# **Lab: Streaming to MongoDB with a Resilient Sink (rpk connect)**

## **Goal**

Build a resilient Redpanda Connect pipeline that writes data to **MongoDB**, **automatically retries** transient failures, and **routes persistent failures** to a **Dead-Letter Queue (DLQ)**.

## **Purpose of the Lab**

Downstream systems (like DBs) can be unavailable or misconfigured. You’ll use Redpanda Connect’s **retry** and **fallback**wrappers to prevent data loss:

1. Simulate failure (typo in collection name) → observe retries → messages land in **DLQ**.
2. Fix config and re-run → data lands in MongoDB successfully.

## **Prerequisites**

* A **Redpanda Cloud** cluster and **rpk** profile (e.g., rpk-cloud), or any reachable Redpanda brokers.
* **Docker** and **Docker Compose** installed.
* **rpk** and **rpk connect** available on your machine.

## **Project Layout**

|  |
| --- |
| rp-mongo-connect-lab/ ├── user\_profiles.jsonl ├── mongo-sink.yaml ├── docker-compose.yml └── .env |

## **Part 1 — Local MongoDB**

### **Step 1: Prepare the Project**

|  |
| --- |
| mkdir rp-mongo-connect-lab cd rp-mongo-connect-lab |

Create **docker-compose.yml** (MongoDB + mongo-express UI):

|  |
| --- |
| version: "3.8"  services:  mongodb:  image: mongo:6  container\_name: mongo\_db\_resilience  restart: always  environment:  MONGO\_INITDB\_ROOT\_USERNAME: root  MONGO\_INITDB\_ROOT\_PASSWORD: example  ports:  - "27017:27017"   mongo-express:  image: mongo-express:latest  container\_name: mongo\_express\_ui  depends\_on:  - mongodb  environment:  ME\_CONFIG\_MONGODB\_SERVER: mongodb # <-- must match service name  ME\_CONFIG\_MONGODB\_AUTH\_DATABASE: admin  ME\_CONFIG\_MONGODB\_ADMINUSERNAME: root  ME\_CONFIG\_MONGODB\_ADMINPASSWORD: example  ME\_CONFIG\_BASICAUTH\_USERNAME: admin # for UI login  ME\_CONFIG\_BASICAUTH\_PASSWORD: admin  ports:  - "8083:8081" |

Start the database:

|  |
| --- |
| docker compose up -d docker compose up -d mongo-express docker logs -n 100 mongo\_express\_ui |

Now open **http://localhost:8083** and log in with:

* **UI Basic Auth:** admin / admin (from ME\_CONFIG\_BASICAUTH\_\*)
* **MongoDB connection:** uses root / example automatically from the env vars you set.

## **Part 2 — Simulate & Handle Failures**

### **Step 2: Prepare Redpanda & Data**

Create topics (Cloud profile shown; replace with your profile if different):

|  |
| --- |
| rpk topic create user-profiles --profile rpk-cloud rpk topic create user-profiles-dlq --profile rpk-cloud |

### **Create user\_profiles.jsonl:**

|  |
| --- |
| cat > user\_profiles.jsonl <<'EOF' {"profile\_id": 101, "email": "alice@example.com", "country": "US"} {"profile\_id": 102, "email": "bob@corp.com", "country": "CA"} EOF |

Create **.env** (fill with your Cloud creds):

|  |
| --- |
| RP\_BOOTSTRAP="d2heugc8ljef72usu9gg.any.ap-south-1.mpx.prd.cloud.redpanda.com:9092" RP\_USERNAME=aman RP\_PASSWORD=jJOSBV0ng8qgpfg8NOMlk4iS7EIbEv MONGO\_HOST=localhost MONGO\_PORT=27017 MONGO\_USER=root MONGO\_PASS=example MONGO\_DB=redpanda\_lab |

Produce data:

|  |
| --- |
| rpk topic produce user-profiles --profile rpk-cloud < user\_profiles.jsonl |

### **Step 3: Create the Resilient Pipeline (with an intentional failure)**

Create **mongo-sink.yaml**. We’ll intentionally use a **bad collection name** (user\_profilessss) to force errors:

|  |
| --- |
| pipeline:  processors: []  input:  kafka:  addresses:  - ${RP\_BOOTSTRAP}  topics:  - user-profiles  consumer\_group: mongo-sink-group  start\_from\_oldest: true  tls:  enabled: true  sasl:  mechanism: SCRAM-SHA-256  user: ${RP\_USERNAME}  password: ${RP\_PASSWORD}  output:  mongodb:  url: "mongodb://${MONGO\_USER}:${MONGO\_PASS}@${MONGO\_HOST}:${MONGO\_PORT}/?authSource=admin"  database: ${MONGO\_DB}  collection: user\_profiles  # hyphenated operation name  operation: insert-one  # write\_concern must be an object  write\_concern:  w: majority  document\_map: |  root = {  "profile\_id": this.profile\_id,  "email": this.email,  "country": this.country  } |

Run the connector:

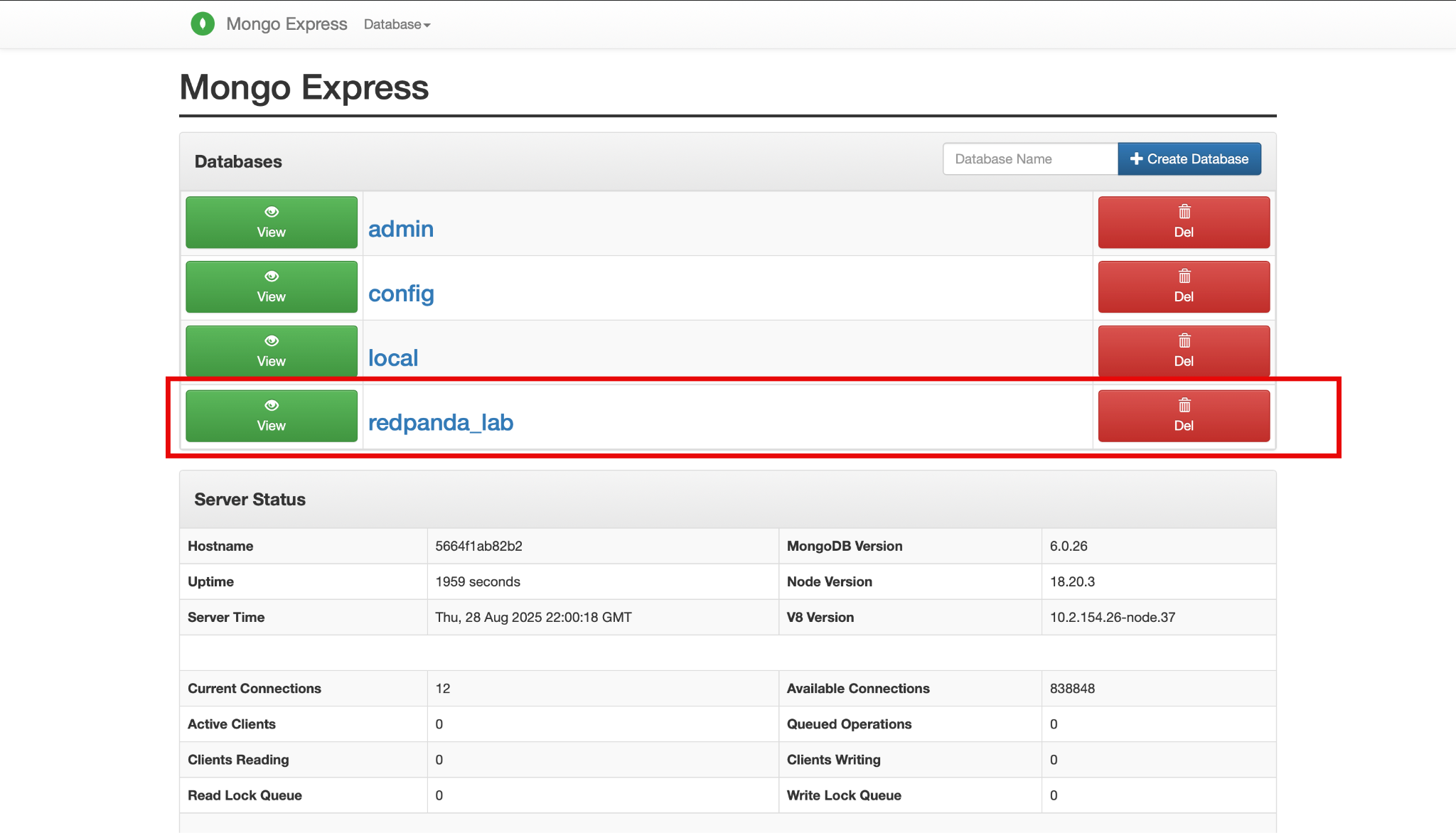
|  |
| --- |
| rpk connect run --env-file .env ./mongo-sink.yaml |

**Observe logs**: You’ll see errors about writes failing (wrong collection), retried 3 times, then messages routed to **DLQ**.

### **Step 5: Verify in MongoDB**

**Via mongo-express (UI):**

* Open http://localhost:8083 → find DB redpanda\_db → collection user\_profiles.



* You should see both documents.

**Via mongosh inside container (alternate):**

|  |
| --- |
| docker exec -it mongo\_db\_resilience mongosh -u root -p example --authenticationDatabase admin # inside mongosh: use redpanda\_lab db.user\_profiles.find().limit(10) db.user\_profiles.countDocuments() |

## 

## **Troubleshooting**

* **DLQ is empty but inserts fail**: Keep the connector running; it will retry up to 3 times first, then route to DLQ.
* **Can’t connect to MongoDB**: Confirm MONGO\_URI points to mongodb://root:example@localhost:27017. If you run Connect inside Docker, use mongodb:27017.
* **Nothing in MongoDB**: Confirm you **fixed the collection name** and **changed the consumer group** to reread from earliest.
* **TLS/SASL mismatch**: If your Redpanda Cloud cluster uses SCRAM-SHA-512 instead of 256, update the mechanism accordingly.

**Cleanup**

|  |
| --- |
| # Stop MongoDB services docker compose down  # Remove topics (optional) rpk topic delete user-profiles user-profiles-dlq --profile rpk-cloud |

## **What you learned**

* Designing a **resilient sink** with **retry + DLQ** using Redpanda Connect.
* Observing failure → DLQ → fix → success without losing data.
* Verifying end-to-end with **Console**, **DLQ consumption**, and **MongoDB UI/CLI**.