Lab 9: Resilient CDC from PostgreSQL

**Goal:** Build a fault-tolerant Change Data Capture (CDC) pipeline using Flink SQL. This pipeline will stream changes in real-time from a PostgreSQL database and demonstrate Flink's powerful ability to handle database schema changes without interrupting the data flow.

# 

# INDEX

[**Purpose of this Lab**](#_94blrtbsvlxt) **3**

[**Prerequisites**](#_two4zjt6p6gq) **3**

[**Project Structure**](#_81hzi9gaeyqk) **3**

[**Part 1: Understanding the Versioning Change**](#_3ir3smq3dh5v) **4**

[**Part 2: Project and Connector Setup**](#_bfb2j885lzra) **4**

[**Part 3: Configuring the Docker Environment**](#_w0tup8p7qp1u) **5**

[**Part 4: Developing the Flink SQL CDC Script**](#_xbplfwmxv32m) **8**

[**Part 5: Executing the End-to-End Pipeline**](#_g9aki8f4jt3l) **9**

[**Part 6: Verification and Schema Evolution**](#_g93z5f4q5uzp) **12**

[**Part 7: Cleanup**](#_g6fmnvodnvqz) **15**

[**Part 8: Next Steps**](#_z8jqdfh45fls) **16**

# 

# Purpose of this Lab

This lab introduces one of the most powerful features in modern data engineering: Change Data Capture (CDC). You will learn how to use the Flink SQL CDC connector to capture every INSERT, UPDATE, and DELETE operation from a PostgreSQL table and turn it into a continuous stream of events.

The primary objective is to observe Flink's resilience to upstream schema evolution. You will start a streaming job, then alter the source PostgreSQL table by adding a new column. You will see firsthand how Flink detects and adapts to this change automatically, a critical feature for building robust, low-maintenance data pipelines.

By completing this lab, you will:

Implement a CDC Pipeline: Use the Flink SQL connector to capture real-time changes from a relational database.

Understand CDC Event Structure: See how row-level database changes are represented in a data stream.

Manage Schema Evolution: Witness Flink's ability to handle ALTER TABLE commands on the source without requiring a job restart.

Work with a Containerized Flink Environment: Gain experience running a complete Flink stack (JobManager, TaskManager, Database) using Docker Compose.

# Prerequisites

An Ubuntu-based environment with Docker and Docker Compose installed.

Basic familiarity with SQL commands (INSERT, UPDATE, ALTER TABLE).

Completion of previous labs is recommended but not strictly required, as this lab runs in a self-contained Docker environment.

# Project Structure

By the end of this lab, your project directory will be structured as follows:

|  |
| --- |
| ~/flink-lab-9/ ├── jars/ │ ├── flink-sql-connector-postgres-cdc-2.4.2.jar # CDC connector │ └── postgresql-42.6.0.jar # JDBC driver ├── postgres/ │ └── init.sql # Initial database schema script ├── sql/ │ └── cde\_pipeline.sql # The Flink SQL job script └── docker-compose.yaml # Defines all services |

# Part 1: Understanding the Versioning Change

For Labs 1-8, we used Apache Flink 2.0.0. However, as of this writing, the Flink CDC connectors have not been updated to support this latest version. To use this critical feature, we must temporarily use an older, compatible version of Flink. This lab will use **Flink 1.18**, deployed entirely within Docker, ensuring it does not interfere with the Flink 2.0.0 installation used for other labs. This approach mirrors real-world scenarios where engineers must select specific tool versions to match the available connector ecosystem.

# Part 2: Project and Connector Setup

**Step 1: Create the Project Directory**

|  |
| --- |
| mkdir -p ~/flink-lab-9/{jars,postgres,sql} cd ~/flink-lab-9 |

****

**Step 2: Download the Required Connectors**

We need two JAR files: the PostgreSQL CDC connector for Flink and the standard PostgreSQL JDBC driver.

Download the Flink CDC Connector for Postgres

|  |
| --- |
| wget -P ./jars/ https://repo.maven.apache.org/maven2/com/ververica/flink-sql-connector-postgres-cdc/2.4.2/flink-sql-connector-postgres-cdc-2.4.2.jar |



Download the PostgreSQL JDBC Driver

|  |
| --- |
| wget -P ./jars/ https://repo.maven.apache.org/maven2/org/postgresql/postgresql/42.6.0/postgresql-42.6.0.jar |



Note: These files will be mounted into our Flink containers by Docker Compose, making them available to the job.

# Part 3: Configuring the Docker Environment

**Step 1: Create the PostgreSQL Initialization Script**

This script will run when the PostgreSQL container first starts. It creates a customers table and enables the logical replication Flink needs for CDC.

Create the file postgres/init.sql:

|  |
| --- |
| code postgres/init.sql |

Add the following content:

|  |
| --- |
| -- postgres/init.sql CREATE TABLE customers (  id SERIAL PRIMARY KEY,  first\_name VARCHAR(255),  last\_name VARCHAR(255),  email VARCHAR(255) );  ALTER TABLE customers REPLICA IDENTITY FULL;  INSERT INTO customers (first\_name, last\_name, email) VALUES ('John', 'Doe', 'john.doe@example.com'); |

**Step 2: Define the Docker Compose Services**

Create the main docker-compose.yaml file. This file defines our entire lab environment: a PostgreSQL database, a Flink JobManager, and a Flink TaskManager.

|  |
| --- |
| code docker-compose.yaml |

Add the following content:

|  |
| --- |
| # docker-compose.yaml services:  postgres:  image: postgres:15  container\_name: postgres\_db  ports:  - "5432:5432"  environment:  - POSTGRES\_USER=flinkuser  - POSTGRES\_PASSWORD=flinkpassword  - POSTGRES\_DB=flinkdb  volumes:  - ./postgres/init.sql:/docker-entrypoint-initdb.d/init.sql  command:  - "postgres"  - "-c"  - "wal\_level=logical"   jobmanager:  image: flink:1.18.0-scala\_2.12-java11  container\_name: flink\_jobmanager  ports:  - "8081:8081"  command: jobmanager  environment:  - |  FLINK\_PROPERTIES=  jobmanager.rpc.address: jobmanager  parallelism.default: 2  volumes:  - ./jars:/opt/flink/usrlib # Mount jars to the user library folder   taskmanager:  image: flink:1.18.0-scala\_2.12-java11  container\_name: flink\_taskmanager  depends\_on:  - jobmanager  command: taskmanager  environment:  - |  FLINK\_PROPERTIES=  jobmanager.rpc.address: jobmanager  taskmanager.numberOfTaskSlots: 2  volumes:  - ./jars:/opt/flink/usrlib # Mount jars to the user library folder |

# Part 4: Developing the Flink SQL CDC Script

This SQL script contains all the logic for our pipeline. It defines the source table (the CDC connection to Postgres), a sink table (for printing results to the log), and the query that connects them.

Create the file sql/cdc\_pipeline.sql:

|  |
| --- |
| code sql/cdc\_pipeline.sql |

Add the following content:

|  |
| --- |
| -- sql/cdc\_pipeline.sql  -- Set the SQL dialect to default, which is required for CDC jobs SET 'sql-client.execution.result-mode' = 'tableau';  -- Define the source table using the postgres-cdc connector CREATE TABLE customers\_cdc (  id INT,  first\_name STRING,  last\_name STRING,  email STRING,  -- The PRIMARY KEY is crucial for the connector to correctly interpret UPDATEs  PRIMARY KEY (id) NOT ENFORced ) WITH (  'connector' = 'postgres-cdc',  'hostname' = 'postgres', -- Service name from docker-compose  'port' = '5432',  'username' = 'flinkuser',  'password' = 'flinkpassword',  'database-name' = 'flinkdb',  'schema-name' = 'public',  'table-name' = 'customers',  'decoding.plugin.name' = 'pgoutput',  'slot.name' = 'flink\_cdc\_slot' -- Add this required option );  -- Define a sink table that prints results to the TaskManager logs CREATE TABLE print\_sink (  id INT,  first\_name STRING,  last\_name STRING,  email STRING,  -- Add a field for the new column we will add later  phone\_number STRING ) WITH (  'connector' = 'print' );  -- The main pipeline query. It streams all data from the source to the sink. -- Flink handles schema evolution, so we can select columns that don't exist yet. INSERT INTO print\_sink SELECT  id,  first\_name,  last\_name,  email,  -- This column does not exist yet! We will project a NULL for it.  CAST(NULL AS STRING) AS phone\_number FROM customers\_cdc; |

# Part 5: Executing the End-to-End Pipeline

**Step 1: Start the Docker Environment**

From the ~/flink-lab-9 directory, launch all services.

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

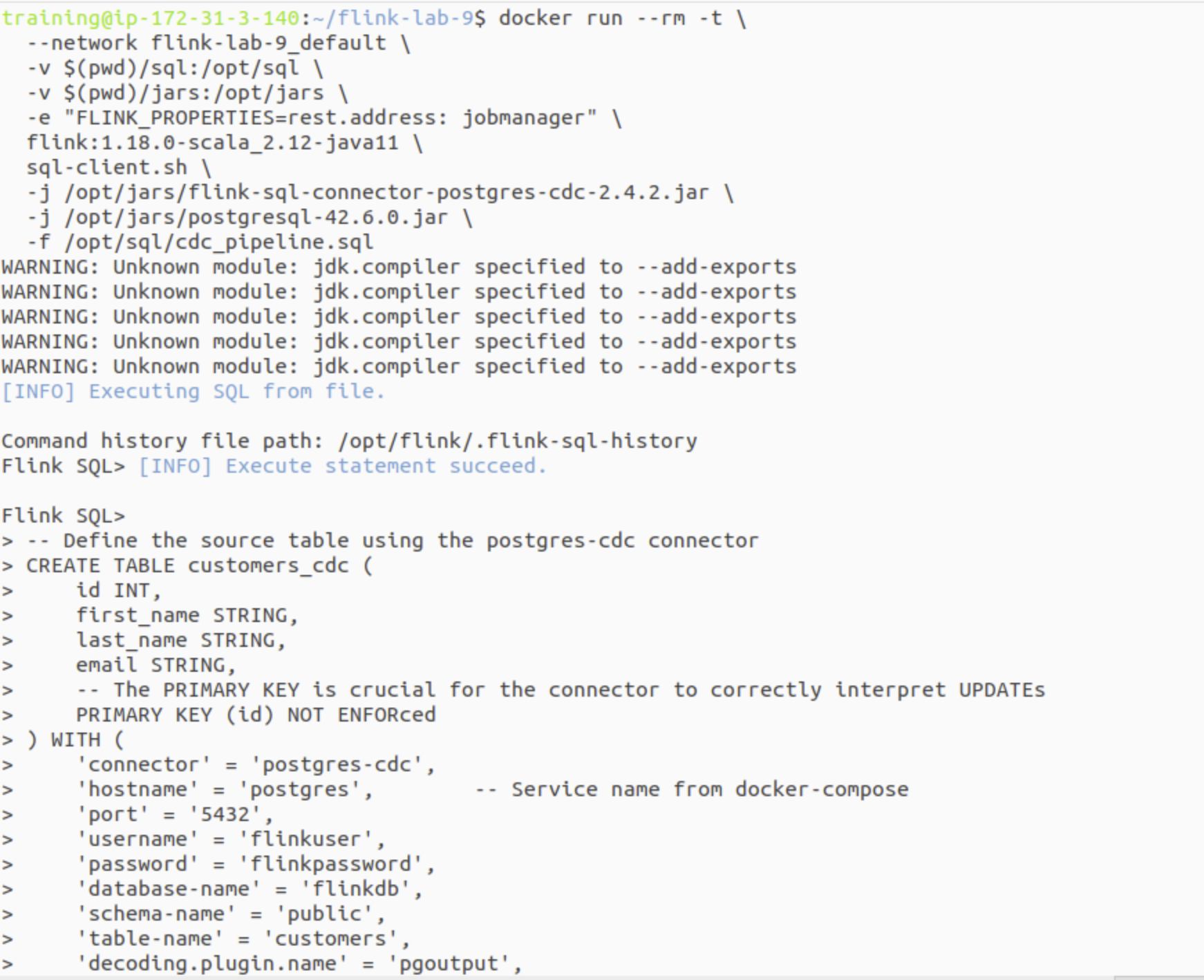


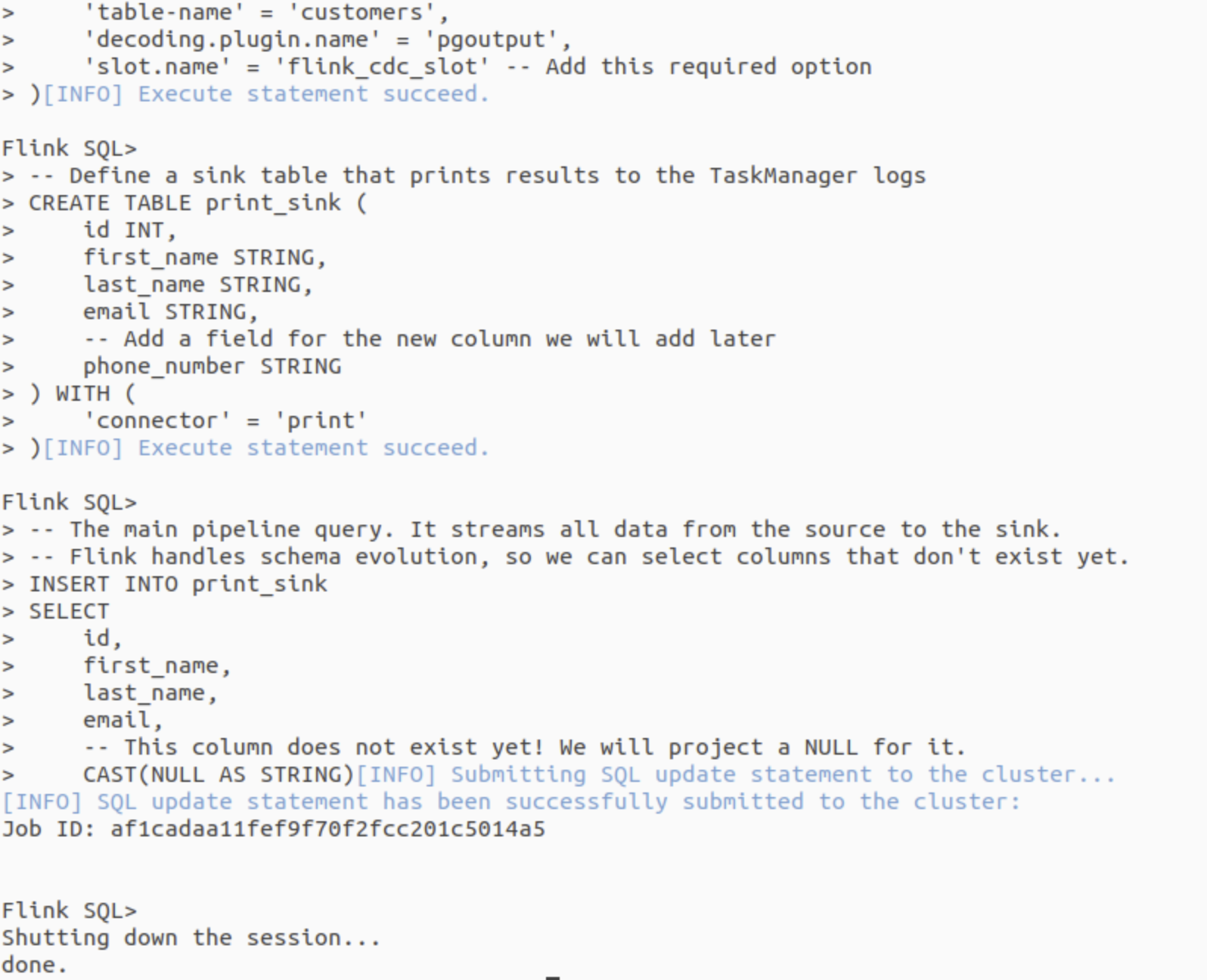
This will start Postgres, the Flink JobManager, and the Flink TaskManager in the background. It may take a minute for all containers to be healthy.

**Step 2: Submit the Flink SQL Job**

We will use Flink's sql-client running inside a temporary container to submit our SQL script to the running cluster.

|  |
| --- |
| docker run --rm -t \  --network flink-lab-9\_default \  -v $(pwd)/sql:/opt/sql \  -v $(pwd)/jars:/opt/jars \  -e "FLINK\_PROPERTIES=rest.address: jobmanager" \  flink:1.18.0-scala\_2.12-java11 \  sql-client.sh \  -j /opt/jars/flink-sql-connector-postgres-cdc-2.4.2.jar \  -j /opt/jars/postgresql-42.6.0.jar \  -f /opt/sql/cdc\_pipeline.sql |





Note: This command connects to the same Docker network as our cluster, mounts the necessary SQL file and connector JARs, and then submits the job before exiting. The job continues running on the cluster.

# Part 6: Verification and Schema Evolution

**Step 1: Check the Flink UI and Logs**

Open the Flink UI in your browser at http://localhost:8081. You should see your SQL job running.

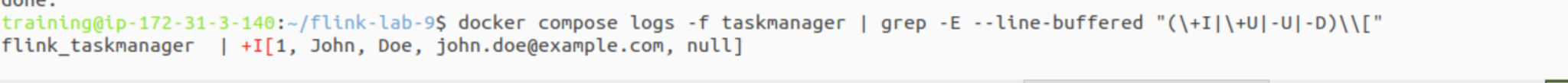
To see the output, view the logs of the taskmanager container. The +I prefix indicates an INSERT operation.

|  |
| --- |
| docker compose logs -f taskmanager | grep -E --line-buffered "(\+I|\+U|-U|-D)\\[" |

This updated command uses a more specific grep pattern to only show the data change lines (+I for insert, +U/-U for update, -D for delete), filtering out all other system logs for a much cleaner view.

You should see the initial record from our init.sql script:

|  |
| --- |
| ... +I[1, John, Doe, john.doe@example.com, <null>] ... |



(Press Ctrl+C to exit the logs when you are finished.)

**Step 2: Perform Database Operations**

Open a new terminal. We will get a shell inside the postgres container to run some SQL commands.

|  |
| --- |
| docker compose exec -it postgres psql -U flinkuser -d flinkdb |

Now, run some INSERT, UPDATE, and DELETE commands. After each one, observe the taskmanager logs in your other terminal.

|  |
| --- |
| -- Inside the psql shell INSERT INTO customers (first\_name, last\_name, email) VALUES ('Jane', 'Smith', 'jane.smith@example.com'); -- Log output will show: +I[2, Jane, Smith, jane.smith@example.com, <null>]  UPDATE customers SET email = 'john.d.doe@example.com' WHERE id = 1; -- Log output will show: -U[1, John, Doe, john.doe@example.com, <null>] -- +U[1, John, Doe, john.d.doe@example.com, <null>]  DELETE FROM customers WHERE id = 2; -- Log output will show: -D[2, Jane, Smith, jane.smith@example.com, <null>] |



**Step 3: Trigger Schema Evolution**

This is the key part of the lab. While the Flink job is still running, let's add a phone\_number column to the customers table.

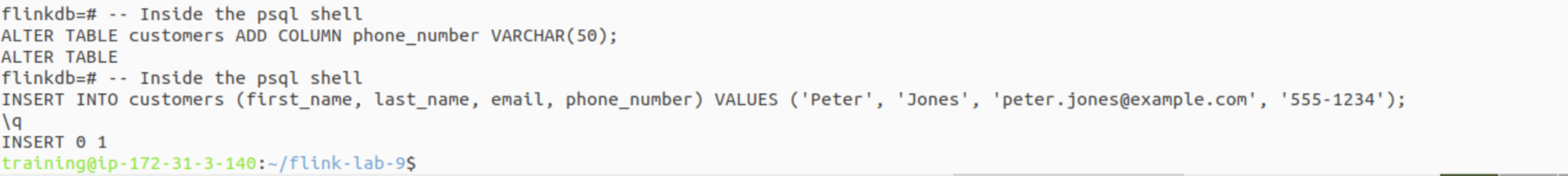
|  |
| --- |
| -- Inside the psql shell ALTER TABLE customers ADD COLUMN phone\_number VARCHAR(50); |

Flink's source connector will automatically detect this change and handle it without crashing the job.

**Step 4: Verify Schema Evolution**

Now, insert a new record that includes the new column.

|  |
| --- |
| -- Inside the psql shell INSERT INTO customers (first\_name, last\_name, email, phone\_number) VALUES ('Peter', 'Jones', 'peter.jones@example.com', '555-1234'); \q |



Check the taskmanager logs one last time.

|  |
| --- |
| docker compose logs -f taskmanager | grep -E --line-buffered "(\+I|\+U|-U|-D)\\[" |

(Press Ctrl+C to exit the logs when you are finished.)

Flink has adapted to the new schema by not crashing, which is the main goal. Our static SQL query, however, continues to project a NULL value for the phone\_number field because it was not written to dynamically select new columns.

|  |
| --- |
| ... -- The job continues to run, proving it handled the DDL change. -- However, our specific SELECT statement does not read the new column's data. -- It continues to insert the NULL value we defined in the script. +I[3, Peter, Jones, peter.jones@example.com, <null>] ... |

# 

# Part 7: Cleanup

Once you have finished, stop and remove all the containers and associated volumes to free up system resources.

From the ~/flink-lab-9 directory

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



If you want to cancel the running Flink job before shutting down, you can do so from the Flink UI.

# Part 8: Next Steps

* **Sinking to a Different System:** Modify the SQL script to sink the CDC stream into another system, like Kafka or a data warehouse table (e.g., Snowflake, as in Lab 11).
* **Transformations:** Add a transformation step in your INSERT ... SELECT query to mask sensitive data (like the email) before it lands in the sink.
* **Error Handling:** Investigate how Flink handles poison pills or malformed data within a CDC stream.
* **MongoDB CDC:** Explore Lab 10, which applies the same CDC principles to a NoSQL database, MongoDB, to see how Flink handles nested JSON documents.