Real-Time Streaming with Azure Databricks and Event Hubs

Comprehensive Step-by-Step Guide: Real-Time Streaming with Azure Databricks and Event Hubs

This guide will walk you through building an end-to-end real-time analytics streaming solution using Azure Databricks, Event Hubs, and Power BI.

Project Overview

You'll learn to:

- Stream data from Azure Event Hubs into Azure Databricks
- Process streaming data using Spark Structured Streaming
- Implement the Medallion Architecture (Bronze, Silver, Gold layers)
- Create near real-time reports in Power BI

Acceptable Latency:

- Event Hubs to Databricks: Up to a few minutes
- Power BI refresh: Up to 20 minutes

Prerequisites

Before starting, ensure you have:

1. **Azure Account and Subscription** (active)
2. **Azure Databricks Workspace** with Unity Catalog enabled
3. **Power BI Desktop** (Windows OS required for final step)
4. **Basic knowledge** of Python, PySpark, and SQL

Phase 1: Azure Event Hubs Setup

Step 1: Create Event Hubs Namespace

1. Navigate to **Azure Portal**
2. Search for **"Event Hubs"** in the search bar
3. Click **"+ Create"**
4. Configure the namespace:
   - **Resource Group:** Select or create one
   - **Namespace name:** `eh-databricks-streaming-p3` (or your preferred name)

- **Location:** Choose a region close to you (e.g., West Europe)
- **Pricing tier:** Select **Basic** (most cost-effective for this demo)
- **Throughput units:** Set to **1** (minimum)

5. Click **"Review + Create"**, then **"Create"**
6. Wait for deployment to complete
7. Click **"Go to resource"**

Step 2: Create an Event Hub Instance

1. In your Event Hubs namespace, navigate to **Entities > Event Hubs** (left sidebar)
2. Click **"+ Event Hub"**
3. Configure:
   - **Name:** `eh-p3`
   - **Partition Count: 2** (default minimum)
   - **Message Retention: 1** hour (sufficient for demo)
4. Click **"Review + Create"**, then **"Create"**

Step 3: Generate Shared Access Policy

1. Click into your newly created Event Hub (`eh-p3`)
2. Navigate to **Settings > Shared access policies**
3. Click **"+ Add"**
4. Configure:
   - **Policy name:** `databricks`
   - **Permissions:** Check **Listen** only
5. Click **"Create"**
6. Click on the `databricks` policy
7. **Copy the "Connection string–primary key"** (you'll need this later)

Step 4: Test Data Generation (Optional)

1. In your Event Hub, navigate to **Features > Generate Data (Preview)**
2. Select **Content Type:** `JSON`
3. Paste this sample JSON payload:

```
{
  "temperature": 20,
  "humidity": 65,
  "windSpeed": 15,
  "windDirection": "North",
  "precipitation": 0,
  "conditions": "Partly Cloudy"
}
```

4. Click **"Send"**
5. Verify the event appears in the data preview

6. Note the structure: `message_body` (your JSON) and metadata columns (`offset`, `sequence_number`, `partition_id`, `enqueued_time`)

Phase 2: Azure Databricks Configuration

Step 5: Create Databricks Compute Cluster

1. Navigate to your **Azure Databricks workspace**
2. Go to **Compute** (left sidebar)
3. Click **"Create Compute"**
4. Configure:
   - **Policy:** Unrestricted
   - **Cluster mode: Single Node**
   - **Access mode: Single User**
   - **Databricks Runtime Version: 12.2 LTS** (important for Event Hubs connector compatibility)
   - **Node type:** Standard_DS3_v2
   - **Auto termination: 10 minutes** (change from default 120)
5. Verify **Unity Catalog is supported** in the summary
6. Click **"Create Compute"**

Step 6: Install Azure Event Hubs Connector Library

1. While the cluster is starting, click on your cluster name
2. Navigate to the **Libraries** tab
3. Click **"Install new"**
4. Select **"Maven"** as source
5. In the search dropdown, select **"Maven Central"**
6. Search for: `eventhubs-spark` (single word)
7. Select `azure-eventhubs-spark_2.12` (matches runtime version)
8. Choose the **latest release** (e.g., `2.3.22`)
9. Click **"Install"**
10. Wait for installation to complete (library status should show "Installed")

> **Note:** If future versions support later runtimes, adjust accordingly.

`com.microsoft.azure:azure-eventhubs-spark_2.12:2.3.22`

Azure Databricks Configuration Required

- Single Node Compute Cluster: `12.2 LTS (includes Apache Spark 3.3.2, Scala 2.12)`
- Maven Library installed on Compute Cluster: `com.microsoft.azure:azure-eventhubs-spark_2.12:2.3.22`

Phase 3: Import Databricks Notebook

Step 7: Access GitHub Resources

1. Open the GitHub repository: Real-Time Streaming with Azure Databricks
   - Look for `sample_data.json` (sample JSON structure)
   - Look for `P3-DB_Streaming.ipynb`
2. Copy the **raw URL** of the `.ipynb` notebook file

Step 8: Import Notebook to Databricks

1. In Databricks workspace, navigate to **Workspace**
2. Go to your user folder
3. Click the **three dots (...)** next to your folder > **Import**
4. Paste the **notebook URL** from GitHub
5. Click **"Import"**
6. The notebook should now appear in your workspace

Step 9: Attach Cluster to Notebook

1. Open the imported notebook
2. At the top, click **"Connect"** dropdown
3. Select your cluster (the one you created in Step 5)
4. Ensure the cluster is **running** (green indicator)

Phase 4: Bronze Layer - Raw Data Ingestion

Step 10: Import Required Libraries

Run the first cell in your notebook:

```python
from pyspark.sql.functions import *
from pyspark.sql.types import *
```

Step 11: Create Medallion Architecture Structure

Run this cell to create the catalog and schemas:

PYTHON

```python
try:
    spark.sql("""
        CREATE CATALOG IF NOT EXISTS streaming
        MANAGED LOCATION 'abfss://metastore-p3@yagamiazdbsa.dfs.core.windows.net'
    """)
    print("Catalog 'streaming' created successfully")
except Exception as e:
    print(f"Catalog creation error: {e}")



try:
    spark.sql("CREATE SCHEMA IF NOT EXISTS streaming.bronze")
    print("Schema 'bronze' created")
except Exception as e:
    print(f"Bronze schema error: {e}")

try:
    spark.sql("CREATE SCHEMA IF NOT EXISTS streaming.silver")
    print("Schema 'silver' created")
except Exception as e:
    print(f"Silver schema error: {e}")

try:
    spark.sql("CREATE SCHEMA IF NOT EXISTS streaming.gold")
    print("Schema 'gold' created")
except Exception as e:
    print(f"Gold schema error: {e}")
```

Verify: Check the **Catalog Explorer** (left sidebar) - you should now see:

- `streaming` catalog
- `bronze`, `silver`, `gold` schemas (empty)

Step 12: Configure Event Hubs Connection

In the notebook, locate the connection configuration cell and update:

```python
# Paste your Event Hubs connection string here
connectionString = "Endpoint=sb://eh-databricks-streaming-p3.servicebus.windows.net/;SharedAccessKeyName=databricks;SharedAccessKey=w84H+W2xuNc0XkOt8XOo8amQ1Hrnz7B7y+AEhOoI4IU=;EntityPath=eh-p3"
eventHubName = "eh-p3"

ehConf = {
  'eventhubs.connectionString' :
sc._jvm.org.apache.spark.eventhubs.EventHubsUtils.encrypt(connectionString),
  'eventhubs.eventHubName': eventHubName
}
```

> **Important:** Replace `[your-namespace]` and `[your-key]` with your actual values from Step 3.

Step 13: Initialize the Stream (Test)

First, test reading the stream without persisting:

```python
# Reading stream: Load data from Azure Event Hub into DataFrame 'df' using the previously configured settings
df = spark.readStream \
    .format("eventhubs") \
    .options(**ehConf) \
    .load() \

# Displaying stream: Show the incoming streaming data for visualization and debugging purposes
df.display()
```

Expected Behavior:

- Stream initializes but shows no data (because no new events since stream started)
- Go to Azure Event Hubs > Generate Data
- Send a test event (use the JSON from Step 4)
- Return to notebook - you should see the event appear
- **Note:** Data is in **binary format** in the `body` column

Step 14: Write Stream to Bronze Table

Now persist the data:

```python
# Reading stream: Load data from Azure Event Hub into DataFrame 'df' using the previously configured settings
df = spark.readStream \
    .format("eventhubs") \
    .options(**ehConf) \
    .load() \

# Displaying stream: Show the incoming streaming data for visualization and debugging purposes
df.display()

# Writing stream: Persist the streaming data to a Delta table 'streaming.bronze.weather' in 'append' mode with checkpointing
df.writeStream\
    .option("checkpointLocation", "/mnt/streaming/bronze/weather")\
    .outputMode("append")\
    .format("delta")\
    .toTable("streaming.bronze.weather")
```

What's happening:

- `checkpointLocation`: Stores streaming state for fault tolerance
- `outputMode("append")`: New data is continuously appended
- Data is written to `streaming.bronze.weather` Delta table

Step 15: Verify Bronze Data

1. Generate multiple events in Event Hubs (vary the temperature, conditions, etc.)
2. Each event should appear in the stream display
3. Query the table to verify persistence:

```sql
%sql
SELECT * FROM streaming.bronze.weather
```

Check DBFS: Navigate to **Catalog > Browse DBFS > mnt > streaming > bronze > weather** to see checkpoint files.

> **Critical Warning:** Streams run indefinitely. Always **interrupt** (stop) streams when not in use and **manually terminate** your cluster to avoid costs!

Phase 5: Silver Layer - Data Transformation

Step 16: Define JSON Schema

Before processing, define the structure of your JSON payload:

PYTHON

```python
# Defining the schema for the JSON object

json_schema = StructType([
    StructField("temperature", IntegerType()),
    StructField("humidity", IntegerType()),
    StructField("windSpeed", IntegerType()),
    StructField("windDirection", StringType()),
    StructField("precipitation", IntegerType()),
    StructField("conditions", StringType())
])
```

Step 17: Transform Bronze to Silver

Read from Bronze table and transform:

PYTHON

```python
# Reading and Transforming: Load streaming data from the 'streaming.bronze.weather'
Delta table, cast 'body' to string, parse JSON, and select specific fields
df = spark.readStream\
    .format("delta")\
    .table("streaming.bronze.weather")\
    .withColumn("body", col("body").cast("string"))\
    .withColumn("body",from_json(col("body"), json_schema))\
    .select("body.temperature", "body.humidity", "body.windSpeed",
"body.windDirection", "body.precipitation", "body.conditions",
col("enqueuedTime").alias('timestamp'))

# Displaying stream: Visualize the transformed data in the DataFrame for
verification and analysis
df.display()
```

Transformations explained:

1. **Cast to string:** Makes binary data human-readable
2. **Parse JSON:** Converts string to structured JSON object
3. **Select columns:** Extracts individual fields using dot notation
4. **Alias:** Renames `enqueuedTime` to `timestamp`

Step 18: Write to Silver Table

PYTHON

```python
# Writing stream: Save the transformed data to the 'streaming.silver.weather' Delta table in 'append' mode with checkpointing for data reliability
df.writeStream\
    .option("checkpointLocation", "/mnt/streaming/silver/weather")\
    .outputMode("append")\
    .format("delta")\
    .toTable("streaming.silver.weather")
```

Step 19: Verify Silver Layer

1. Generate a new event in Event Hubs (e.g., temperature = 25)
2. Check Bronze stream - should show binary data
3. Check Silver stream - should show parsed, tabular data
4. Query Silver table:

SQL

```sql
%sql
SELECT * FROM streaming.silver.weather
ORDER BY timestamp DESC
```

Key difference: Silver stream persists even after restart because it reads from a Delta table (Bronze), not directly from Event Hubs.

Phase 6: Gold Layer - Aggregation & Refinement

Step 20: Implement Windowed Aggregations

Create 5-minute rolling aggregations with watermarking:

```python
# Aggregating Stream: Read from 'streaming.silver.weather', apply watermarking and windowing, and calculate average weather metrics
df = spark.readStream\
    .format("delta")\
    .table("streaming.silver.weather")\
    .withWatermark("timestamp", "5 minutes") \
    .groupBy(window("timestamp", "5 minutes")) \
    .agg(avg("temperature").alias('temperature'), avg("humidity").alias('humidity'), avg("windSpeed").alias('windSpeed'), avg("precipitation").alias('precipitation'))\
   .select('window.start', 'window.end', 'temperature', 'humidity', 'windSpeed', 'precipitation')

# Displaying Aggregated Stream: Visualize aggregated data for insights into weather trends
df.display()
```

Understanding Watermarking:

- **Purpose:** Handles late-arriving data gracefully
- **Logic:** Window closes only when `(max_event_time - watermark) > window_end`
- **Example:**
  - Window: 10:50-11:00
  - Watermark: 5 minutes
  - Latest event: 11:02
  - Calculation: 11:02 - 5 min = 10:57 (not > 11:00, so window stays open)
  - Late event at 10:53 arrives → still included!
  - Window closes when an event with timestamp > 11:05 arrives

Step 21: Write to Gold Table

```python
# Writing Aggregated Stream: Store the aggregated data in 'streaming.gold.weather_aggregated' with checkpointing for data integrity
df.writeStream\
    .option("checkpointLocation", "/mnt/streaming/weather_summary")\
    .outputMode("append")\
    .format("delta")\
    .toTable("streaming.gold.weather_summary")
```

Step 22: Test Gold Layer

1. Generate 2-3 events in quick succession (within 5 minutes)

2. Check Gold stream - should show aggregated averages
3. Generate another event after 5+ minutes
4. Should create a new window

Query closed windows only:

```sql
%sql
SELECT * FROM streaming.gold.weather_summary
ORDER BY window_start
```

> **Note:** Only **closed windows** are persisted. Open windows won't appear in the table until they close (per watermark rules).

Phase 7: Power BI Integration

Step 23: Use Partner Connect

1. In Databricks, scroll to **Partner Connect** (left sidebar)
2. Search for or select **Power BI**
3. Select your **interactive compute cluster** (not SQL endpoint)
4. Click **"Download connection file"**
5. Save the `.pbids` file to your **Downloads** folder

Step 24: Open Connection in Power BI Desktop

> **Requirement:** Windows OS with Power BI Desktop installed

1. Navigate to **Downloads** folder
2. Double-click the `.pbids` file
3. Power BI Desktop opens automatically
4. Sign in using **Azure Active Directory** when prompted
5. A dialog appears showing your Unity Catalog hierarchy

Step 25: Select Data

1. Expand the **streaming** catalog
2. Expand the **silver** schema
3. Check the **weather** table (easier for demo than gold due to watermark delay)
4. Click **"Load"**
5. Verify **Storage Mode** shows **DirectQuery** (bottom of screen)

> **DirectQuery explained:** No data is imported. Power BI sends queries to Databricks in real-time. Visuals refresh based on cluster performance.

Step 26: Create a Table Visual

1. Click the **Table** visual icon
2. Select all fields from `weather`:
   - `timestamp`
   - `temperature`
   - `humidity`
   - `wind_speed`
   - `wind_direction`
   - `precipitation`
   - `conditions`
3. Data should populate in the table

Step 27: Test Real-Time Updates

1. Go back to **Azure Event Hubs > Generate Data**
2. Send a new event (e.g., temperature = 10, conditions = "Cloudy")
3. Check Databricks notebook - event should flow through Bronze → Silver
4. In **Power BI**, click **Refresh** (ribbon)
5. New row should appear in the table

Auto-refresh: DirectQuery visuals refresh every 15 minutes automatically. Manual refresh is faster for testing.

Phase 8: Cleanup & Best Practices

Step 28: Stop Streams

1. In your Databricks notebook, click **"Interrupt Execution"** (stop button)
2. This stops all running streams in the notebook

Step 29: Terminate Cluster

1. Navigate to **Compute**
2. Click the **Stop** icon next to your cluster
3. Confirm termination

> **Critical:** Streaming jobs prevent auto-termination. Always manually stop clusters to avoid unnecessary costs!

Step 30: Power BI Cleanup

If you're done with Power BI:

Option 1: Delete the report

- Close Power BI Desktop

- Delete the `.pbix` file

Option 2: Switch to Import Mode

- In Power BI Desktop, go to **Transform Data**
- Select all tables
- Change **Storage Mode** from "DirectQuery" to "Import"
- This prevents the report from auto-starting your Databricks cluster

---

Verification Checklist

Use this to confirm each phase is complete:

- [ ] Event Hubs namespace and instance created
- [ ] Shared access policy generated and connection string copied
- [ ] Databricks cluster created with runtime 12.2 LTS
- [ ] Event Hubs connector library installed
- [ ] Notebook imported and attached to cluster
- [ ] Streaming catalog with bronze/silver/gold schemas created
- [ ] Bronze stream reads from Event Hubs and writes to Delta table
- [ ] Silver stream transforms binary data to tabular format
- [ ] Gold stream creates windowed aggregations with watermarking
- [ ] Power BI connected via DirectQuery and displays real-time data
- [ ] All streams stopped and cluster terminated

Troubleshooting Tips

Stream shows no data:

- Ensure stream is running (not stopped)
- Generate a **new** event in Event Hubs (historical data isn't captured)
- Check Event Hubs connection string is correct

Library installation fails:

- Verify runtime version matches library (12.2 LTS ↔ `spark_2.12`)
- Restart cluster after installation

Power BI won't connect:

- Ensure you selected the **interactive cluster**, not SQL endpoint
- Verify Unity Catalog is enabled on your workspace
- Check you're using Windows OS with Power BI Desktop

High costs:

- Always stop streams with "Interrupt"
- Manually terminate cluster when done
- Use auto-termination settings (10 minutes)

Additional Resources

- **GitHub Repository:** Contains notebook and sample data
- **Watermarking Deep Dive:** Blog post linked in video description
- **Azure Event Hubs Docs:** https://docs.microsoft.com/azure/event-hubs/
- **Spark Structured Streaming Guide:** https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
- **Databricks Unity Catalog:** https://docs.databricks.com/data-governance/unity-catalog/

Congratulations! 🎉

You've successfully built an end-to-end real-time streaming analytics solution! You now understand:

- How to ingest streaming data with Azure Event Hubs
- Processing streams with Spark Structured Streaming
- Implementing the Medallion Architecture
- Handling late data with watermarking
- Creating near real-time reports in Power BI

Next Steps:

- Experiment with different window sizes and watermarks
- Add more complex transformations in Silver layer
- Implement additional aggregations in Gold layer
- Explore automated scheduling with Databricks Jobs