**Lab: Handling Poison Pills with a Python Application**

**Goal:** Build a resilient Python streaming application that consumes a raw data stream, uses a try/except block to identify and isolate malformed "poison pill" messages, and routes them to a Dead-Letter Queue (DLQ) topic for later analysis.

# **Purpose of the Lab**

This lab teaches a critical pattern for data quality and application stability directly in your code. You will write a Python consumer that acts as a microservice, validating incoming data. By catching exceptions during JSON parsing, your application will gracefully handle errors, ensuring that good data is forwarded for further processing while bad data is safely quarantined in a DLQ for inspection. This demonstrates how to build robust, fault-tolerant consumers that are essential for reliable data streaming systems.

# **Prerequisites**

* A Redpanda Cloud account with a running cluster.
* An rpk profile configured to connect to your cloud cluster (e.g., rpk-cloud).
* Python 3 is installed on your local machine.

# Project Layout

You will create a new directory for this lab.

|  |
| --- |
| rp-python-resilience-lab/ ├── messages.jsonl ├── poison\_pill\_handler.py └── .env |

# **Part 1: Setting up the Environment**

## **Step 1: Get Cloud Credentials**

If you don't already have them, 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**.

## **Step 2: Prepare the Project**

1. **Create the project directory:**

|  |
| --- |
| mkdir rp-python-resilience-lab cd rp-python-resilience-lab |

1. **Create a sample data file named messages.jsonl:** This file contains a mix of valid JSON and one malformed line (the "poison pill").

**messages.jsonl**

|  |
| --- |
| {"id": 1, "payload": "first message"} {"id": 2, "payload": "second message"} {"id": 3, "payload": "this is not valid json {"id": 4, "payload": "fourth message"} |

1. **Create the topics on Redpanda Cloud:**

|  |
| --- |
| rpk topic create raw-json-stream --profile rpk-cloud rpk topic create processed-json-stream --profile rpk-cloud rpk topic create json-stream-dlq --profile rpk-cloud |

1. **Set up the Python virtual environment:**

|  |
| --- |
| python3 -m venv venv source venv/bin/activate pip install kafka-python python-dotenv certifi python-snappy |

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

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

1. Populate the file with your credentials.  
   **Produce the raw data to the raw-json-stream topic:**

|  |
| --- |
| rpk topic produce raw-json-stream --profile rpk-cloud < messages.jsonl |

# 

# **Part 2: Building and Running the Resilient Python Application**

## **Step 3: Create the Python Handler Script**

Create a file named poison\_pill\_handler.py. This script will act as a long-running service. It consumes messages, tries to parse them as JSON, and then produces them to the appropriate output topic.

**poison\_pill\_handler.py**

|  |
| --- |
| # poison\_pill\_handler.py import json, os, certifi from kafka import KafkaConsumer, KafkaProducer from dotenv import load\_dotenv  # Load environment variables from .env file load\_dotenv()  # --- Configuration --- BROKER\_URL = os.getenv("REDPANDA\_BROKERS") USERNAME = os.getenv("REDPANDA\_USER") PASSWORD = os.getenv("REDPANDA\_PASS") SOURCE\_TOPIC = "raw-json-stream" PROCESSED\_TOPIC = "processed-json-stream" DLQ\_TOPIC = "json-stream-dlq" # --- End Configuration ---  print("Starting resilient Python handler...") print("Press Ctrl+C to stop.")  try:  # Initialize both a consumer and a producer  consumer = KafkaConsumer(  SOURCE\_TOPIC,  bootstrap\_servers=BROKER\_URL, security\_protocol="SASL\_SSL",  sasl\_mechanism="SCRAM-SHA-256", sasl\_plain\_username=USERNAME, sasl\_plain\_password=PASSWORD,  group\_id="python-poison-pill-handler-group", auto\_offset\_reset="earliest",  ssl\_cafile=certifi.where(), api\_version=(2, 0, 2)  )   producer = KafkaProducer(  bootstrap\_servers=BROKER\_URL, security\_protocol="SASL\_SSL",  sasl\_mechanism="SCRAM-SHA-256", sasl\_plain\_username=USERNAME, sasl\_plain\_password=PASSWORD,  ssl\_cafile=certifi.where(), api\_version=(2, 0, 2)  )   for message in consumer:  try:  # Try to parse the message value as JSON  json\_payload = json.loads(message.value)  print(f"Successfully parsed message: {json\_payload}")    # If successful, send the raw bytes to the processed topic  producer.send(PROCESSED\_TOPIC, value=message.value)   except json.JSONDecodeError:  # If JSON parsing fails, it's a "poison pill"  print(f"ERROR: Malformed JSON detected. Sending to DLQ: {message.value}")    # Send the original, raw message bytes to the DLQ topic  producer.send(DLQ\_TOPIC, value=message.value)    producer.flush()  except KeyboardInterrupt:  print("Handler stopped by user.") except Exception as e:  print(f"An unexpected error occurred: {e}") finally:  if 'consumer' in locals():  consumer.close()  if 'producer' in locals():  producer.close()  print("Consumer and producer closed.") |

## **Step 4: Run the Application**

Execute the script. It will process the four existing messages and then wait for more.

|  |
| --- |
| python3 poison\_pill\_handler.py |

**Expected output:** You will see logs indicating that three messages were parsed successfully and one malformed message was sent to the DLQ.

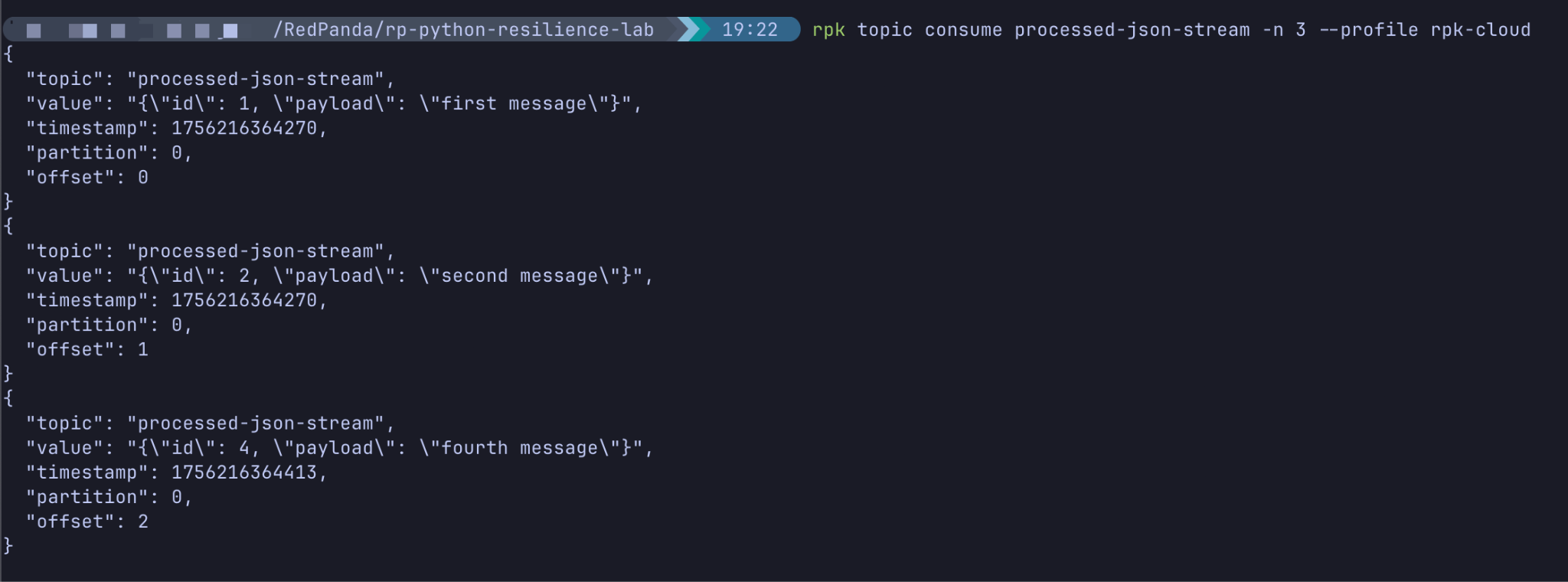
## **Step 5: Verify the Results**

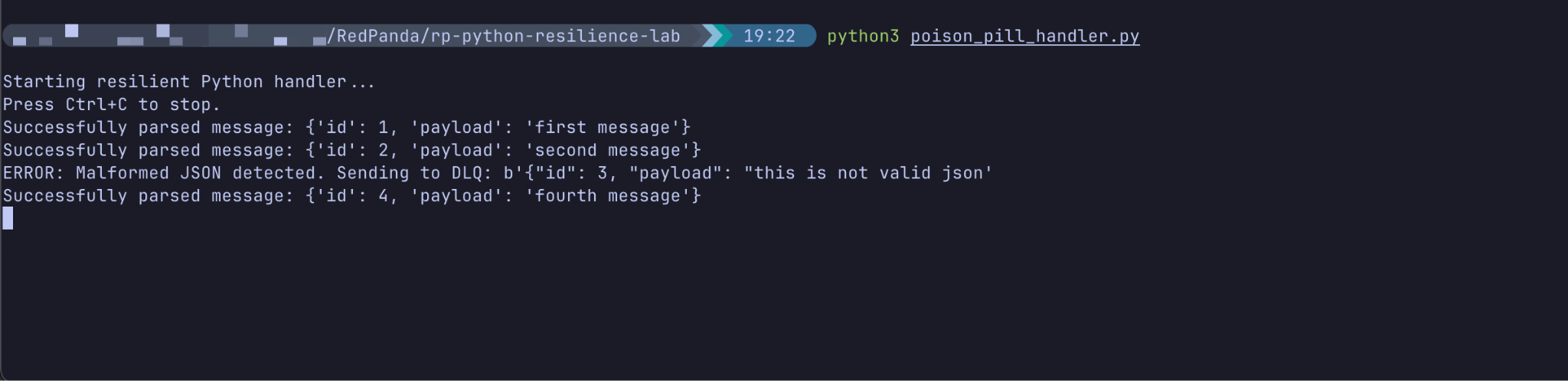
1. **Open a new terminal window** and activate the virtual environment (source venv/bin/activate).
2. **Check the "good" topic:** Consume from processed-json-stream. You should see only the three valid messages.

|  |
| --- |
| rpk topic consume processed-json-stream -n 3 --profile rpk-cloud |

1. **Check the Dead-Letter Queue:** Consume from json-stream-dlq. You should see only the original, malformed "poison pill" message.

|  |
| --- |
| rpk topic consume json-stream-dlq -n 1 --profile rpk-cloud |





# **Cleanup**

1. Stop the Python script by pressing Ctrl+C in the first terminal.
2. Delete the topics:

|  |
| --- |
| rpk topic delete raw-json-stream processed-json-stream json-stream-dlq --profile rpk-cloud |

1. Deactivate the virtual environment.

|  |
| --- |
| deactivate |