Getting Started

`@skyware/labeler` is a lightweight library for operating a labeler.

Before you start, you will need:

- An existing Bluesky account that will act as the labeler
- A domain
- An SSL certificate for your domain

Dev Note

Before you start, there's a few things to note. When you emit a label, that label is permanently stored by Bluesky and other AppViews. Deleting your database will not reset that. It will, however, make it very difficult for you to start emitting labels again, as you will be starting again from label 0, which AppViews think they've already seen.

It's recommended that you **should** use a separate account for development until you know your labeler is working, at which point you can create a new account and delete the database.

You **should** also only have a single database per labeler account. This means that if you're switching between development and production with the same account, make sure to copy the database file between machines.

You should **not** delete your database file unless you will not be using that account as a labeler anymore, in which case you should run `npx @skyware/labeler clear` afterwards to return it to a regular account.

Setup

If you already have a labeler set up, via Ozone or otherwise, you can skip this step.

To initialize the Bluesky account as a labeler, run the following command:

```
npx @skyware/labeler setup
```

The command will walk you through logging in, then ask for the URL where your labeler server will be located. This must be an HTTPS URL. A confirmation code will then be sent to your email. Once you've entered the code, a signing key will be generated if you did not provide one. Be sure to save this key, as you will need it to operate the labeler.

Once complete, you'll have the option to begin defining labels that your labeler will apply. You can also do this later by running `npx @skyware/labeler label add`. For each label, you must define the following information:

```
? Identifier (non-user-facing, must be unique, 100 characters max): » A unique
identifier for the label, used internally.
? Name (user-facing, 64 characters max): » The name that users will see for
the label.
? Description (user-facing): » A description of the label.
? Does the user need to have adult content enabled to configure this label? »
(y/N)
? Label severity:
    Informational - Displays an information symbol next to the label
    Alert - Displays a warning symbol next to the label
    None - The label will not be displayed on content
? Should this label hide content?
    Content - All content, text or media, will be hidden if labeled
    Media - Only media will be hidden if labeled
    None - The label will not hide content
? What should the default setting be for a new subscriber?
    Ignore - The label will not be displayed by default
    Warn - The label will be displayed on content, but will not hide it from
feeds
    Hide - The label will be displayed on content, and labeled content will be
hidden from feeds
```

Running the Labeler

```
import { LabelerServer } from "@skyware/labeler";

const server = new LabelerServer({
    did: process.env.LABELER_DID,
    signingKey: process.env.SIGNING_KEY
});

server.start(14831, (error) => {
    if (error) {
        console.error("Failed to start server:", error);
    } else {
        console.log("Labeler server running on port 14831");
    }
});
```

The `LabelerServer` class takes an object argument with the following properties:

- `did`: The DID of the labeler account.
- `signingKey`: The signing key used to sign labels. This should be the same key you received when setting up the labeler, if you used the setup CLI command.
- `auth` (optional): A function that takes a DID and returns a boolean indicating whether the DID is authorized to emit labels. If not provided, only the labeler account can emit labels.
- `dbPath` (optional): The path to the database file. Defaults to `./labels.db`.
- `dbUrl` (optional): The URL of the database. Use this instead of `dbPath` if you want to use a remote libSQL database.
- `dbToken` (optional): The authentication token for the database. Required if a remote `dbUrl` is provided.

After initializing the server, call the `start` method with the port number to listen on and a callback that will be called when the server starts. If an error occurs, the callback will be

called with the error.

Once your labeler is running, you will need to point your labeler's domain to the server and set up SSL. You can use a tool like Caddy to do this.

If all has gone well, you're ready to start labeling content!

```
await server.createLabel({
    // If you're labeling a record, this should be an at:// URI pointing to the record.
    // If you're labeling an account, this should be the account's DID.
    uri: "did:plc:427uaandx74txvzakxthrjwu",
    // Assuming you've previously defined a label with the identifier "cool-cat".
    val: "cool-cat",
});
```

To undo a label, emit the same label with the `neg` property set to `true` to `neg`ate the label.

```
await server.createLabel({
    uri: "did:plc:427uaandx74txvzakxthrjwu",
    val: "cool-cat",
    neg: true
});
```

Next, read on to learn about Automated Labeling.