**Lab Guide: Advanced Structured Streaming with Apache Spark**

**Lab Overview**

This lab provides hands-on practice with Apache Spark Structured Streaming concepts including micro-batch vs continuous modes, watermarks, output modes, checkpointing, Auto Loader, streaming to Delta Lake, triggers, graceful restart, and Kafka exactly-once processing.

**Tools Required:**

* Apache Spark 3.5+
* Delta Lake 2.0+
* Kafka (local or cloud)
* Cloud Storage or DBFS (for Auto Loader)
* Databricks (recommended) or any Spark-compatible IDE

**Lab 1: Micro-Batch vs Continuous**

**Objective:** Compare latency and processing behavior of micro-batch and continuous modes.

**Steps:**

1. Start Spark session.
2. Create a dummy streaming source (rate source):

val df = spark.readStream.format("rate").option("rowsPerSecond", 10).load()

1. Write using micro-batch:

df.writeStream.format("console").option("truncate", false).start()

1. Repeat using continuous mode:

df.writeStream.format("console").trigger(Trigger.Continuous("1 second")).start()

1. Observe differences in output timing.

**Lab 2: Watermarks & Event Time Aggregation**

**Objective:** Handle late data with watermarking.

**Steps:**

1. Prepare JSON data with event timestamps (some intentionally delayed).
2. Use Auto Loader to read files with:

val df = spark.readStream.format("cloudFiles")

.option("cloudFiles.format", "json")

.load("/mnt/data/events")

1. Apply watermark:

df.withWatermark("event\_time", "10 minutes")

.groupBy(window(col("event\_time"), "5 minutes"))

.count()

1. Write to console or Delta sink. Observe dropped late records.

**Lab 3: Output Modes**

**Objective:** Explore different streaming output modes.

**Steps:**

1. Use streaming count aggregation:

val df = spark.readStream.format("rate").load()

val agg = df.groupBy(window(col("timestamp"), "10 seconds")).count()

1. Try each mode:

agg.writeStream.outputMode("complete").format("console").start()

agg.writeStream.outputMode("append").format("console").start()

agg.writeStream.outputMode("update").format("console").start()

1. Observe result differences.

**Lab 4: Checkpointing and Recovery**

**Objective:** Enable fault tolerance using checkpoints.

**Steps:**

1. Write to Delta with checkpointing:

df.writeStream

.format("delta")

.option("checkpointLocation", "/mnt/checkpoints/lab4")

.start("/mnt/delta/lab4")

1. Simulate failure (stop Spark job).
2. Restart job with same code and checkpoint path.
3. Observe recovery and no data duplication.

**Lab 5: Auto Loader with Schema Inference**

**Objective:** Ingest cloud files incrementally with Auto Loader.

**Steps:**

1. Upload sample CSV files to DBFS or cloud.
2. Read using Auto Loader:

val df = spark.readStream.format("cloudFiles")

.option("cloudFiles.format", "csv")

.option("cloudFiles.inferColumnTypes", "true")

.load("/mnt/data/csv")

1. Observe inferred schema and streaming behavior.
2. Add new file and confirm ingestion.

**Lab 6: Stream to Delta Lake**

**Objective:** Store real-time data in Delta Lake.

**Steps:**

1. Use any streaming source.
2. Write to Delta:

.writeStream

.format("delta")

.option("checkpointLocation", "/mnt/chkpt/lab6")

.start("/mnt/delta/lab6")

1. Query Delta table:

spark.read.format("delta").load("/mnt/delta/lab6").show()

1. Observe append and ACID behavior.

**Lab 7: Trigger Variations**

**Objective:** Understand trigger types and their behaviors.

**Steps:**

1. Try with ProcessingTime:

.writeStream.trigger(Trigger.ProcessingTime("10 seconds")).start()

1. Try Once trigger:

.writeStream.trigger(Trigger.Once()).start()

1. Try AvailableNow:

.writeStream.trigger(Trigger.AvailableNow()).start()

1. Observe execution patterns.

**Lab 8: Graceful Restart of Streaming Job**

**Objective:** Simulate and recover from job failure.

**Steps:**

1. Start long-running streaming job with checkpoint.
2. Terminate Spark session.
3. Restart cluster/notebook.
4. Re-run same code with checkpoint location.
5. Observe continued processing from last checkpoint.

**Lab 9: Kafka Streaming with Exactly-Once Guarantees**

**Objective:** Use Kafka + Delta to implement exactly-once semantics.

**Steps:**

1. Kafka Source:

val kafkaDF = spark.readStream.format("kafka")

.option("kafka.bootstrap.servers", "localhost:9092")

.option("subscribe", "iot\_events")

.option("startingOffsets", "earliest")

.load()

1. Transform and write to Delta:

kafkaDF.selectExpr("CAST(key AS STRING)", "CAST(value AS STRING)")

.writeStream.format("delta")

.option("checkpointLocation", "/mnt/chkpt/kafka")

.start("/mnt/delta/kafka")

1. Validate output:

spark.read.format("delta").load("/mnt/delta/kafka").count()

1. Restart and confirm no duplication.

**Lab Summary**

These labs offer practical exposure to core streaming concepts. Each experiment builds your understanding of reliable, scalable, and fault-tolerant streaming pipelines using Spark + Delta + Kafka.

**Further Reading & References:**

* <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>
* <https://docs.delta.io/latest/index.html>
* <https://spark.apache.org/docs/latest/structured-streaming-kafka-integration.html>