tried.

- DNS TXT _atproto.${handle} record.
- GET https://${handle}/.well-known/atproto-did

bit.ly/4mAQOjC

> dig TXT _atproto.gdey.me

```
; <<>> DiG 9.10.6 <<>> TXT _atproto.gdey.me
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 31260
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;_atproto.gdey.me.        IN    TXT;; WHEN: Wed Aug 06 08:42:22 PDT 2025
;; MSG SIZE  rcvd: 94

;; ANSWER SECTION:
_atproto.gdey.me.    300    IN    TXT    "did=did:plc:vavedcex7kfly24u6hyqhqyk"

;; Query time: 78 msec
;; SERVER: 100.100.100.100#53(100.100.100.100)
```

```
> curl -s "https://georgetakei.bsky.social/.well-known/atproto-did"
```

```
did:plc:y4zs4cabaezzwx3bz2e5nnj2
```

Aside: Decentralized Identifiers (DID)

did:plc:vavedcex7kfly24u6hyqhqyk

The DID is broken into three parts, separated by “:”

1. \`did\` identifier, that identifies this as a did, should always be ‘did’
2. Method: The method, currently only two methods are supported
   1. plc : custom method made for ATProto. Currently relies on a central directory that is run by BlueSky

   2. web : based on the w3c standard did:web Method Specification

3. Arg: for the method.

bit.ly/4mAQOjC

DID -> DID Document

The DID document, gives us the location of the PDS server for the DID as well as some other important information like the public key for account.

For the PLC method we need to query the directory at “https://plc.directory/${did}”

bit.ly/4mAQOjC

DID Document

```
curl -s https://plc.directory/did:plc:vavedcex7kfly24u6hyqhqyk
```

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/multikey/v1",
    "https://w3id.org/security/suites/secp256k1-2019/v1"
  ],
  "id": "did:plc:vavedcex7kfly24u6hyqhqyk",
  "alsoKnownAs": [
    "at://gdey.me"
  ],
  "verificationMethod": [ … ],
  "service": [
    {
      "id": "#atproto_pds",
      "type": "AtprotoPersonalDataServer",
      "serviceEndpoint": "https://yellowfoot.us-west.host.bsky.network"
    }
  ]
}
```

Here is the PDS server: $\longrightarrow$ "serviceEndpoint"

bit.ly/4mAQOjC

Go Code for resolving ID, from Gallery App.

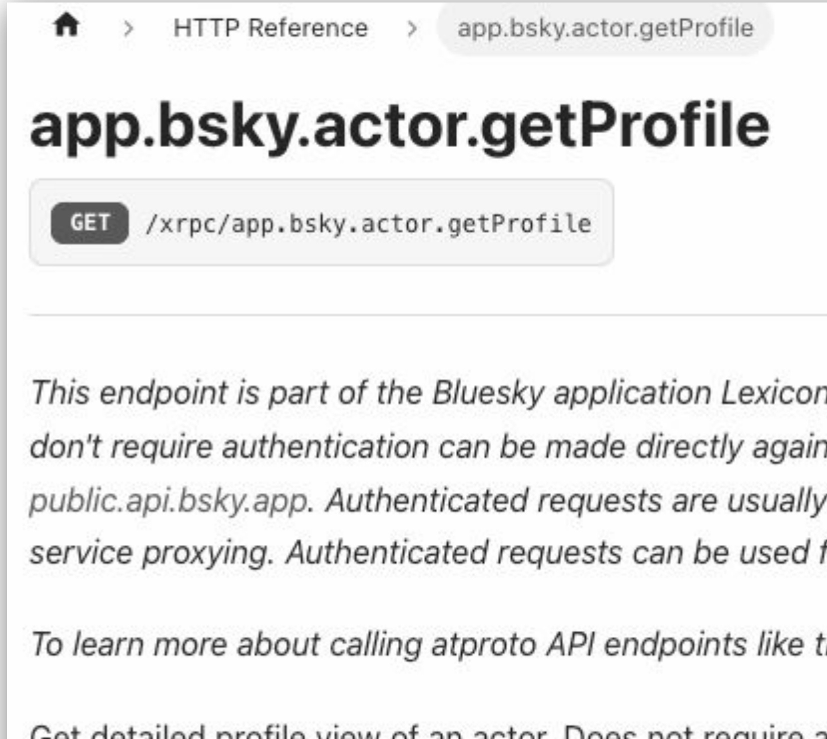
```
atId, err := syntax.ParseAtIdentifier(handle)

if err != nil {...}

return identity.DefaultDirectory().Lookup(ctx, *atId)
```

bit.ly/4mAQOjC

Aside: How xRPC works.

We are quickly going to go over the basics of how RPC works in @Proto.

- @Proto is built on HTTP
  - action QUERY maps to GET
  - action PROCEDURE maps to POST
- URL Looks like the following
  - {{action}} {{host}} + "/xrpc/"+method?parameters



app.bsky.actor.getProfile

GET /xrpc/app.bsky.actor.getProfile

This endpoint is part of the Bluesky application Lexicon don't require authentication can be made directly agai public.api.bsky.app. Authenticated requests are usually service proxying. Authenticated requests can be used

To learn more about calling atproto API endpoints like t

Get detailed profile view of an actor. Does not require

bit.ly/4mAQOjC

Aside: get profile using public endpoint

```
+ curl -s -X GET 'https://public.api.bsky.app/xrpc/app.bsky.actor.getProfile?actor=did:plc:vavedcex7kfly24u6hyqhqyk'
+ jq -S
{
  "associated": {
    "activitySubscription": {
      "allowSubscriptions": "followers"
    },
    "feedgens": 0,
    "labeler": false,
    "lists": 0,
    "starterPacks": 0
  },
  "avatar": "https://cdn.bsky.app/img/avatar/plain/did:plc:vavedcex7kfly24u6hyqhqyk/bafkreidvdwmjz64jf5gzbaskmetscfv6o374oyhxwnta2icuy6uhkeegau@jpeg",
  "createdAt": "2024-11-25T00:34:58.746Z",
  "description": "Just a nobody and a fool, trying hard to not be the greater fool.",
  "did": "did:plc:vavedcex7kfly24u6hyqhqyk",
  "displayName": "Gautam Dey",
  "followersCount": 383,
  "followsCount": 382,
  "handle": "gdey.me",
  "indexedAt": "2024-12-11T23:27:49.842Z",
  "labels": [],
  "postsCount": 272
}
```

bit.ly/4mAQOjC

Building A Simple AppView

Gallery App

App Should:

1. Show latest 5 pictures of person at gdey.ngrok.io/@{handle}
2. If the person is logged in via gdey.ngrok.io/login (using OAuth)
3. Use 🩵,🦋,🌝
   gdey.ngrok.io/gallery/{handle}/{action}/{record}/{cid}

bit.ly/4mAQOjC

Code Time?!

bit.ly/4mAQOjC

Other Applications Built On @Proto

- Tangled (https://tangled.sh) — git code collaboration platform
- Paste Sphere (https://pastesphere.link/) – a paste bin for sharing code
- Smoke Signal (https://smokesignal.events) – event and RSVP management
- Leaflet (https://leaflet.pub) — Markdown blog service
- Streamplace (https://stream.place) — Live streaming

The only thing missing here is service you are going to build!

bit.ly/4mAQOjC

Thank you!

I’m Gautam Dey, you can find me on BlueSky @gdey.me
And please Go visit the sponsors, without whom this conference can not happen.

References

- https://docs.bsky.app/docs/get-started
    - https://docs.bsky.app/docs/api/app-bsky-actor-get-profile
- https://atproto.com/specs/oauth#authorization-servers
- https://atproto.com/
- https://datatracker.ietf.org/doc/html/rfc9207
- https://github.com/bluesky-social/atproto/discussions/2350
- https://github.com/bluesky-social/indigo
- https://github.com/bluesky-social/atproto/discussions/2656
- https://github.com/bluesky-social/proposals/tree/main/0004-oauth
- https://github.com/lexicon-community/awesome-lexicons
- https://blog.smokesignal.events/posts/3ltg5xg3me22c-3ltg5xg3me22c-building-and-testing-atprotocol-applications
- https://dev.to/pipodev/bluesky-oauth2-client-with-vanilla-javascript-1f6h
- https://github.com/rmtuckerphx/bluesky-developer-guide/tree/main
- https://github.com/CharlesDardaman/blueskyfirehose
- https://w3c-ccg.github.io/did-method-web/#web-did-method-specification
- https://blueskydirectory.com/glossary/did