Structured Streaming in Databricks

1. Streaming Data Fundamentals

Core Concepts

- **Micro-batch Processing**: Processes data in small batches (as fast as 100ms intervals)
- **Exactly-once Processing**: Guarantees no duplicates or missing data
- **Stateful Operations**: Maintains context across batches (e.g., aggregations)
- **Event-time Processing**: Handles late-arriving data with watermarks

Streaming vs Batch Comparison

Feature	Batch Processing	Streaming Processing
Latency	High (minutes-hours)	Low (seconds-minutes)
Data Scope	Bounded (finite)	Unbounded (infinite)
Execution	Single run	Continuous
State	Stateless	Stateful operations possible

📌 **Diagram Idea:** Batch vs Streaming Data Flow

2. Structured Streaming Architecture

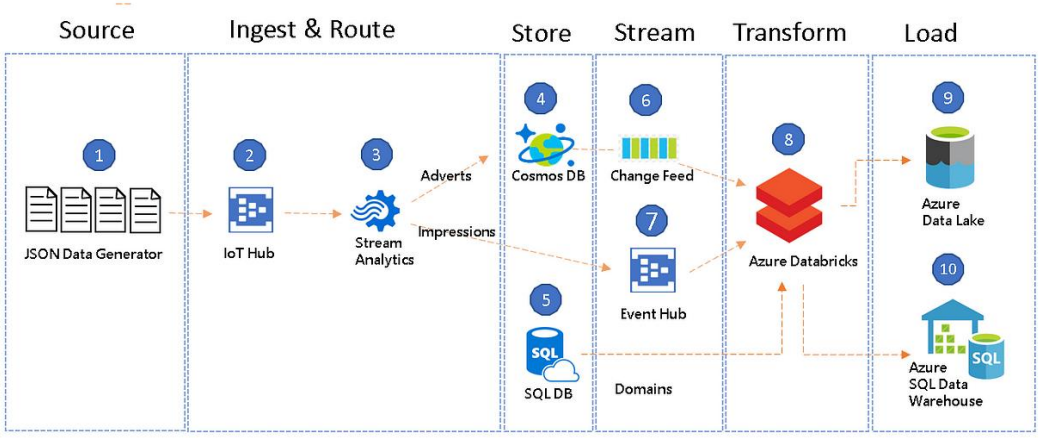
Core Components

1. **Source**: Kafka, Files, Sockets, etc.
2. **Streaming DataFrame**: Logical plan of transformations
3. **Sink**: Delta Lake, Kafka, Console, etc.
4. **Checkpoint**: Tracks progress for fault tolerance

Execution Model

```
(spark.readStream
  .format("source")
  .load()
  .transformations()
  .writeStream
  .format("sink")
  .start())
```

Structured Streaming Pipeline Architecture



3. Streaming Sources

File Stream

```
stream = (spark.readStream
  .format("cloudFiles")  # Auto-detects new files
  .option("cloudFiles.format", "json")
  .option("cloudFiles.schemaLocation", "/checkpoints/schema")
  .load("/raw/events/"))
```

Kafka Stream

```
stream = (spark.readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host1:port1,host2:port2")
  .option("subscribe", "topic1")
  .option("startingOffsets", "earliest")
  .load())
```

Socket Stream (Testing)

```
stream = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()
```

4. Transformations on Streams

Stateless Transformations

```
# Projection
stream.select("user_id", "event_time")

# Filter
stream.filter("amount > 100")

# Simple aggregations
stream.groupBy("user_id").count()
```

Stateful Transformations

```
# Windowed aggregations
from pyspark.sql.functions import window

(stream
  .withWatermark("event_time", "10 minutes")
  .groupBy(
    window("event_time", "5 minutes"),
    "user_id"
  )
  .agg(sum("amount").alias("total_amount")))
```

Joins

```
# Stream-static join
stream.join(static_df, "user_id", "left")

# Stream-stream join
stream1.join(
  stream2,
  expr("""
    stream1.user_id = stream2.customer_id AND
    stream1.event_time BETWEEN stream2.time AND stream2.time + INTERVAL 1
HOUR
  """),
  "inner"
)
```

5. Streaming Sinks

Delta Lake Sink

```
(stream.writeStream
  .format("delta")
  .outputMode("append")  # or "complete", "update"
  .option("checkpointLocation", "/checkpoints/delta")
  .start("/delta/streaming_output"))
```

Kafka Sink

```
(stream.selectExpr("CAST(user_id AS STRING) AS key", "to_json(struct(*)) AS
value")
  .writeStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "host:port")
  .option("topic", "output_topic")
  .start())
```

Foreach Batch (Custom Logic)

```
def process_batch(batch_df, batch_id):
    batch_df.persist()
    # Custom transformations
    batch_df.write.mode("append").saveAsTable("processed_events")
    batch_df.unpersist()

(stream.writeStream
  .foreachBatch(process_batch)
  .start())
```

6. Output Modes Explained

Mode	Description	Supported Operations
Append	Only new rows added	Filters, projections, watermarked aggregations
Complete	Full output each trigger	All aggregations
Update	Only changed rows	All operations except aggregations without watermarks

7. Watermarking & Late Data

Event-time Handling

```
from pyspark.sql.functions import current_timestamp

(stream
  .withColumn("processing_time", current_timestamp())
  .withWatermark("event_time", "10 minutes")  # Late threshold
  .groupBy(window("event_time", "5 minutes"))
  .count())
```

Late Data Policies

```
# Drop late data (default)
spark.conf.set("spark.sql.streaming.dropLateEventsInWatermark", "true")

# Process with separate handler
late_data = stream.filter("event_time < watermark")
```

8. Fault Tolerance & Checkpoints

Checkpoint Structure

```
/checkpoints/stream1/
├── commits/       # Track processed batches
├── offsets/       # Source offsets
└── state/         # Operator states (aggregations)
```

Recovery Scenarios

```
# Restart from checkpoint
(spark.readStream
  .format("delta")
  .option("checkpointLocation", "/checkpoints/existing")
  .load()
  .writeStream
  .format("delta")
  .option("checkpointLocation", "/checkpoints/existing")  # Same path!
  .start())
```

9. Monitoring Streaming Queries

Programmatic Monitoring

```
query = stream.writeStream...start()

# Get status
print(query.status)       # Current progress
print(query.lastProgress) # Detailed metrics

# Active streams
for q in spark.streams.active:
    print(f"ID: {q.id}, Status: {q.status}")
```

Spark UI Metrics

- **Input Rate**: Records/second
- **Processing Rate**: Batches/second
- **Latency**: Trigger interval vs processing time
- **State**: Size of maintained state

📌 **Diagram Idea:** Streaming Query Metrics Dashboard

10. Advanced Patterns

Streaming Deduplication

```
(stream
  .withWatermark("event_time", "10 minutes")
  .dropDuplicates(["event_id", "event_time"]))
```

Sessionization

```
from pyspark.sql.functions import session_window

(stream
  .groupBy(
    session_window("event_time", "5 minutes"),
    "user_id"
  )
  .agg(count("*").alias("events")))
```

ML on Streams

```
# Load pre-trained model
model = mlflow.spark.load_model("models:/fraud_detection/Production")

# Apply to stream
predictions = stream.transform(lambda df: model.transform(df))
```

11. Performance Tuning

Optimization Techniques

```
# Cluster config
spark.conf.set("spark.sql.shuffle.partitions", "200")  # For stateful ops

# Source options
  .option("maxFilesPerTrigger", "100")  # Control batch size
```

```
  .option("maxOffsetsPerTrigger", "50000")  # Kafka rate limit

# Sink options
  .option("trigger", "30 seconds")  # Processing interval
```

Troubleshooting Slow Streams

1. Check **watermark delays**
2. Monitor **state store size**
3. Verify **source partitioning**
4. Check **sink bottlenecks**

12. Real-World Pipeline Example

End-to-End IoT Pipeline

```
# Read from Kafka
raw_stream = (spark.readStream
  .format("kafka")
  .option("kafka.bootstrap.servers", "kafka:9092")
  .option("subscribe", "iot_events")
  .load())

# Parse JSON
from pyspark.sql.functions import from_json, col
schema = "device_id STRING, temperature DOUBLE, humidity DOUBLE, timestamp
TIMESTAMP"

parsed_stream = raw_stream.select(
  from_json(col("value").cast("string"),
schema).alias("data")).select("data.*")

# Process with watermarks
processed_stream = (parsed_stream
  .withWatermark("timestamp", "10 minutes")
  .groupBy(
    window("timestamp", "5 minutes"),
    "device_id"
  )
  .agg(
    avg("temperature").alias("avg_temp"),
    max("humidity").alias("max_humidity")
  ))

# Write to Delta Lake + Alert on anomalies
(processed_stream.writeStream
  .format("delta")
  .outputMode("complete")
  .option("checkpointLocation", "/checkpoints/iot_agg")
  .start("/delta/iot_aggregations"))

(processed_stream.filter("avg_temp > 100")
  .writeStream
  .format("console")
  .start())
```

13. Testing Streaming Applications

Unit Testing with Memory Streams

```
test_data = [(1, "2023-01-01 00:00:00"), (2, "2023-01-01 00:01:00")]
test_df = spark.createDataFrame(test_data, ["value", "timestamp"])

# Create test stream
memory_stream = MemoryStream[Int]
memory_stream.addData(test_df)
stream = memory_stream.toDF()

# Apply transformations
result = transform_stream(stream)

# Assert results
assert result.count() == 2
```

Integration Testing

```
# Start test Kafka cluster
with KafkaCluster() as kafka:
    kafka.create_topic("test_topic")

    # Start stream
    query = (spark.readStream
      .format("kafka")
      .option("bootstrap.servers", kafka.broker)
      .load()
      .writeStream
      .format("memory")
      .queryName("test_query")
      .start())

    # Publish test data
    kafka.publish("test_topic", test_messages)

    # Verify
    assert spark.table("test_query").count() > 0
```

14. Common Pitfalls & Solutions

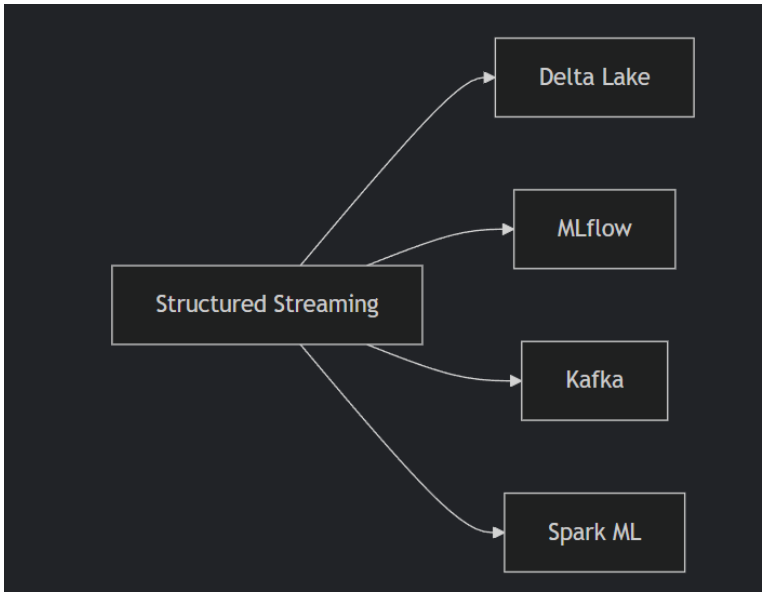
Issue	Solution
Stuck streams	Check watermark settings, increase timeout
State store growth	Add watermark, set TTL
Sink bottlenecks	Repartition before write, optimize destination
Schema evolution	Use mergeSchema option
Checkpoint errors	Clean corrupt checkpoints, restart from offset

15. Future of Structured Streaming

Upcoming Features

- **Arbitrary Stateful Processing**: Enhanced state management APIs
- **Improved Python Stateful APIs**: Better support for complex logic
- **Enhanced Monitoring**: More detailed metrics
- **Dynamic Resource Allocation**: Auto-scale executors based on load

Streaming Ecosystem Integration



16. Complete Cheat Sheet

Core Operations

```
# Read stream
df = spark.readStream.format("source").options().load()

# Write stream
(df.writeStream
  .format("sink")
  .outputMode("mode")
  .option("checkpointLocation", "...")
  .start())

# Manage streams
spark.streams.active  # List queries
query.stop()          # Stop gracefully
```

Common Options

```
python
.option("maxFilesPerTrigger", 100)      # File source
.option("maxOffsetsPerTrigger", 50000)  # Kafka
.option("truncate", False)              # Console sink
.option("mergeSchema", "true")          # Delta sink
```

17. Learning Resources

Official Documentation

- Structured Streaming Programming Guide
- Databricks Streaming Documentation
- Kafka Integration Guide

Recommended Books

- "Spark: The Definitive Guide" - Chapters on Streaming
- "Designing Data-Intensive Applications" - Streaming Systems Theory