分散システム・オープン Web への関心

安田陽真

目次

1 関心の理由・きっかけ

私が分散システム・オープン Web に関心を持ったきっかけは、中学生の頃に Twitter を誤って凍結された経験です。過去の投稿や収集した情報、築いてきたつながりが、運営企業の一存で失われることを実感しました。

その後出会ったのが、分散型 SNS「Misskey」です。最初は豊富な絵文字リアクションに惹かれただけでしたが、次第にトップページにも掲げられている「分散型 SNS」という仕組み自体に興味を持つようになりました。調べていくうちに、Misskey が ActivityPub というプロトコル上で動作していること、他にも Nostr や ATProtocol といった分散型 SNS プロトコルが存在することを知りました。

半年ほど Misskey をメイン SNS として使う中で、サーバーの障害などに遭遇することがありました。「分散型のはずなのになぜサーバーダウンが起きるのか」という疑問から、各プロトコルの比較を始めます。Nostr の依存が薄いリレーモデル(データを複数のリレーにばらまくモデル)や、ActivityPub のインスタンス連合モデルを調べる過程で、ActivityPub を採用する Misskey や Mastodon は、実質的に人数が特定インスタンスに集中する「小さな中央集権型 SNS」の集合になっていることに気づきました。

一方で ATProtocol は構造のイメージがなかなか掴めず、集中的にリサーチを重ねました。その結果、ATProtocol が ActivityPub とは根本的に異なる「データとサービスの分離」というアプローチを取っていることを理解し、その背後にあるプロトコルやデータ構造、暗号技術そのものに強く惹かれるようになりました。

こうしたオープンプロトコルの上では、プロトコルさえ守れば実装は自由で、誰でもネットワークに参加できます。また暗号技術を用いれば、従来は信頼機関のお墨付きに依存していた事柄を、自力で証明できるようになります。この寛容さと力強さに、私は惹かれていきました。

本章では、こうした関心に沿った取り組みを時系列で示します。

2　AniBlue ～ATProtocol 上でアニメの視聴履歴管理～

開発時期: 高校 1 年(2024 年 11～12 月)

2.1　概要

AniBlue は、ATProtocol 上でアニメの視聴管理をすることができるアプリケーションです。あるアニメについて何話まで視聴した、という視聴履歴を保存することができます。AniBlue は、ATProtocol の学習目的で開発しましたが、結果として日本の Bluesky コミュニティで 10 人以上のユーザーに利用されるアプリケーションとなりました。

2.2　開発の動機・考え

ATProtocol について知った段階で、学習用の小さなサンプルアプリケーションのアイデアとしてアニメ視聴管理が浮かび、開発しました。

2.3　仕様・技術

ATProtocol は、主に PDS、BGS、AppView という 3 つのコンポーネントにより構成されています。 ここでは AniBlue の技術説明のため、各コンポーネントについて簡潔に説明します。

- PDS (Personal Data Server)
  - ATProtocol 上でのユーザーの全てのデータ(投稿、いいね、フォロー etc...)を保存する、暗号学的署名済みの個人用データリポジトリです。ATProtocol では、PDS 上のデータがユーザーデータのソース・オブ・トゥルースとされます。PDS 上のデータはすべて JSON であり、その型は Lexicon という JSON Schema ライクな独自のスキーマ言語で記述されます。
- BGS (Big Graph Server)
  - ネットワーク全体をクロールするリレーサーバーです。複数の PDS に接続し、リアルタイムで PDS 上のデータ変更履歴を受け取ります。接続しているすべての PDS の変更履歴を、Firehose と呼ばれる WebSocket のインターフェースを用いて、後述する AppView に中継します。
- AppView
  - ATProtocol を用いた Web サービス本体です。BGS から流れてくるネットワーク全体の変更履歴をフィルタし、必要なデータ型の変更履歴を取得、DB に保存して、インデックスを作成します。

AniBlue は、このうち PDS と AppView というコンポーネントの上で動作しています。データフローは以下の通りです。

1. AniBlue クライアントは、OAuth プロトコルに従った認可によって、ユーザーの PDS に対する書き込み権限を得ます。
2. ユーザーは、PDS 上に `app.netlify.aniblue.status` というコレクション名で識別される、アニメの視聴状態を示した JSON を書き込みます。
3. アプリケーション側は、この JSON を元に視聴履歴を描画します。

JSON のサンプルを、以下に示します。

```JSON
{
  "$type": "app.netlify.aniblue.status",
  "status": [
    {
      "id": 10115,
      "title": "ラブライブ！スーパースター!! 3期",
      "status": "watched",
      "favorite": true,
      "thumbnail": "",
      "episode_text": "第12話"
    },
    {
      "id": 7204,
      "title": "ラブライブ！虹ヶ咲学園スクールアイドル同好会",
      "status": "watched",
      "favorite": true,
      "thumbnail": "https://www.lovelive-anime.jp/nijigasaki/logo.png",
      "episode_text": "第13話"
    },
    {
      "id": 2274,
      "title": "ゆゆ式",
      "status": "watched",
      "favorite": true,
      "thumbnail": "",
      "episode_text": "第12話"
    },
  ...省略
```

AniBlue は PDS をデータストアとしてシンプルに利用しています。アニメに関するメタデータの取得には、Annict API というアニメに関するオープンな API を利用しており、先程の JSON に登場した“id”は Annict API 上での識別番号です。

2.4 成果

このアプリケーションは、学習用の小さなサンプルアプリケーションでしたが、幸いなこと

に、10 人以上の方々に触れていただくことができました。[1][2] 自分の開発したアプリケーションを多数のユーザーに触れてもらうという経験が少なかったため、とても印象的だったことを覚えています。

AniBlue は、使用しているフレームワークのアップデート・脆弱性対応コストなどから約 1 年でサービスを終了しましたが、視聴履歴そのものは利用者の PDS に残っています。このように、サービスが終了したとしても、データがしっかりと利用者の手元にあるという点が「データとサービスの分離」アプローチの核心だと考えています。

[1] https://velocity.jp/anime/3200/
[2] https://atasinti.chu.jp/dad3/archives/74723

3　Stellar ～Bluesky で絵文字リアクション～

開発時期: 高校 1 年(2025 年 1～3 月)

3.1　概要

Stellar は、Bluesky 上で絵文字リアクションを行うことができる独自クライアントです。ATProtocol 上で絵文字を表現するための Bluemoji という規格に準拠しています。この Bluesky クライアントは、日本だけでなく海外の Bluesky コミュニティで複数のユーザーに使用され、分散型 SNS の動向を追うブログ The Fediverse Report[3]に掲載されました。また、Bluemoji の仕様を策定しているメンテナーからの反応もあり、Bluesky 上での絵文字リアクションの実証として一定の成果を収めました。

3.2　開発の動機・考え

ATProtocol の代表的な SNS 実装、Bluesky を利用する中で、Misskey のような絵文字リアクションができないことに不満を感じていました。リサーチを進めると、ATProtocol 上の絵文字リアクション仕様である Bluemoji というプロジェクトを発見しましたが、その仕様を用いて絵文字リアクションをすることができるクライアントが発見できず、自分で開発することにしました。

3.3　仕様・技術

2.2 で述べた AniBlue では、要件が「ユーザーに一対一で紐づいているデータを取得する」というものであったため、AppView から直接 PDS に対してリクエストを投げて、必要なデータを取得することができました。ですが、Stellar では少し事情が異なります。Stellar では「ある投稿についた絵文字リアクション全てを収集する」というネットワークを横断したような query が必要になります。そのため、AniBlue では利用しなかった BGS の利用が肝となります。データフローは以下のとおりです。

1. 各ユーザーがそれぞれの PDS に、`blue.maril.stellar.reaction` というコレクション名で識別されるリアクションデータ本体を作成します。
2. PDS が `blue.maril.stellar.reaction` の作成イベントを BGS に渡します。
3. Stellar の API サーバーは、BGS から `blue.maril.stellar.reaction` に対する作成/削除イベントのみを監視しています。先程の作成イベントは、BGS を通って Stellar の API サーバーまで中継されます。

[3] https://fediversereport.com/last-week-in-the-atmosphere-2025jan-d/

4. Stellar の API サーバーは BGS から中継された作成イベントをローカル DB に保存します。
5. Stellar クライアントは、Bluesky の投稿を表示する段階で、Stellar API サーバーにその投稿に対する絵文字リアクションリストを query し、描画します。

AniBlue では、クライアントは直接 PDS にリクエストを投げていました。しかし、PDS は純粋な API 付きドキュメント DB であり、ネットワークを横断した検索をすることはできません。ATProtocol では、このような場合 BGS が収集しているネットワーク全体の更新履歴を一箇所のインデックスに集中させ、そこで高速な検索を行うというパターンが一般的で、Stellar もそのパターンに則っています。

3.4 成果

実際に独自の AppView とクライアント実装を用いて、Bluesky 上で絵文字リアクションが可能であることを実証しました(図 1)。

図 1: Bluesky 上で絵文字リアクションがついている様子

本クライアントは、日本のみならず海外の Bluesky コミュニティでも利用され、分散型 SNS の動向を追うブログ The Fediverse Report[4]に掲載された他、Bluemoji の規格を策定しているメンテナーからの反応[5]もありました(図 2)。

[4]https://fediversereport.com/last-week-in-the-atmosphere-2025jan-d/
[5]https://bsky.app/profile/aendra.com/post/3lgxd4yd2sc2n

図 2: Stellar に対する Bluemoji メンテナーからの反応

一方、この実装を進める中で感じた、「インデックスを集中させる設計」への違和感が、次の開発のきっかけとなります。

4　polka ～よりローカルなネットワークを目指して～

開発時期: 高校 2 年(2025 年 4 月～2026 年 3 月)

4.1　概要

ATProtocol では AppView 層において、インデックスが 1 つのサーバーに集中しているという気づきから、「なぜこのような中央集権的な点が存在するのだろう」と考えたことを出発点とし、SecHack365 で polka という分散 Web プロトコルの策定に取り組みました。polka は、「世界中のどんな情報でも瞬時に受け取れる」という価値観よりも、「同じ興味を持った仲間と、ゆるやかに繋がれる」ことを重視し、ATProtocol や Nostr といった各分散 SNS プロトコルの一部仕様を借用、そこに独自の技術要素等を加えることで、興味関心に従って分散的に繋がれる Web を実現しました。また、本プロトコルの開発で SecHack365 を修了しました。

4.2　開発の動機・考え

Stellar の開発を通して、ATProtocol のネットワーク横断検索の仕組みに触れる中で、「なぜこのような中央集権的な点が存在するのだろう」という疑問を持ちました。この疑問から、P2P レプリケーションによってインデックス層を分散させる分散型 SNS を構想し、SecHack365 に採択されました。

しかし開発を進める中で、NAT やパフォーマンスといった P2P 技術特有の構造的な問題に直面しました。この壁をきっかけに、「なぜこのような中央集権的な点が存在するのだろう」と改めて自分に問い直し、ATProtocol の設計判断を見直したところ、ATProtocol のメイン実装である Bluesky には「検索性と中央集権性」というトレードオフと、それに基づく強い前提があることに気づきました。

ネットワーク全体を横断的に検索するには、インデックスを一箇所に集約するのが最も効率的です。そんな中、Bluesky は「どこから見てもリアルタイムに同じ情報が見える」ことを前提に SNS を設計しています。一方、ActivityPub のようなプロトコルにはそもそもグローバル検索という概念がなく、見えるデータは所属するサーバーに依存します。

では、ActivityPub のほうが分散的なのでしょうか。私はそうは考えていません。私にとっての「分散的」とは「アイデンティティが奪われないこと」です。ActivityPub では、`alice@example.com` のように、id が所属インスタンスに紐づいています。インスタンス間の移行自体は可能でも、所属が変われば id も変わり、アイデンティティの一部を失うことになります。

その点、ATProtocol の DID は PDS から独立しており、移行しても id が変わりません。id が所属に縛られないという意味では、たしかに ATProtocol のほうが分散的だと言えます。

ただし、アイデンティティは id だけで成り立つものではありません。「自分が誰であるか」は「自分が誰と繋がっているか」という関係性によっても構成されます。そしてその関係性は、他者から参照可能でなければ社会的には機能しません。ソーシャルグラフは、まさにこの関係性としてのアイデンティティを最も具体的に表現するものです。

ここで問題になるのが、前述した検索性と中央集権性のトレードオフです。ソーシャルグラフを「誰から見ても同じ情報がリアルタイムに見える」形で検索可能にするということは、その可視性を BGS や AppView という中央のインデックスに委ねることを意味します。ActivityPub が id を所属に縛っていたのに対し、ATProtocol は id を解放する一方で、検索性という別の層でアイデンティティのもう一つの側面を中央に預けている、と言えます。

私は、Bluesky が重視した「どこから見ても同じ情報がリアルタイムに見える」という UX よりも、id だけでなく関係性も含めたアイデンティティを自分の手で持てることのほうが重要だと考えています。この価値観から、私は polka を開発しました。

4.3 仕様・技術

polka は、「タグとリンクでつながる、新しい分散 Web」をキャッチコピーとした、タグを中心とした分散的な情報ネットワークを構築するためのプロトコルです。

polka は、個人サイト文化に着想を得ています。個人サイトは完全に自分のものです。自分のドメイン、自分のサーバー、自分の HTML ファイルでは、誰にも削除されることなく、自由に表現できます。polka は、セルフホストの自由さに、現代の暗号技術による検証可能性を組み合わせます。ユーザーは自分のドメインでデータをホスティングするため、サービス提供者に依存せず、アカウント BAN の心配もありません。そのうえで、すべてのデータは電子署名されており、改ざんされたデータは暗号学的に検証不可能となって自動的に排除されます。個人サイトの自律性と暗号学的な信頼性、この 2 つを両立させることこそが、polka の目指す世界です。

polka のもう一つの重要なコンセプトは、「ソーシャルグラフを大規模インフラに依存させない」ことです。従来の分散型 SNS では、ソーシャルグラフは大規模なリレーやインデクサーに保存されており、これらのインフラが停止すればつながりそのものが失われてしまいます。これに対し polka では、ソーシャルグラフは各ユーザーのサーバーに保存されます。ユーザーがフォローしているタグのリストはユーザーのサーバーに保存され、相手の HTTPS サーバーから直接データを取得する仕組みです。polka では、ユーザー発見のために Nostr リレーを用いていますが、これは本当に「一時的な発見」のためだけに使われます。一度つながりを見つけれ

ば、そのつながりはローカルに保存され、以降は直接 HTTPS でアクセスするようになります。これは現実世界で誰かと知り合う過程に似ています。最初はイベントや SNS で出会うとしても、一度つながってしまえば、そのイベントが消えても関係そのものは続いていくものです。

さらに polka を特徴づけるのが、タグを中心とした緩やかなつながりです。polka では、ユーザーではなく「タグ」をフォローします。たとえば Alice さんのネットワークに関する投稿だけに興味があるなら、`alice.example.com/tech/network` をフォローすればよく、タグは階層構造を持つため、`alice.example.com/tech` をフォローすればその配下の `network` や `os` などすべてのサブタグも取得されます。これにより、細かい興味に絞ることも、広いトピック全体を購読することも自在にできます。また、タグには中央的な管理者がいません。`network` のようなタグは、それを使いたい人が自由に使うものであり、同じタグを使う人々が自然につながることでコミュニティが形成されていきます。

4.3.1 タグ構造の一例

```
📁 alice.example.com/
  └ tech/
      ├ web/
      │   └ [投稿1, 投稿2, ...]
      └ ai/
          └ [投稿3, 投稿4, ...]
  └ photo/
      └ landscape/
          └ [投稿5, 投稿6, ...]

📁 shino.example.com/
  └ tech/
      ├ web/
      │   └ [投稿7, 投稿8, ...]
      └ os/
          └ [投稿9, 投稿10, ...]
```

こうして polka が目指すのは、大規模なグローバルネットワークではなく、小さな興味のクラスターが緩やかにつながり合う「スモールワールド」です。すべてのユーザーがすべてのユーザーを発見する必要はなく、自分の興味のあるタグに関わる人々とつながれば、それで十分なのです。この設計によって、大規模なインフラを必要とせずとも、意味のあるつながりが育まれていきます。

技術的には、データ層の ATProtocol Repo、ストレージ層の Git と静的ファイルホスティング、アイデンティティ層の did:web と HTTPS/DNS、発見層の Nostr と Bloom Filter、そして検証層の電子署名(Secp256k1)という複数のレイヤーから構成されています。データ層はデータの構造化

と検証を担い、Merkle Search Tree によってすべてのデータをまとめて検証できるようにします。ストレージ層はデータの永続化と配信を担当し、Git でバージョン管理しながら静的ファイルとしてHTTPSで配信します。アイデンティティ層はドメインと公開鍵の紐付けを担い、既存のHTTPS/DNSインフラを信頼の基盤とします。発見層はユーザーの発見を担当し、Nostrリレーに自分の存在を広告することで、興味のあるタグを持つユーザーを見つけます。そして検証層では、すべてのデータが署名され検証されることで、データの真正性が保証されます。これらのレイヤーが協調して動作することで、polka の分散型 Web が実現されています。

polkaのデータは、ATProtocolのリポジトリ形式で管理されています。その中核をなすMerkle Search Tree(MST)は、複数のデータをソート済みの木構造で管理し、ルートハッシュによって全体を検証できる仕組みです。polka では、タグの階層構造、投稿、プロフィールなど、すべてのデータがこの MST に格納され、そのルートハッシュに署名することでデータ全体の整合性が保証されます。実際のデータは、CID をキーとしたブロックストアに保存され、各ブロックはCBOR形式でエンコードされてそのハッシュがCIDとなります。クライアントは必要なブロックだけを取得できるため、リポジトリ全体をダウンロードする必要はありません。また、MST のルート、DID、署名を含む Commit オブジェクトが、リポジトリ全体のスナップショットとして検証の起点となります。

polka のデータ配信は極めてシンプルで、静的ファイルを HTTPS で配信するだけです。ユーザーのローカルでは、データは CAR 形式のファイルとしてすべてのブロックを含む形で保存されます。データを公開するときには、この CAR ファイルから個別のブロックファイルが生成され、各ブロックは`{CID}`という名前のファイルとして保存されるとともに、ルート CID は`ROOT`というファイルに書き込まれます。生成されたブロックファイルは Git リポジトリにコミットされ、これによりデータの変更履歴が記録され、必要に応じて過去の状態に戻すことも可能になります。こうして作られた Git リポジトリは、GitHub Pages、Netlify、Caddy、Nginx など、任意の静的ファイルホスティング方法で公開できます。クライアントは、`https://example.com/polka/ROOT`でルート CID を取得し、`https://example.com/polka/{cid}`で各ブロックを取得します。

リポジトリは、Secp256k1 電子署名によって保護されています。署名の流れとしては、ユーザーがデータを作成・更新すると、MST が再構築されて新しいルートハッシュが生成され、そのルートを含む Commit オブジェクトが作成されて秘密鍵で署名され、署名された Commit とブロックが保存されます。一方、検証の流れとしては、クライアントがまず ROOT からルート CID を取得し、そのルート CID で Commit ブロックを取得したうえで、did:web からユーザーの公開鍵を取得して Commit の署名を検証し、署名が正しければ MST を走査してデータを取

得します。改ざんされたデータは署名検証で失敗し、CID が一致しないブロックは拒否される仕組みになっています。

次に、アイデンティティについてです。polka では、ドメイン名と公開鍵の紐付けに did:web を使用します。ユーザーは、自分のドメインの`.well-known/did.json` に DID Document を配置し、このドキュメントには公開鍵と polka リポジトリのサービスエンドポイントが記載されます。DID Document には複数の公開鍵を登録できるため、鍵のローテーションや複数デバイスでの運用も可能です。クライアントは、いずれかの鍵で署名が検証できれば、そのデータを正当なものとして受け入れます。

前述した通り、polka ではユーザーごとにタグの階層構造を持ちます。たとえば `tech/os` や `anime/lovelive/mai` のような階層があり、これらのタグは相互にリンクし合ってグラフ構造を形成します。polka の設計においては、このグラフ全体を検証可能にする必要があり、1 つのタグだけでなくグラフ全体の整合性を効率よく証明する仕組みが求められました。ATProtocol のデータ構造、特に Merkle Search Tree は、まさにこの用途に最適で、複数のデータをまとめてハッシュ化し、そのルートハッシュに署名することでグラフ全体の正しさを効率的に証明できます。つまり polka は ATProtocol を、Bluesky のような大規模 SNS としてではなく、単純な「検証可能なデータベース」として使っているのです。

一方で、分散型システムには「他のユーザーをどうやって見つけるか」という根本的な課題があります。polka はこの問題に対して、Nostr の既存インフラを活用し、ユーザーは Nostr リレーに自分の存在を広告することで、興味のあるタグを持つ他のユーザーを発見します。ここで重要なのは、Nostr はあくまで一時的な発見の手段でしかないという点です。一度つながりを見つければ、その後のデータ取得は HTTPS 経由で直接行われるため、リレーがダウンしても既存のフォロー関係は影響を受けません。この設計により、大規模なリレーインフラに依存せず、最小限の調整だけでネットワークが機能するようになっています。

ユーザー発見の技術的フローは以下のとおりです。ユーザーは、自分が使っているタグを Bloom Filter にエンコードし、Nostr リレーに広告します。このイベントの kind は 25565 で、content には DID と Bloom Filter が含まれます。Bloom Filter は確率的データ構造であり、クライアントは受信した Bloom Filter に対して興味のあるタグが含まれているかをチェックします。偽陽性の可能性はあるものの偽陰性はなく、マッチした場合、クライアントは相手の DID を解決して実際のデータを取得し確認します。なお、Bloom Filter の偽陽性は、自分と関係のないトピックのユーザーも発見できるスパイスとして位置付けられています。一度マッチしたユーザーを見つけたら、そのユーザーのタグノードはローカルに保存され、以降は Nostr リレーを経由せず HTTPS 経由で直接データを取得するようになります。つまり Nostr は「ブートストラップ」の役割を担うものであり、継続的なデータ取得には使用されません。

4.4 成果

1 年の polka の開発を通して、自らの課題意識について深く考え、本質的な課題意識に気づくことができました。また、ATProtocol のデータ構造の細部や、DID といったアイデンティティについても深く学ぶことができました。また、本プロトコルの開発で SecHack365 を修了しました(図 3)。

図 3: SecHack365 成果発表会での様子

5　nijiiro ～オープンプロトコルを活かしたPDS～

開発時期: 高校3年(2026年7～8月)

5.1　概要

現状、大半のBlueskyユーザーがBlueskyの提供するPDSに所属し、Blueskyの管理するdid:plcというアイデンティティを使っています。その原因は、公式のPDS実装自体がとても重厚長大で、複数のユーザーが登録する前提で開発されていることや、小さなリファレンス実装が少ないことにあると考えます。この状況を解決するために、did:webというドメインを利用するシンプルなアイデンティティを用いて、Blueskyのインフラを利用せずにBlueskyに参加することができる小さなPDSを実装しました。

5.2　開発の動機・考え

私はこれまで、ATProtocolについて様々な側面から探求してきましたが、ふと振り返ると自分がBlueskyの提供するPDSに所属し、鍵管理すら自分でしていないことに気づきました。これは、私だけでなく他の多くのユーザーに当てはまります。これでは、ATProtocolのオープンプロトコルとしての強みを何一つ活かせていません。そこで私は、どうすればBlueskyの提供するインフラへの依存を最小限にできるか、理解しやすい小さな実装を用意できるかを考えました。

5.3　仕様・技術

まず、DIDについてです。ATProtocolにおいて、アイデンティティの基本単位はDIDです。Blueskyに通常の登録フローで登録すると、Bluesky側のPLCサーバーというアイデンティティを管理するサーバー上で鍵ペアが生成され、自動的にdid:plcというDIDが発行されます。鍵をBlueskyが持っているということは、本来は構造的に不可能であるはずの、Blueskyによるデータの改ざんなどが理論的には可能になってしまいます。

そもそも、Blueskyはdid:plcしかサポートしていないと考えられがちですが、実はdid:webもサポートしています。did:webとは、did:plcよりもシンプルなDIDメソッドで、対象のドメインの/.well-known/did.jsonに公開鍵情報などを置く、DNS認証に近いDIDメソッドです。興味深いことに、Blueskyのコアメンバーである Dan Abramov 氏は、did:plcについて、将来的に中央集権化を進め、did:webは分散を求める人のためのオプションになるだろう、と述べてい

[6]https://uit-inside.linecorp.com/episode/192

ます。[6]私自身も近い考えを持っており、利用方法がよりシンプルで管理しやすい did:web を採用することにしました。

次に、データ本体についてです。Bluesky を使っていると、PDS 内のデータ構造というのはブラックボックスになりがちですが、実際はただの Static なファイルです。もう少し具体的に言うと、ATProtocol のリポジトリ構造は Merkle Search Tree(MST)という構造を使うよう規定されていますが、それは論理的な構造であり、実際に配置されるデータは BlockStore と呼ばれるファイルハッシュを key、バイナリを value とするシンプルなデータ構造です。

ここで、私は PDS のブラックボックス的なイメージを解消するため、PDS を明示的に 2 層構造に分けることにしました。まず、データ層です。これは ATProtocol のデータスキーマに沿った任意のデータ(投稿、いいね、フォロー etc...)で、未署名の JSON です。次に、配信層です。これは、データ層のデータから MST を計算し、その結果に対して署名をした BlockStore 形式のビルド結果です。PDS はこの Static なビルド結果を配信する位置づけとします。

すると、多機能なデータベースサーバーに見えていた PDS が、実際はただハッシュ計算・署名をした結果を API で配信している HTTP サーバーに見えてきます。このように、私は自分の PDS に対する理解を深めるだけでなく、ATProtocol を学ぶ入門者にとってもわかりやすい実装を心がけました。

5.4 成果

これらの意思決定に従い PDS を実装し、実際に Bluesky に投稿を行うことができました(図 4)。また、その過程で今まで曖昧だった Firehose 周りの挙動や、BGS と連携をとるためのシーケンス番号の原則などについての理解を深めることができました。現在、段階的に Bluesky 側の PDS から自前実装の PDS に移行を進めています。

図 4: nijiiro PDS 上の投稿が Bluesky で表示されている様子