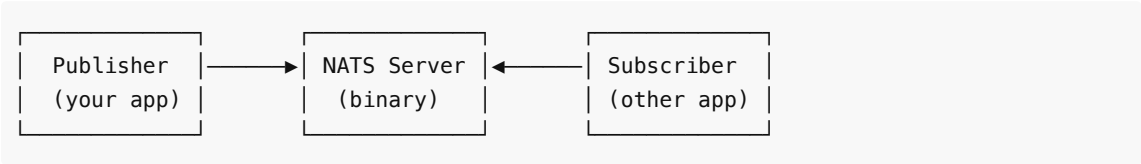


NATS 4 Noobs

A hands-on NATS training repository. Learn NATS messaging patterns progressively through a real-time wildlife observation globe.

What is NATS?

NATS is a **single binary** (`nats-server`) that acts as a message router. That's it.



- **Not a database** (unless you enable JetStream)
- **Not a framework**
- Just an ultra-fast message router

Repository Structure

```
nats_4_noobs/
├── nats/
│   └── docker-compose.yml      # NATS server (evolves per level)
├── inaturalist-watcher/
│   └── src/index.ts           # Polls iNaturalist, serves HTTP (lvl0) → publishes
├── NATS (lvl1)
│   ├── wildlive/
│   │   └── src/                # Bun + Hono globe app
│   ├── types/
│   │   └── observation.ts      # Shared Observation type
│   ├── Makefile               # Root commands: make watcher, make app, make dev
│   └── TRAINING.md
```

- **main branch:** Final state with all NATS features
- **lvlX branches:** Each level builds on the previous

Data Source

iNaturalist API v2 - `fields` param selects only needed data (~500B vs ~50KB/observation)

```
curl "https://api.inaturalist.org/v2/observations?
per_page=5&order=desc&photos=true&fields=id,species_guess,geojson,created_at,taxon.pre"
```

Types: `types/observation.ts`

Docs: <https://api.inaturalist.org/v2/docs/>

Training Levels

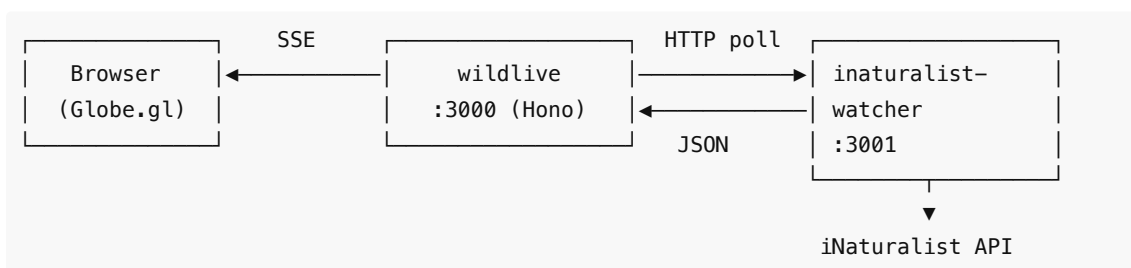
Level	Concept	Architecture	What You Learn
-------	---------	--------------	----------------

lvl0	Baseline	Watcher → HTTP → Wildlive	Starting point, no NATS yet
lvl1	Pub/Sub basics	Watcher → NATS → Wildlive	First NATS connection, decoupling
lvl2	Subjects	Points colored by taxon	Message routing
lvl3	Wildcards + filters	Toggle buttons per taxon	Dynamic subscribe/unsubscribe
lvl4	Request/Reply	Click point → species details	Request/response pattern
lvl5	Auth zero-trust	Role switcher (observer/dashboard/admin)	NKeys, permissions
lvl6	JetStream Stream	Replay button + pause/resume	Persistence, history
lvl7	JetStream KV	Live counters per taxon	Shared state

Level Details

lvl0 - Starting Point

- A service (`inaturalist-watcher :3001`) regularly produces data (wildlife observations)
- Our app (`wildlive :3000`) wants to display it on a 3D globe
- Current approach: **wildlive polls the watcher via HTTP** every 1s



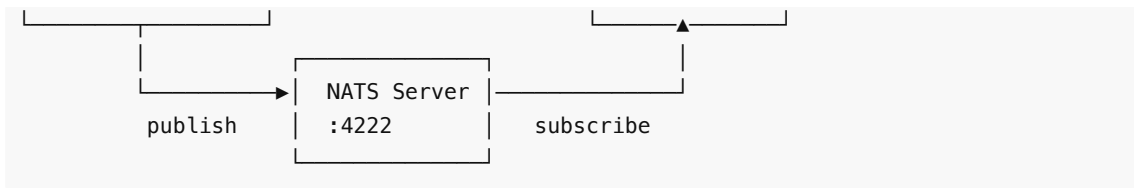
```

make watcher # Terminal 1
make app     # Terminal 2
# → http://localhost:3000
  
```

- It works, **but**:
 - Wildlive must know the watcher's URL → **coupling**
 - Only one consumer can drain the buffer → **no fan-out**
 - Polling = up to 1s latency → **not real-time**
 - If watcher restarts, buffer is lost → **no persistence**

lvl1 - Pub/Sub Basics





Goal: Watcher publishes to NATS, wildlive subscribes. No more HTTP between the two.

Checklist

- ☒ 1. NATS infrastructure already configured

The repo ships with everything needed to run NATS locally:

- `nats/docker-compose.yml` — NATS server container config
- `Makefile` — `make nats` (start) and `make nats-stop` (stop) targets
- `render.yaml` — production Blueprint with NATS as a private service (`pserv`)

```
# nats/docker-compose.yml
services:
  nats:
    image: nats:2.12-alpine
    container_name: nats
    ports:
      - "4222:4222" # Client connections
      - "8222:8222" # Monitoring UI
    command: ["--http_port", "8222"]
```

- ☐ 2. Start NATS + verify

```
make nats
# Open http://localhost:8222 → NATS monitoring dashboard
```

- ☐ 3. Install NATS client in both services

```
cd inaturalist-watcher && bun add nats
cd ../wildlive && bun add nats
```

- ☐ 4. inaturalist-watcher: publish to NATS instead of HTTP

`toObservation()` and `fetchObservations()` stay untouched. We modify `ingest()` and replace the **transport section**.

`nc` = NATS Connection. `sc` = StringCodec, encodes/decodes messages (NATS transports bytes).

Why bytes? NATS, Kafka, RabbitMQ — all transport opaque byte arrays (`Uint8Array`). Language agnostic + flexible. The broker never interprets content — it just moves bytes. Your app encodes (object → JSON string → bytes) and decodes (bytes → string → object).

In `inaturalist-watcher/src/index.ts` :

a) Add the `nats` import at the top:

```
import { connect, StringCodec } from 'nats'
```

b) Delete `let buffer: Observation[] = []` — observations go straight to NATS.

c) In `ingest()`, replace `buffer.push(...observations)` with:

```
for (const obs of observations) {  
  nc.publish('nature.observation', sc.encode(JSON.stringify(obs)))  
}
```

d) Replace `// --- HTTP transport (lvl0) ---` section (up to `// --- iNaturalist API ---`) with:

```
// --- NATS transport (lvl1) ---  
  
const nc = await connect({ servers: 'localhost:4222' })  
const sc = StringCodec()  
console.log(`[watcher] connected to NATS at ${nc.getServer()}`)  
  
setInterval(ingest, 15_000)  
ingest()
```

*nc and sc are used in `ingest()` above — works because `ingest()` is only **called** below, after initialization.*

- ☐ 5. wilddlive: subscribe instead of HTTP poll

`broadcast()` and Hono routes stay untouched. Only the data source changes.

In `wilddlive/src/index.tsx`:

a) Add the `nats` import at the top:

```
import { connect, StringCodec } from 'nats'
```

b) Replace `// --- Data source: HTTP poll (lvl0) ---` (the whole section + the `receiveObservations()` call) with:

```
// --- Data source: NATS subscribe (lvl1) ---  
  
const nc = await connect({ servers: 'localhost:4222' })  
const sc = StringCodec()  
  
async function receiveObservations() {  
  console.log(`[wilddlive] connected to NATS at ${nc.getServer()}`)  
  for await (const msg of nc.subscribe('nature.observation')) {  
    const obs = JSON.parse(sc.decode(msg.data)) as Observation  
    broadcast(obs)  
  }  
}
```

```
receiveObservations()
```

- ☐ 6. Test the full flow

```
# Terminal 1 - NATS
make nats

# Terminal 2 - Watcher (publishes to NATS)
make watcher

# Terminal 3 - Wildlive (subscribes from NATS)
make app
```

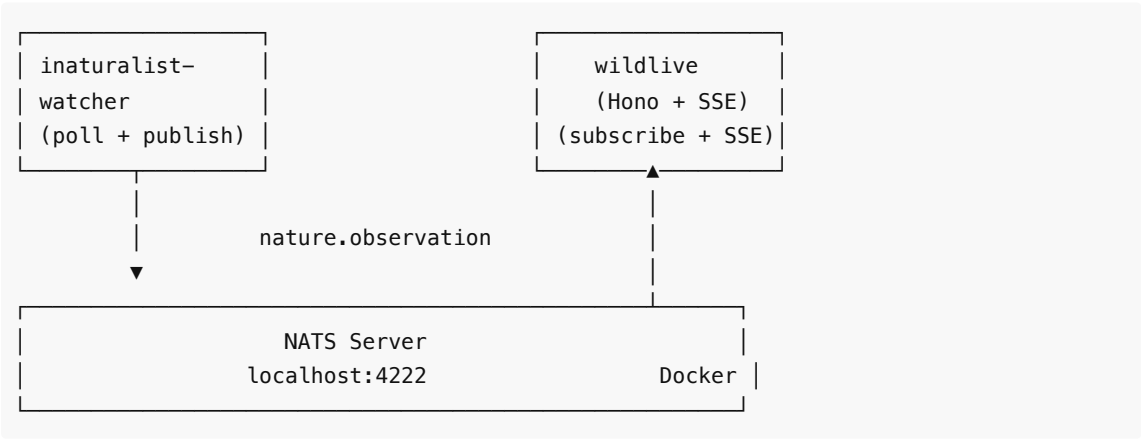
Verify:

- <http://localhost:3000> → Globe with worldwide wildlife observations
- <http://localhost:8222> → NATS monitoring (connections: 2, messages flowing)

Before / After

lv0 (HTTP)	lv1 (NATS)
Watcher exposes HTTP endpoint	Watcher publishes to NATS
Wildlive polls watcher via HTTP	Wildlive subscribes from NATS
Wildlive knows watcher URL → coupled	Neither knows the other → decoupled
One consumer drains buffer	Any number of subscribers
~1s delay (polling)	Real-time push (< 1ms)

Architecture lv1



lv2 - Subjects 📄



● bird
 ● insect
 ● mammal
 ● plant

- Publisher routes by taxon: `nature.Aves` , `nature.Insecta` , etc.
- Subscriber on `nature.>` receives all
- Points colored by subject/taxon
- Understand subject hierarchy

lvl3 - Subscription Filters 📝

[🐦 Birds] ON	[🦋 Insects] OFF	[🐾 Mammals] ON	[🌱 Plants] OFF
🌐 Globe (only birds + mammals visible)			

- Click "Insects OFF → ON" = `nc.subscribe('nature.Insecta')`
- Click "Birds ON → OFF" = `sub.unsubscribe()`
- Datastar manages reactive button state

lvl4 - Request/Reply 📝

🌐 Globe
 📍 ← click

📷 [Photo]
 American Robin
 Erithacus rubecula
 📍 Rennes, France
 👤 naturalist42

- Click → `nc.request('species.details', id)` → response with enriched data
- Responder service fetches additional info from iNaturalist API

lvl5 - Zero-Trust Auth 📝

Connected as: [observer ▼]

observer → can publish to nature.>

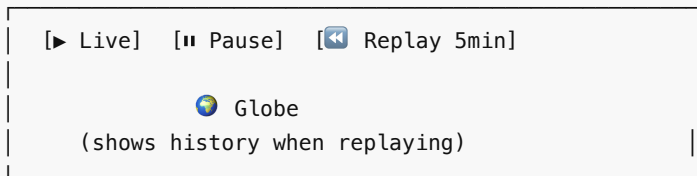
dashboard → can only subscribe

admin → full access

[Publish test] ← works as observer, fails as dash

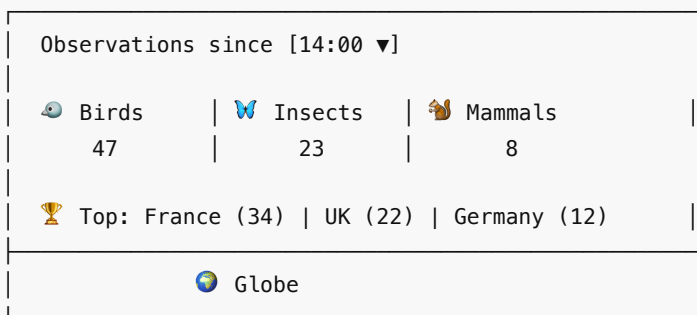
- Demonstrates NKeys/JWT auth
- Granular permissions per subject
- Switch roles to see permission errors

IvI6 - JetStream Stream 📄



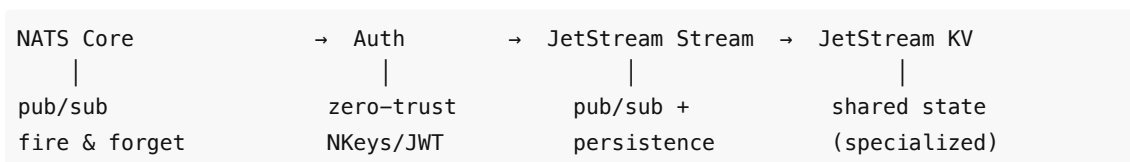
- Enable `--jetstream` flag
- Create NATURE stream on `nature.>`
- Replay: fetch observations from the last N minutes
- Pause/Resume: stop receiving, then catch up
- Demonstrates persistence and delivery policies

IvI7 - JetStream KV 📄



- KV keys: `count.Aves`, `count.Insecta`, `count.by_country.FR`
- `kv.watch('count.>')` for reactive updates
- Time selector resets counters

Key Concepts Progression



- **Core** = messages in flight, no storage
- **Auth** = who can pub/sub to what
- **Stream** = "What happened?" (event log, replay)
- **KV** = "What's the current state?" (last value)

NATS Topics Structure

```
nature.Aves      # Birds
nature.Insecta   # Insects
nature.Mammalia  # Mammals
nature.Plantae   # Plants
nature.Fungi     # Fungi
nature.>         # All observations (wildcard)
```

NATS Installation

Local Development

```
# Option 1: Binary
brew install nats-server # macOS
nats-server

# Option 2: Docker
docker run -p 4222:4222 nats:latest
```

With JetStream (lvl6+)

```
# Same binary, just add the flag
nats-server --jetstream

# Or Docker
docker run -p 4222:4222 nats:latest --jetstream
```

Production Options

1. **Synadia Cloud** (managed): `tls://connect.ngs.global`
2. **Self-hosted**: `nats-server` on your VPS with `systemd`
3. **Kubernetes**: Official Helm chart

Stack

- **Runtime**: Bun
- **Framework**: Hono
- **Frontend**: Datastar + Rocket (Web Components)
- **3D**: Globe.gl / Three.js
- **Messaging**: NATS

Development

```
# Install dependencies
make install

# lvl0: Two terminals
make watcher      # Terminal 1 - iNaturalist poller (:3001)
make app          # Terminal 2 - Globe app (:3000)
```

```
# lvl1+: Three terminals
make nats          # Terminal 1 - NATS server
make watcher       # Terminal 2 - Publisher
make app           # Terminal 3 - Subscriber

# Other commands
make nats-stop     # Stop NATS
make lint          # Biome lint
make test          # Unit tests
```