Lab 10: Analyzing User Behavior with Tunable Session Windows

**Goal:** Implement a data-driven session window using Flink SQL to group user activity into sessions and analyze the impact of tuning the session inactivity gap.

# 

# INDEX

[**Purpose of this Lab**](#_onotgk2ia9f5) **3**

[**Prerequisites**](#_cpvxetosxhn1) **4**

[**Project Structure**](#_vwgefpyffal1) **4**

[**Part 1: Project and Environment Setup**](#_22v7yrocu1un) **4**

[**Part 2: Setting Up the Kafka Cluster**](#_rygvqc30pn9b) **6**

[**Part 3: Developing the Flink Session Window Application**](#_gqsanyjqpc79) **7**

[**Part 4: Executing and Analyzing the Impact of the Gap**](#_b1s3rz9lu4wj) **11**

[**Part 5: Verification**](#_y3pxk52412ca) **14**

[**Part 6: Cleanup**](#_loumk5bge98q) **15**

[**Part 7: Next Steps**](#_1pa6mrfnjer3) **15**

# 

# Purpose of this Lab

This lab moves beyond fixed-time windows (like Tumbling Windows) to **Session Windows**, which are defined by data itself. A session window groups a sequence of events for a specific key (e.g., a user\_id) and is closed only after a defined period of inactivity occurs—a session gap. This is incredibly useful for analyzing user behavior, tracking website visits, or monitoring device activity.

You will write a Flink SQL query that groups user clicks into sessions. The critical part of this lab is to run the analysis twice with two different session gap parameters (5 minutes and 1 minute) and compare the results. This will give you direct insight into how a single parameter can change your understanding of user engagement.

By completing this lab, you will:

**Implement Session Windows:** Use the SESSION function in a streaming SQL query to group data based on periods of inactivity.

**Understand Data-Driven Windowing:** See how windows are not fixed in time but are instead determined by the arrival patterns of the data itself.

**Perform Parameter Tuning:** Directly observe how changing the session gap parameter affects the number and duration of the resulting sessions.

**Analyze User Behavior:** Learn a fundamental technique for sessionization, a common requirement in user behavior analytics.

**Reinforce SQL Pipeline Skills:** Continue to build end-to-end streaming pipelines using only SQL DDL and DML.

# Prerequisites

This lab assumes you have successfully completed Labs 1 through 6 and are using an Ubuntu environment. Your Flink cluster should already have the Kafka connector JAR in its lib directory.

# Project Structure

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

|  |
| --- |
| ~/flink-lab-7/  ├── venv/ # The isolated Python virtual environment ├── docker-compose.yaml # Defines our Kafka service (reused) ├── producer.py # Script to generate mock user click data with variable delays └── session\_window\_analysis.py # The Flink SQL job script for this lab |

# Part 1: Project and Environment Setup

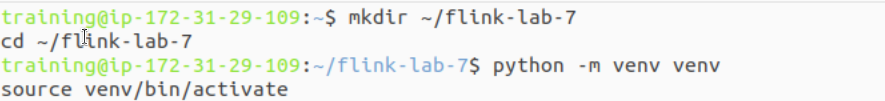
**Step 1: Create Project Directory and Virtual Environment**

Create and navigate to the new lab directory:

|  |
| --- |
| mkdir ~/flink-lab-7 cd ~/flink-lab-7 |

Initialize and activate a Python virtual environment:

|  |
| --- |
| python -m venv venv source venv/bin/activate |



**Step 2: Install Python Dependencies**

With the venv active, install apache-flink and the Python client for Kafka.

|  |
| --- |
| pip install "apache-flink==2.0.0" kafka-python |



**Step 3: Configure Flink for the New Project**

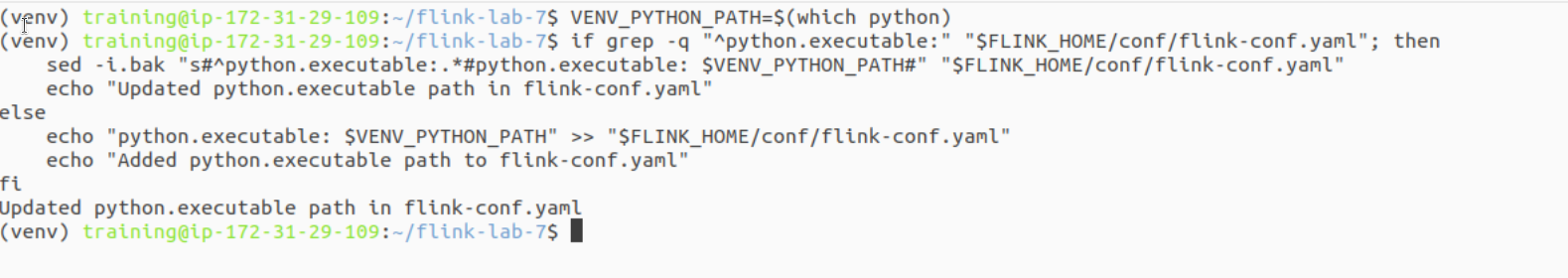
Update Flink's configuration to point to the correct Python executable for this lab.

Get the absolute path to the Python executable in the new venv:

|  |
| --- |
| VENV\_PYTHON\_PATH=$(which python) |

Update the flink-conf.yaml file:

|  |
| --- |
| if grep -q "^python.executable:" "$FLINK\_HOME/conf/flink-conf.yaml"; then  sed -i.bak "s#^python.executable:.\*#python.executable: $VENV\_PYTHON\_PATH#" "$FLINK\_HOME/conf/flink-conf.yaml"  echo "Updated python.executable path in flink-conf.yaml" else  echo "python.executable: $VENV\_PYTHON\_PATH" >> "$FLINK\_HOME/conf/flink-conf.yaml"  echo "Added python.executable path to flink-conf.yaml" fi |



# Part 2: Setting Up the Kafka Cluster

**Step 1: Define and Launch the Kafka Service**

We will reuse the same docker-compose.yaml configuration from the previous labs.

Create the file in ~/flink-lab-7:

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

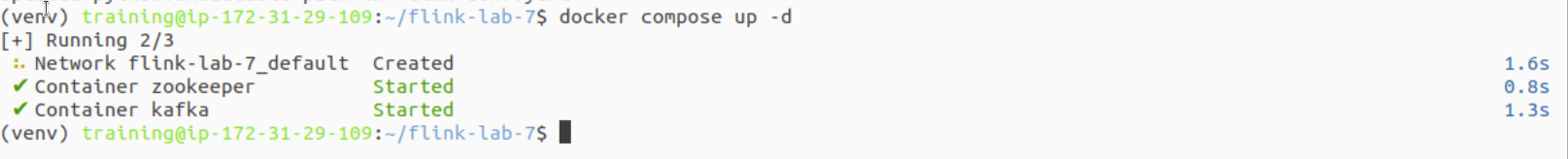
Add the following content:

|  |
| --- |
| # docker-compose.yaml services:  zookeeper:  image: confluentinc/cp-zookeeper:7.3.2  container\_name: zookeeper  ports: ["2181:2181"]  environment:  ZOOKEEPER\_CLIENT\_PORT: 2181  ZOOKEEPER\_TICK\_TIME: 2000  kafka:  image: confluentinc/cp-kafka:7.3.2  container\_name: kafka  ports: ["9092:9092"]  depends\_on: [zookeeper]  environment:  KAFKA\_BROKER\_ID: 1  KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181  KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT\_HOST://localhost:9092  KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT  KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT  KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1 |

From the ~/flink-lab-7 directory, start the services:

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

### 



# Part 3: Developing the Flink Session Window Application

**Step 1: Implement the Kafka Producer**

Create producer.py. This script is designed to simulate realistic user behavior by generating bursts of clicks followed by longer periods of inactivity, which will test our session window logic.

Create the file:

|  |
| --- |
| code producer.py |

Add the following code:

|  |
| --- |
| # producer.py import json import time import random from datetime import datetime from kafka import KafkaProducer from kafka.admin import KafkaAdminClient, NewTopic from kafka.errors import TopicAlreadyExistsError, NoBrokersAvailable  KAFKA\_TOPIC = 'user\_clicks' KAFKA\_BROKERS = 'localhost:9092'  def create\_producer\_and\_topic():  # ... (Same robust connection logic as Lab 6)  retries = 10  while retries > 0:  try:  admin\_client = KafkaAdminClient(bootstrap\_servers=KAFKA\_BROKERS)  try:  topic = NewTopic(name=KAFKA\_TOPIC, num\_partitions=1, replication\_factor=1)  admin\_client.create\_topics(new\_topics=[topic], validate\_only=False)  print(f"Topic '{KAFKA\_TOPIC}' created.")  except TopicAlreadyExistsError:  print(f"Topic '{KAFKA\_TOPIC}' already exists.")  finally:  admin\_client.close()   producer = KafkaProducer(  bootstrap\_servers=KAFKA\_BROKERS,  value\_serializer=lambda v: json.dumps(v).encode('utf-8')  )  print("Successfully connected to Kafka.")  return producer  except NoBrokersAvailable:  retries -= 1  print(f"Kafka not available, retrying in 5 seconds... ({retries} retries left)")  time.sleep(5)  raise RuntimeError("Failed to connect to Kafka.")  if \_\_name\_\_ == '\_\_main\_\_':  producer = create\_producer\_and\_topic()  user\_ids = [f'user\_{i}' for i in range(1, 5)]   print("Producing mock user click events... Press Ctrl+C to terminate.")  try:  while True:  # Select a user to generate a session for  current\_user = random.choice(user\_ids)  print(f"\n--- Starting a click session for {current\_user} ---")   # Generate a burst of 3 to 6 clicks  for \_ in range(random.randint(3, 6)):  event\_time = int(time.time() \* 1000)  event = {'user\_id': current\_user, 'click\_time': event\_time}  producer.send(KAFKA\_TOPIC, value=event)  print(f"Sent click for {current\_user} at {datetime.fromtimestamp(event\_time/1000)}")  # Short delay between clicks within a session  time.sleep(random.uniform(5, 20))   # Simulate user inactivity to cause a session break  # This long pause is key for testing the session gap  inactivity\_period = random.uniform(70, 150) # Between 1.1 and 2.5 minutes  print(f"--- {current\_user} is now inactive for {inactivity\_period:.2f} seconds ---")  time.sleep(inactivity\_period)   except KeyboardInterrupt:  print("\nStopping producer.")  finally:  producer.flush()  producer.close() |

**Step 2: Implement the Flink SQL Script**

Create the main application file, session\_window\_analysis.py. Note the SESSION(ts, INTERVAL '5' MINUTE) clause, which we will modify later.

Create the file:

|  |
| --- |
| code session\_window\_analysis.py |

Add the following code:

|  |
| --- |
| # session\_window\_analysis.py from pyflink.datastream import StreamExecutionEnvironment from pyflink.table import StreamTableEnvironment  def main():  env = StreamExecutionEnvironment.get\_execution\_environment()  table\_env = StreamTableEnvironment.create(stream\_execution\_environment=env)  env.enable\_checkpointing(5000)   # 2. Create a source table from the 'user\_clicks' Kafka topic  table\_env.execute\_sql("""  CREATE TABLE user\_clicks (  user\_id STRING,  click\_time BIGINT,  -- Define the event-time attribute  ts AS TO\_TIMESTAMP\_LTZ(click\_time, 3),  -- Define a watermark strategy  WATERMARK FOR ts AS ts - INTERVAL '5' SECOND  ) WITH (  'connector' = 'kafka',  'topic' = 'user\_clicks',  'properties.bootstrap.servers' = 'localhost:9092',  'properties.group.id' = 'flink-sql-session-group',  'scan.startup.mode' = 'latest-offset',  'format' = 'json'  )  """)   # 3. Create a sink table to print results to the console  table\_env.execute\_sql("""  CREATE TABLE print\_sink (  user\_id STRING,  session\_start TIMESTAMP(3),  session\_end TIMESTAMP(3),  click\_count BIGINT  ) WITH (  'connector' = 'print'  )  """)   # 4. Define and execute the session window query.  # A session window closes when an inactivity gap of 5 minutes is detected.  table\_env.execute\_sql("""  INSERT INTO print\_sink  SELECT  user\_id,  SESSION\_START(ts, INTERVAL '5' MINUTE) as session\_start,  SESSION\_END(ts, INTERVAL '5' MINUTE) as session\_end,  COUNT(\*) as click\_count  FROM user\_clicks  GROUP BY  user\_id,  SESSION(ts, INTERVAL '5' MINUTE)  """)  if \_\_name\_\_ == '\_\_main\_\_':  main() |

# Part 4: Executing and Analyzing the Impact of the Gap

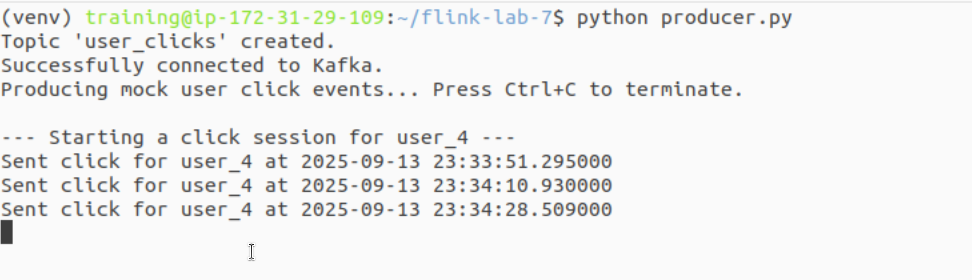
This part is interactive and has two phases.

**Phase 1: Run with a 5-Minute Session Gap**

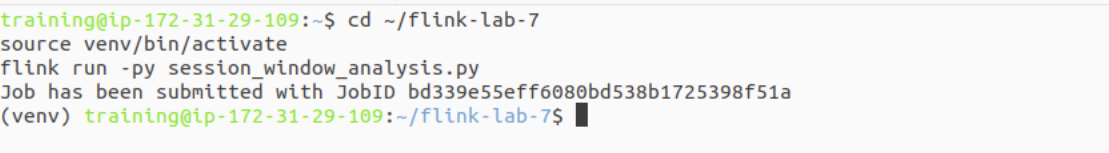
**Start Flink Cluster (Terminal 1):** stop-cluster.sh && start-cluster.sh

**Launch Data Producer (Terminal 2):**

|  |
| --- |
| cd ~/flink-lab-7 source venv/bin/activate python producer.py |

**Submit Flink Job (Terminal 3):**

|  |
| --- |
| cd ~/flink-lab-7 source venv/bin/activate flink run -py session\_window\_analysis.py |



**Observe:** Watch the producer log. It will generate a burst of clicks for a user, then pause for 1-2 minutes. Now, check the TaskManager log.

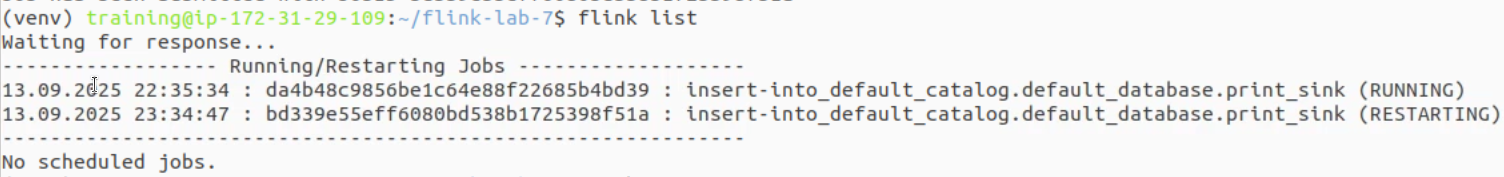
|  |
| --- |
| tail -f $(ls -t $FLINK\_HOME/log/flink-\*-taskexecutor-\*.out | head -n 1) |

  
You will likely **not** see any output for a long time. This is because the producer's inactivity period (max 2.5 minutes) is shorter than the session gap (5 minutes). Flink keeps the session open, waiting for more events.

**Phase 2: Run with a 1-Minute Session Gap**

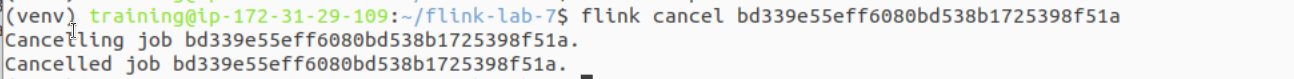
**Stop the Flink job:** In **Terminal 3**, find your job's ID by listing the running jobs, and then cancel it. Keep the producer in Terminal 2 running.  
First, list the running jobs to get the ID

|  |
| --- |
| flink list |



Then, use the ID to cancel the job (replace <jobId> with the actual ID from the list)

|  |
| --- |
| flink cancel <jobId> |



**Modify the SQL Script:** Open session\_window\_analysis.py and change the two occurrences of INTERVAL '5' MINUTE to INTERVAL '1' MINUTE.

|  |
| --- |
| # ... inside session\_window\_analysis.py ...  # ...  SELECT  user\_id,  SESSION\_START(ts, INTERVAL '1' MINUTE) as session\_start,  SESSION\_END(ts, INTERVAL '1' MINUTE) as session\_end,  COUNT(\*) as click\_count  FROM user\_clicks  GROUP BY  user\_id,  SESSION(ts, INTERVAL '1' MINUTE) # ... |

**Save the file** and resubmit the job in Terminal 3:

|  |
| --- |
| flink run -py session\_window\_analysis.py |

# Part 5: Verification

Now, monitor the TaskManager logs again in Terminal 3.

After the producer completes an inactivity period for a user (which is always > 1 minute), you will see output in the Flink logs almost immediately. It will look similar to this:



You have now successfully verified the core concept:

With a **5-minute gap**, the sessions were rarely closed because the user inactivity was not long enough.

With a **1-minute gap**, the sessions were closed correctly after each burst of activity because the inactivity period exceeded the gap.

This demonstrates how critical the gap parameter is for defining what constitutes a "session" in your business logic.

# Part 6: Cleanup

**Stop the Flink job:** If it's still running from Phase 2, cancel it from **Terminal 3** using its Job ID.

|  |
| --- |
| flink list flink cancel <jobId> |

S**top the producer:** Press Ctrl+C in Terminal 2.

**Stop the Flink cluster:** stop-cluster.sh

# Part 7: Next Steps

**Session Start/End Gap:** Explore the more advanced SESSION\_WITH\_START function, which allows you to define a starting boundary in addition to the inactivity gap.

**Dynamic Gaps:** Research how the DataStream API's ProcessWindowFunction can be used with session windows to implement more complex logic, such as having the gap size change dynamically based on the data.

**Combine with Other Aggregations:** Add more aggregations to your query. For example, calculate the duration of each session by subtracting SESSION\_START from SESSION\_END.