Lab 8: Capturing and Flattening MongoDB CDC Streams

**Goal:** Extend Change Data Capture (CDC) concepts to a NoSQL database. This lab will guide you through capturing a real-time change stream from a MongoDB collection containing nested documents and using Flink SQL to flatten this complex data into a flat, relational structure.

### 

# INDEX

[**Purpose of this Lab**](#_9lz2xlftomns) **3**

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

[**Project Structure**](#_ctx0139bbitj) **3**

[**Part 1: Understanding NoSQL CDC**](#_pgv8kyuhc17q) **4**

[**Part 2: Project and Connector Setup**](#_iudai9yq3ycv) **4**

[Step 1: Create the Project Directory](#_jhjs1yw1dfpl) 4

[Step 2: Download the MongoDB CDC Connector](#_qb41g6jzyz3w) 4

[**Part 3: Configuring the Docker Environment**](#_b1ei7ctm8ei2) **5**

[Step 1: Create the MongoDB Replica Set Initialization Script](#_7zrovwpydf1e) 5

[Step 2: Define the Docker Compose Services](#_6frgygp7mnc8) 6

[**Part 4: Developing the Flink SQL Flattening Script**](#_n0lqjp46qau9) **7**

[**Part 5: Executing the End-to-End Pipeline**](#_qogpnl24o8g3) **8**

[Step 1: Start the Docker Environment](#_49p3y3ajzsg7) 8

[Step 2: Submit the Flink SQL Job](#_rj1zndy1u8zz) 9

[**Part 6: Verification and Data Transformation**](#_fajqh5lo5u39) **11**

[Step 1: Check the Flink UI](#_8o8l9n8vz6lx) 11

[Step 2: Insert Nested Data into MongoDB](#_4t78m7iqmyjn) 11

[Step 3: Observe the Flattened Output](#_pj4mvss8k07r) 13

[Step 4: Update a Nested Field](#_ty2heew0otgl) 14

[**Part 7: Cleanup**](#_i2c4x6qks0qf) **15**

[**Part 8: Next Steps**](#_mply1mhjij00) **15**

# 

# Purpose of this Lab

While Lab 9 focused on the structured world of relational databases, this lab dives into the flexible, semi-structured nature of NoSQL. Real-world applications often use MongoDB to store complex, nested JSON-like documents. The challenge is to make this data available for analytics, which typically requires a flat, tabular format.

You will learn how to configure a MongoDB replica set (a prerequisite for CDC), use the Flink MongoDB CDC connector to stream changes from the database's oplog, and apply Flink SQL functions to parse and transform nested fields into a simple, flat structure ready for analysis.

By completing this lab, you will:

* **Implement NoSQL CDC:** Capture real-time change events from a MongoDB collection.
* **Configure a MongoDB Replica Set:** Understand and implement the necessary database configuration for CDC.
* **Handle Nested Data Structures:** Define Flink SQL tables with complex ROW types to match nested JSON.
* **Flatten Complex Data:** Write a Flink SQL query to transform a nested data stream into a flat table.
* **Apply Concepts in a Fully Containerized Environment:** Manage a multi-service application with Flink and a NoSQL database using Docker Compose.

# Prerequisites

* An Ubuntu-based environment with Docker and Docker Compose installed.
* Successful completion of Lab 9 is highly recommended, as this lab builds directly on those concepts.
* Basic familiarity with JSON data structures.

# Project Structure

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

|  |
| --- |
| ~/flink-lab-10/ ├── jars/ │ └── flink-sql-connector-mongodb-cdc-2.4.2.jar # The MongoDB CDC connector ├── mongo/ │ └── init-replica.sh # Script to initialize the replica set └── sql/  └── flatten\_stream.sql # The Flink SQL job script └── docker-compose.yaml # Defines all services |

# Part 1: Understanding NoSQL CDC

Unlike PostgreSQL which uses a Write-Ahead Log (WAL), MongoDB's CDC mechanism relies on its **oplog** (operations log). The oplog is a special collection that records all data-modifying operations. For Flink to reliably tail this log, MongoDB must be running in a **replica set** configuration (even a single-node replica set is sufficient). This ensures the oplog has the properties needed for external systems like Flink to stream changes consistently.

# Part 2: Project and Connector Setup

## Step 1: Create the Project Directory

|  |
| --- |
| mkdir -p ~/flink-lab-10/{jars,mongo,sql} cd ~/flink-lab-10 |



## Step 2: Download the MongoDB CDC Connector

Download the Flink CDC Connector for MongoDB

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



# 

# Part 3: Configuring the Docker Environment

## Step 1: Create the MongoDB Replica Set Initialization Script

This script will be run by a temporary container to configure our single MongoDB instance as a replica set, which is required for CDC.

Create the file mongo/init-replica.sh:

|  |
| --- |
| code mongo/init-replica.sh |

Add the following shell script content. This script waits for the mongodb service to be ready and then executes the command to initialize the replica set.

|  |
| --- |
| #!/bin/bash echo "Starting replica set initialization" until mongosh --host mongodb --eval "print(\"waited for connection\")" do  sleep 2 done echo "Connection finished" echo "Creating replica set" mongosh --host mongodb <<EOF rs.initiate(  {  \_id : 'rs0',  members: [  { \_id: 0, host: "mongodb:27017" }  ]  } ) EOF echo "Replica set created" |

## Step 2: Define the Docker Compose Services

Create the docker-compose.yaml file. This is more complex than Lab 9, as it includes the MongoDB server and a temporary mongo-init container to run the script above.

code docker-compose.yaml

Add the following content:

|  |
| --- |
| # docker-compose.yaml services:  mongodb:  image: mongo:6.0  container\_name: mongodb  ports:  - "27017:27017"  command: ["--replSet", "rs0", "--bind\_ip\_all"]   mongo-init:  image: mongo:6.0  depends\_on:  - mongodb  volumes:  - ./mongo/init-replica.sh:/scripts/init-replica.sh  entrypoint: [ "bash", "/scripts/init-replica.sh" ]   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: 1  volumes:  - ./jars:/opt/flink/usrlib   taskmanager:  image: flink:1.18.0-scala\_2.12-java11  container\_name: flink\_taskmanager  depends\_on:  - jobmanager  - mongo-init  command: taskmanager  environment:  - |  FLINK\_PROPERTIES=  jobmanager.rpc.address: jobmanager  taskmanager.numberOfTaskSlots: 2  volumes:  - ./jars:/opt/flink/usrlib |

# 

# Part 4: Developing the Flink SQL Flattening Script

This SQL script will define a source table with a nested structure mirroring a MongoDB document and a flat sink table. The INSERT...SELECT statement will perform the transformation.

Create the file sql/flatten\_stream.sql:

|  |
| --- |
| code sql/flatten\_stream.sql |

Add the following content:

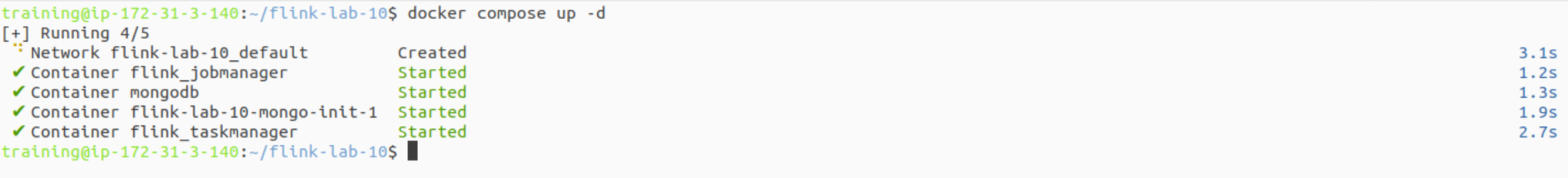
|  |
| --- |
| -- sql/flatten\_stream.sql SET 'sql-client.execution.result-mode' = 'tableau';  -- Define the source table with a nested structure CREATE TABLE users\_cdc (  \_id STRING,  full\_name STRING,  user\_details ROW<  address ROW<street STRING, city STRING>,  contact\_email STRING  >,  PRIMARY KEY (\_id) NOT ENFORCED ) WITH (  'connector' = 'mongodb-cdc',  'hosts' = 'mongodb:27017',  'database' = 'userdb',  'collection' = 'users' );  -- Define a flat sink table for the transformed data CREATE TABLE print\_sink\_flat (  user\_id STRING,  full\_name STRING,  street STRING,  city STRING,  email STRING ) WITH (  'connector' = 'print' );  -- The main pipeline query to flatten the nested source and insert into the sink INSERT INTO print\_sink\_flat SELECT  \_id AS user\_id,  full\_name,  user\_details.address.street,  user\_details.address.city,  user\_details.contact\_email AS email FROM users\_cdc; |

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

## Step 1: Start the Docker Environment

From the ~/flink-lab-10 directory, launch all services.

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

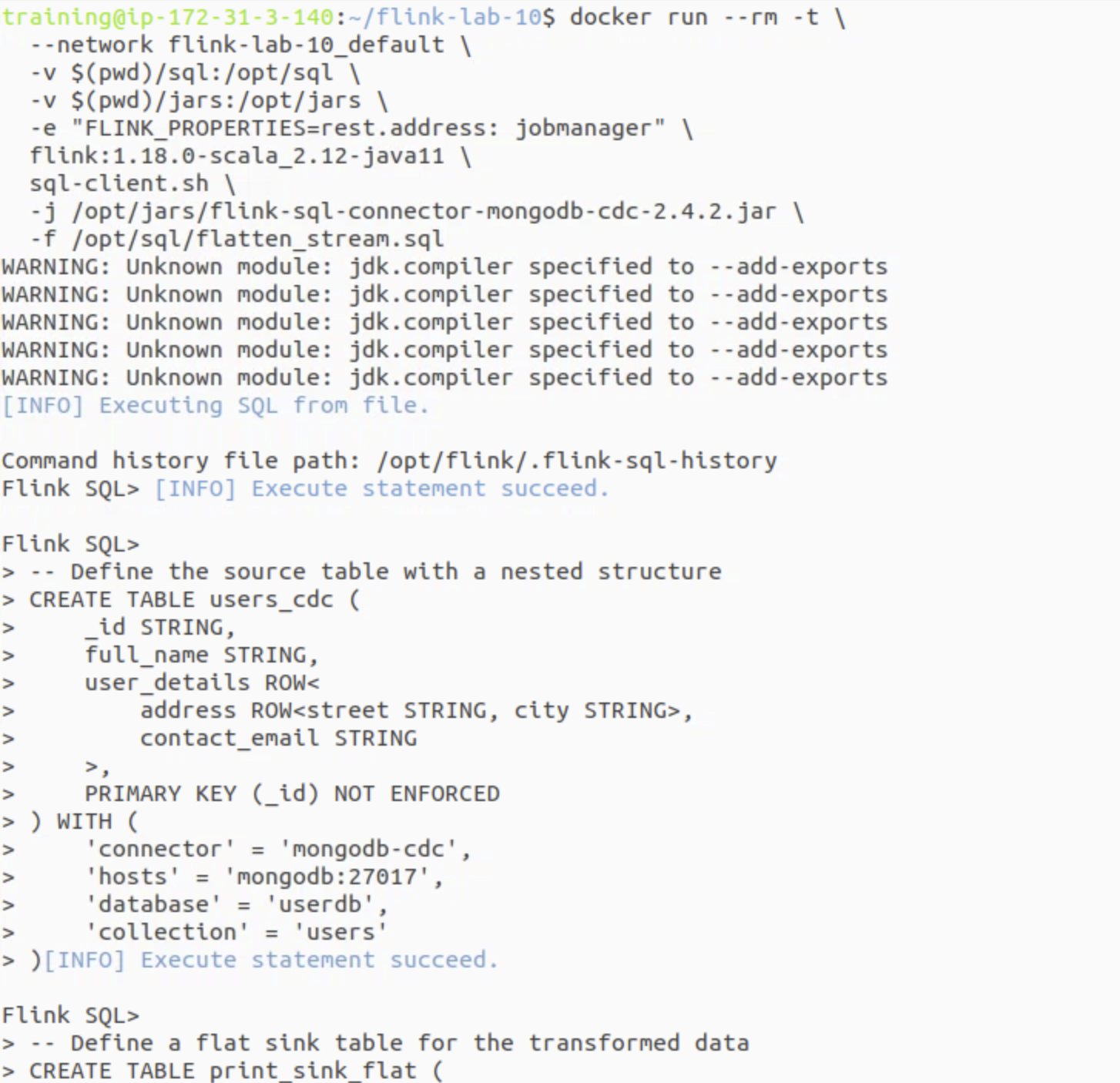
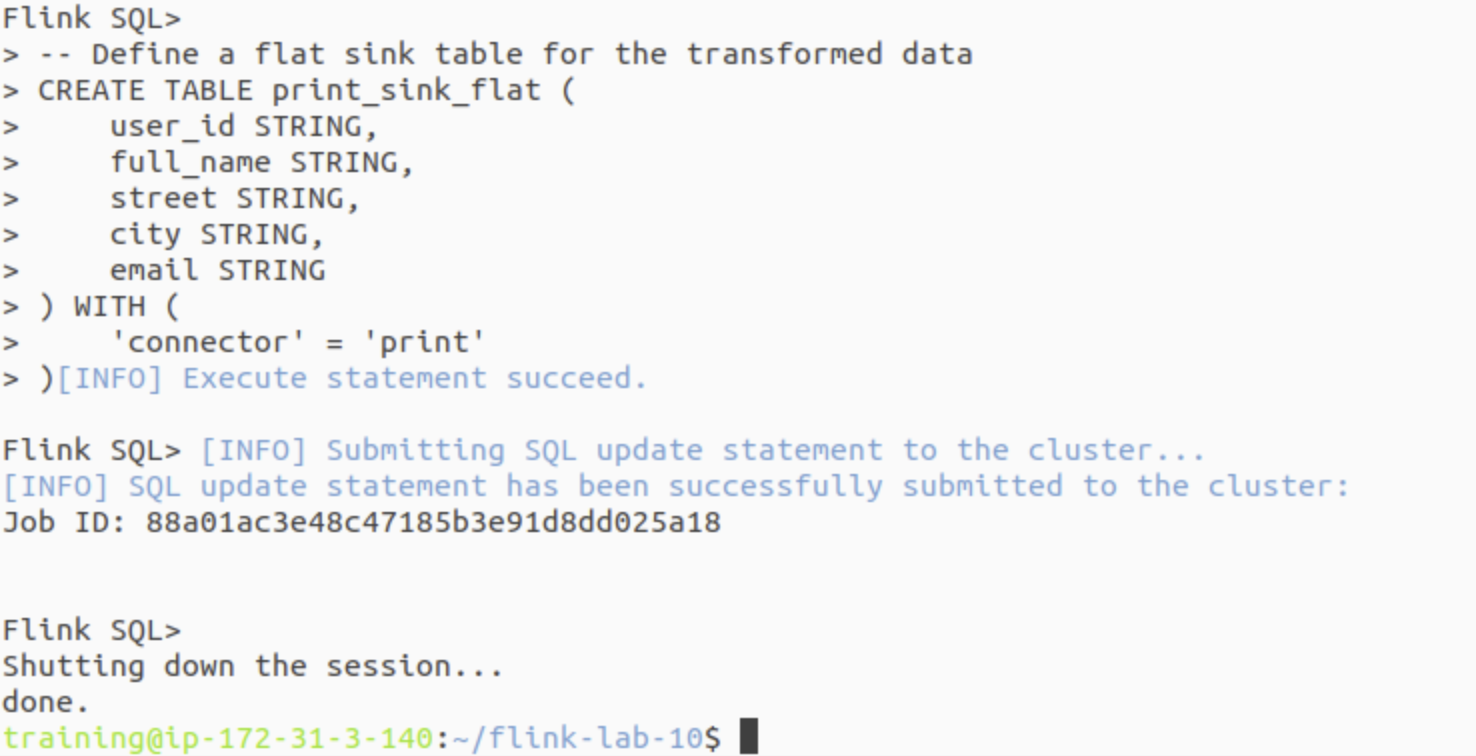


It may take a minute for the replica set to initialize. You can check the logs of the mongo-init container (docker compose logs mongo-init) which should end with "Replica set created".

## Step 2: Submit the Flink SQL Job

Use the sql-client in a temporary container to submit the job.

|  |
| --- |
| docker run --rm -t \  --network flink-lab-10\_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-mongodb-cdc-2.4.2.jar \  -f /opt/sql/flatten\_stream.sql |

# Part 6: Verification and Data Transformation

## Step 1: Check the Flink UI

Open the Flink UI at <http://localhost:8081> to confirm the job is running.

## Step 2: Insert Nested Data into MongoDB

Open a new terminal and connect to the MongoDB container.

|  |
| --- |
| docker compose exec -it mongodb mongosh |

Inside the mongosh shell, switch to the userdb database and insert a nested document into the users collection.

|  |
| --- |
| // Inside the mongosh shell use userdb; |



Next, insert a nested document into the users collection. Make sure you press Enter after the final }.

|  |
| --- |
| db.users.insertOne({  full\_name: "Alice Williams",  user\_details: {  address: {  street: "123 Maple St",  city: "Anytown"  },  contact\_email: "alice.w@example.com"  } }); |

Verification: A successful command will output a confirmation message like this. If you do not see this, the document was not inserted.

|  |
| --- |
| {  "acknowledged": true,  "insertedId": ObjectId("...")  } |

Note: Keep this mongosh shell open. You will use it again in Step 4.



## Step 3: Observe the Flattened Output

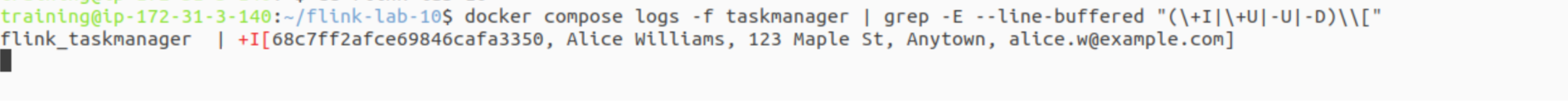
In your first terminal, check the taskmanager logs. You will see the nested MongoDB document has been transformed into a single, flat row.

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

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

The output should look like this (the ID will vary):

|  |
| --- |
| +I[66e5a6b..., Alice Williams, 123 Maple St, Anytown, alice.w@example.com] |



## Step 4: Update a Nested Field

Now, update a deeply nested field in MongoDB and see how Flink captures it as a change event.

|  |
| --- |
| // Inside the mongosh shell db.users.updateOne(  { full\_name: "Alice Williams" },  { $set: { "user\_details.address.city": "Newville" } } ); |

**Verification:** A successful update will output a confirmation.

|  |
| --- |
| {  "acknowledged": true,  "insertedId": null,  "matchedCount": 1,  "modifiedCount": 1,  "upsertedCount": 0 } |



Finally, exit the mongo shell.

|  |
| --- |
| exit |

Checking the taskmanager logs again will show the UPDATE event, represented by a deletion of the old record (-U) and an insertion of the new, updated record (+U), correctly flattened.

|  |
| --- |
| -U[66e5a6b..., Alice Williams, 123 Maple St, Anytown, alice.w@example.com] +U[66e5a6b..., Alice Williams, 123 Maple St, Newville, alice.w@example.com] |

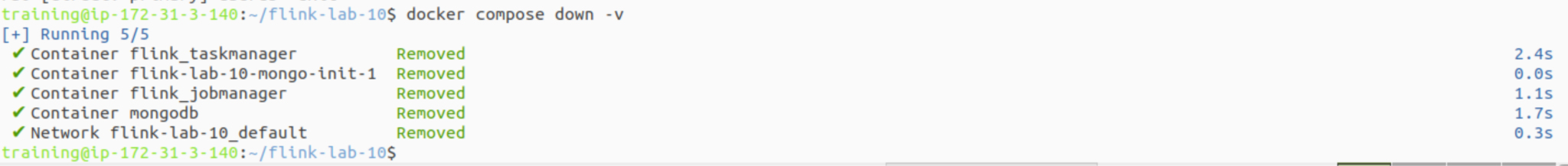
### 

# Part 7: Cleanup

When you are finished, stop and remove all containers and volumes.

From the ~/flink-lab-10 directory

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



# Part 8: Next Steps

* **Handling Arrays:** Modify the MongoDB documents and the Flink job to handle arrays (e.g., a list of phone numbers) using Flink SQL's UNNEST function to create multiple rows from a single document.
* **Sinking to a Relational Table:** Instead of the print sink, create a second database container (like PostgreSQL from Lab 9) and write a Flink SQL job that sinks the flattened MongoDB stream into a structured SQL table.
* **Data Type Conversion:** Experiment with different data types in MongoDB (like dates or numbers) and use Flink's CAST functions to ensure they are correctly typed in the flattened output.