**Lab: Streaming to PostgreSQL with a Resilient JDBC Sink (rpk connect)**

**Goal:** Build a resilient Redpanda Connect pipeline that attempts to write data to a PostgreSQL database, automatically retries on transient failures, and routes persistently failing messages to a Dead-Letter Queue (DLQ).

# **Purpose of the Lab**

This lab simulates a common real-world scenario: a downstream system (like a database) is temporarily unavailable or misconfigured. You will learn how to use Redpanda Connect's retry and fallback output wrappers to handle these failures gracefully. This prevents data loss and makes your pipeline robust against intermittent issues. By first simulating a failure and observing the retry/DLQ behavior, and then fixing the configuration to see the successful outcome, you will gain practical experience in building fault-tolerant data sinks.

# **Prerequisites**

* A Redpanda Cloud account with a running cluster and an rpk profile (e.g., rpk-cloud).
* Docker and Docker Compose installed on your local machine.
* The rpk CLI and rpk connect binary installed on your local machine.

# **Project Layout**

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

# **Part 1: Setting up a Local Database**

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

1. **Create the project directory:**

|  |
| --- |
| mkdir rp-postgres-connect-lab cd rp-postgres-connect-lab |

1. **Create a Docker Compose file for PostgreSQL:** Create docker-compose.yml. This will run a local database instance.

|  |
| --- |
| services:  postgres:  image: postgres:14  container\_name: postgres\_db\_resilience  environment:  POSTGRES\_USER: redpanda\_user  POSTGRES\_PASSWORD: redpanda\_password  POSTGRES\_DB: redpanda\_db  ports:  - "5432:5432" |

1. **Start the database:**

|  |
| --- |
| docker compose up -d |

# **Part 2: Simulating and Handling Failures**

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

1. **Get Cloud Credentials:** Create a new user (e.g., resilience-user) in the **Security -> Users** tab of the Redpanda Cloud UI and grant it **Allow All** permissions in the **ACLs** tab. Save the **Username**, **Password**, and your cluster's **Broker Address**.
2. **Create the topics on Redpanda Cloud:**

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

1. **Create a sample data file named user\_profiles.jsonl:**

|  |
| --- |
| {"profile\_id": 101, "email": "alice@example.com", "country": "US"} {"profile\_id": 102, "email": "bob@corp.com", "country": "CA"} |

1. **Create the Environment File (.env):**

|  |
| --- |
| # .env REDPANDA\_BROKERS="<YOUR\_BROKERS\_URL>" REDPANDA\_USER="<YOUR\_USERNAME>" REDPANDA\_PASS="<YOUR\_PASSWORD>" |

Populate it with your cloud credentials.

1. **Produce the data to the user-profiles topic:**

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

## **Step 3: Create the Resilient Pipeline Configuration**

Create a file named postgres-sink.yaml. **Crucially, we will introduce a typo in the table name (user\_profilessss) to force a failure.**

**postgres-sink.yaml**

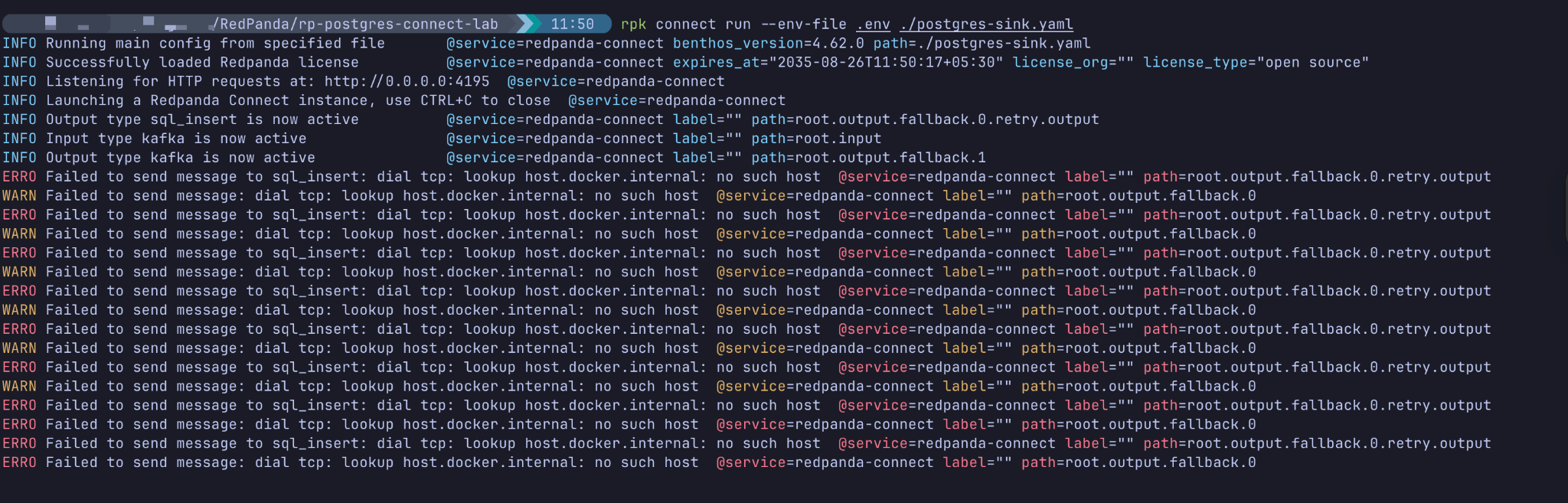
|  |
| --- |
| # postgres-sink.yaml input:  kafka:  addresses: [ "${REDPANDA\_BROKERS}" ]  topics: [ "user-profiles" ]  consumer\_group: "postgres-sink-group"  start\_from\_oldest: true  tls: { "enabled": true }  sasl:  mechanism: "SCRAM-SHA-256"  user: "${REDPANDA\_USER}"  password: "${REDPANDA\_PASS}"  output:  fallback:  - retry:  max\_retries: 3  backoff:  initial\_interval: 1s  max\_interval: 5s  output:  sql\_insert:  driver: postgres  dsn: postgresql://redpanda\_user:redpanda\_password@localhost:5432/redpanda\_db?sslmode=disable # <-- CORRECTED  table: user\_profilessss # <-- INTENTIONAL TYPO TO CAUSE FAILURE  columns: [ "profile\_id", "email", "country" ]  args\_mapping: 'root = [this.profile\_id, this.email, this.country]'  - kafka: # This is the Dead-Letter Queue  addresses: [ "${REDPANDA\_BROKERS}" ]  topic: "user-profiles-dlq"  tls: { "enabled": true }  sasl:  mechanism: "SCRAM-SHA-256"  user: "${REDPANDA\_USER}"  password: "${REDPANDA\_PASS}" |

## **Step 4: Run the Pipeline and Observe Failure**

1. Execute the connector.

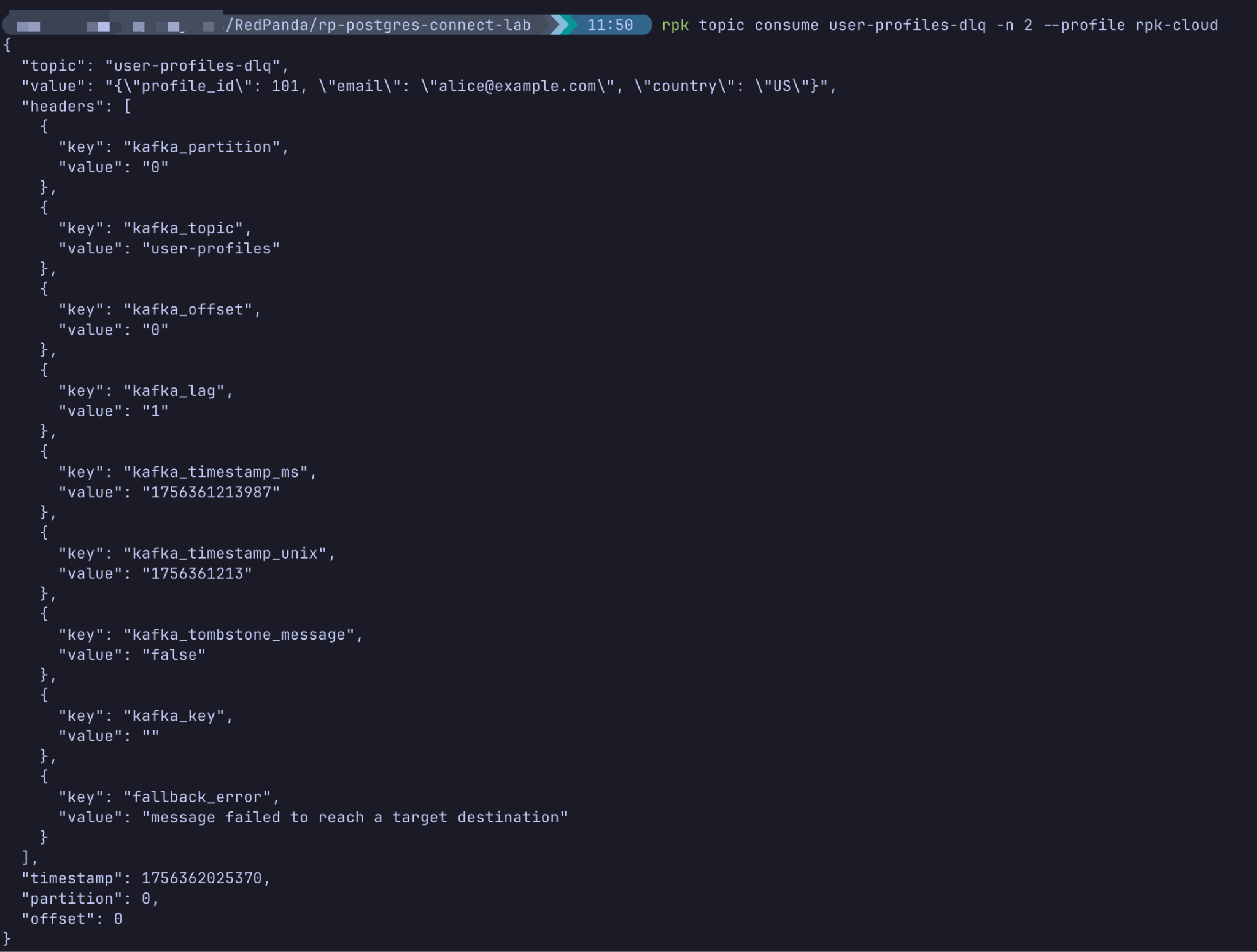
|  |
| --- |
| rpk connect run --env-file .env ./postgres-sink.yaml |

1. **Observe the logs:** You will see error messages indicating the table user\_profilessss does not exist, followed by logs showing that it is retrying the operation. After 3 retries, the process will give up and send the messages to the DLQ.



1. **Open a new terminal window.**
2. **Verify the DLQ:** Consume from the DLQ topic. You will find the two user profile messages that failed to be inserted.

|  |
| --- |
| rpk topic consume user-profiles-dlq -n 2 --profile rpk-cloud |



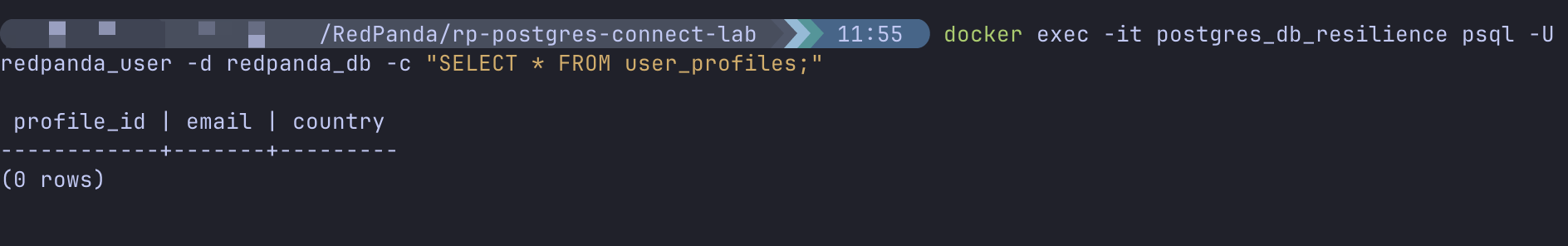
# 

# **Part 3: Fixing the Pipeline and Verifying Success**

## **Step 5: Create the Table and Correct the Configuration**

1. Stop the connector with Ctrl+C in the first terminal.
2. **Create the target table in PostgreSQL:**

|  |
| --- |
| docker exec -it postgres\_db\_resilience psql -U redpanda\_user -d redpanda\_db -c "CREATE TABLE user\_profiles (profile\_id INT PRIMARY KEY, email VARCHAR(255), country VARCHAR(2));" |



1. **Edit postgres-sink.yaml** to fix the table name and change the consumer group. Because the failed messages were sent to the DLQ, the consumer group committed their offsets. To force the connector to re-read from the beginning, we must change its name.

|  |
| --- |
| # ... consumer\_group: "postgres-sink-group-2" # <-- CHANGE THE GROUP NAME # ... table: user\_profiles # <-- CORRECTED # ... |

## **Step 6: Re-run the Pipeline and Verify**

1. Run the corrected connector. It will now connect as a new consumer group and re-process the original messages from the user-profiles topic.

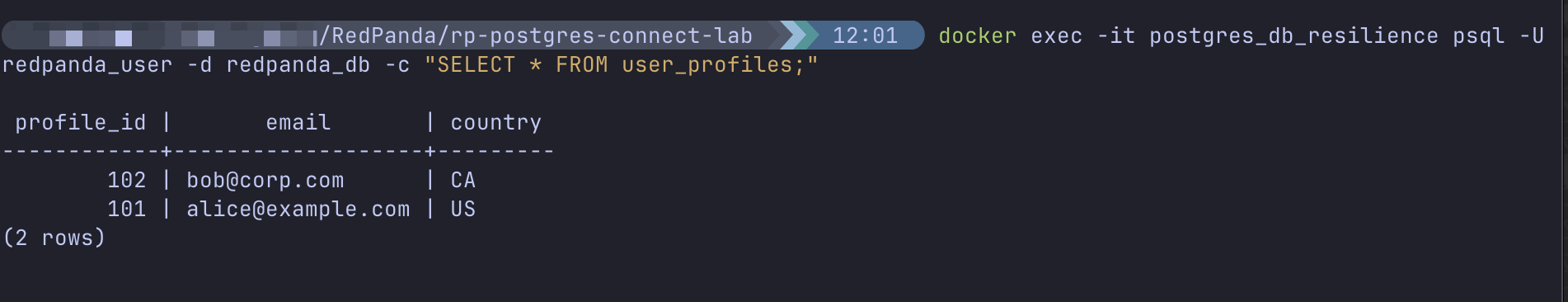
|  |
| --- |
| rpk connect run --env-file .env ./postgres-sink.yaml |

1. **Observe the logs:** This time, you should see no errors. The connector will process the messages and wait.
2. **Open a new terminal window.**
3. **Verify in the database:** Connect to the database and select the data.

|  |
| --- |
| docker exec -it postgres\_db\_resilience psql -U redpanda\_user -d redpanda\_db -c "SELECT \* FROM user\_profiles;" |

**Expected output:** You should see the two user records now successfully inserted into the table.

|  |
| --- |
| profile\_id | email | country ------------+-------------------+---------  101 | alice@example.com | US  102 | bob@corp.com | CA (2 rows) |



# Cleanup

1. Stop the connector with Ctrl+C.
2. Stop and remove the database container:

|  |
| --- |
| docker compose down |

1. Delete the Redpanda topics:

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