VNU-HCM University of Science

Faculty of Information Technology.

CSC14118 – Introduction to Big Data Analysis

Lab 4 Report.

Topic: Spark Streaming

Teachers	**Student Name**	**Student ID**
Mr Huỳnh Lâm Hải Đăng	Nguyễn Thiên Bảo	22127032
Mr Lê Ngọc Thành	Nguyễn Quang Huy	22127157
Mr Vũ Công Thành	Nguyễn Thanh Nam	22127286
Ms Nguyễn Ngọc Thảo	Bùi Minh Quân	22127342

Contents

1 Tasks assignment

No	Member	Task assigned	Completion
1	Nguyễn Thiên Bảo	Implement transform z-score, loading and bonus section	100%
2	Nguyễn Quang Huy	Setting up docker, implement extracting and transform moving	100%
3	Nguyễn Thanh Nam	Setting up docker, implement extracting and transform moving	100%
4	Bùi Minh Quân	Implement transform z-score, load and bonus section	100%

2 Introduction

This lab focuses on buiding a real-time data pipeline to perform streaming processing on cryptocurrency price data, particularly BTC/USDT from Binance. The goal is to implement an ETL (Extract-Transfrom-Load) pipeline using Apache Kafka, Apache Spark Structured Streaming and MongoDB.

3 Directory structure

- `docker-compose.yml`: defining services Kafka, Zookeeper, Spark Master and Worker, Mongo, and spark-client. It sets up the necessary environment to run Spark jobs with Kafka.
- src
  - Extract: `22127032.py`
  - Transform:
    - `22127032_transform.py`
    - `22127032_zscore.py`
  - Load: `22127032.py`
  - Bonus: `22127032.py`

4 Configuration

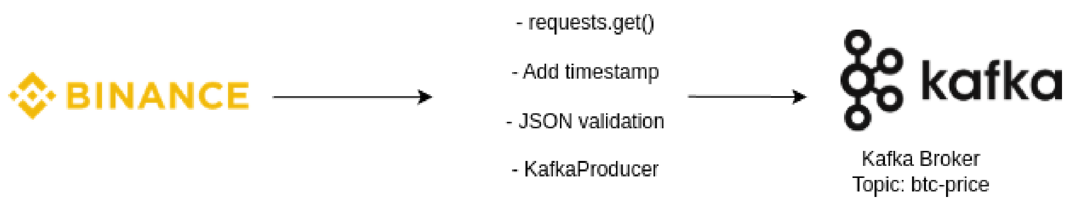
- In this lab, we used a Docker Compose file to start Spark, Apache Kafka, Apache Zookeeper, and MongoDB for the implementation.
- Follow the instructions in the README.md file to execute the lab.

5 Extract

Objective: Fetch real-time Bitcoin (BTC) to USDT price data from the Binance cryptocurrency exchange API. It then enriches this data with a timestamp and publishes it to a specified Apache Kafka topic for downstream processing.

Kafka Configuration:
- Broker External Address: localhost:9092
- Topic: `btc-price`

Program Architect:



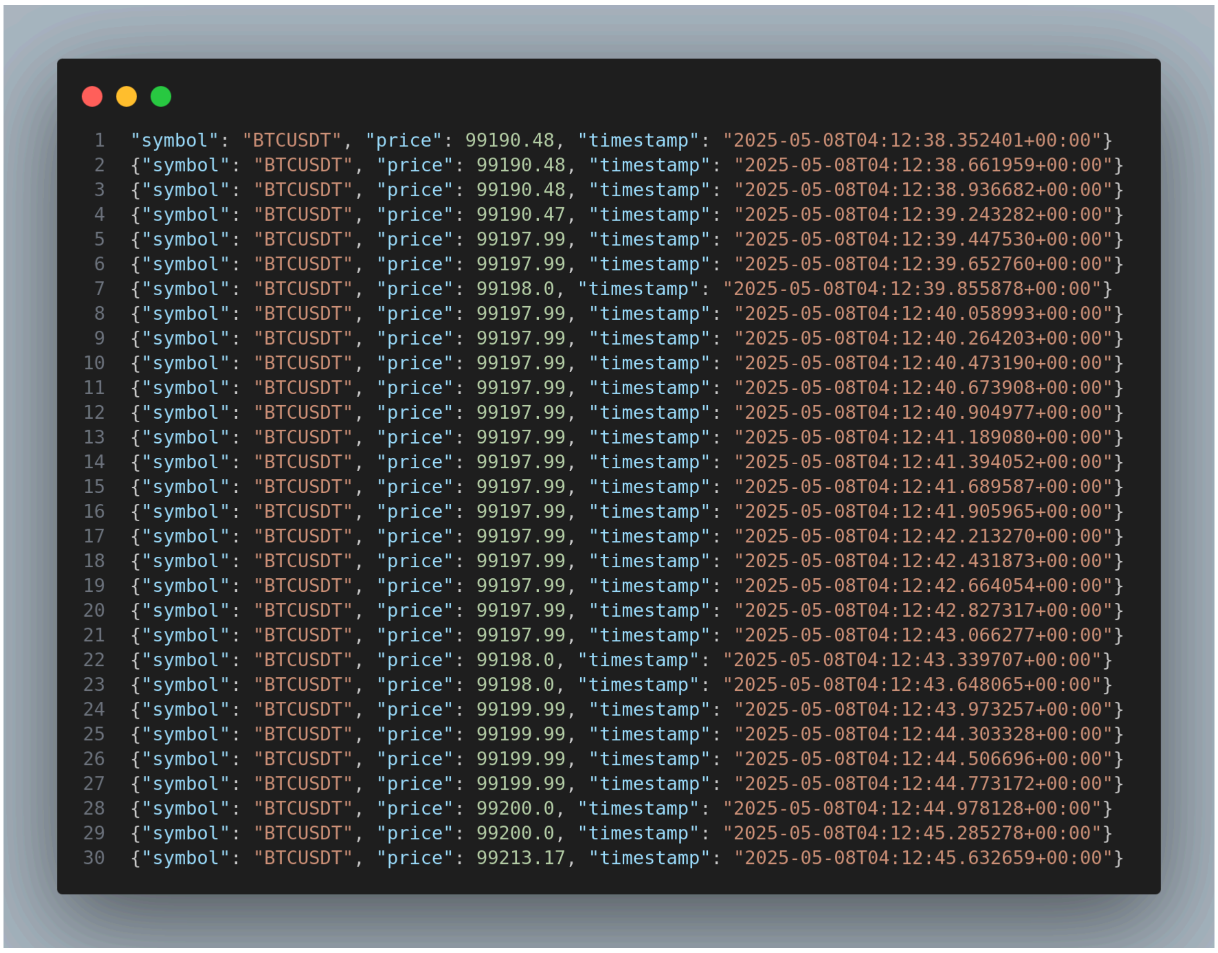
Components::
- Language: Python.
- Libraries:
  - `requests`: Used to make HTTP GET requests to the Binance API to retrieve cryptocurrency price data.
  - `json`: For data serialization.
  - `time`: Used for managing the frequency of API calls (time.sleep()) and for basic performance monitoring of the fetch operation (time.monotonic()).
  - `kafka-python`: Enables the script to send messages (BTC price data) to a Kafka topic. The messages are serialized to JSON format.
  - `datetime`: Utilized to generate UTC timestamps for each data record, ensuring a consistent time reference.

Implemented Methods (Core Logic):
- Fetching Data: Periodically calls the Binance API to get the latest BTC/USDT price.
- Timestamping: Adds a current UTC timestamp in ISO8601 format to each fetched record.
- Data Validation & Transformation: Checks for the presence of essential fields ("symbol", "price") in the API response and attempts to convert the price to a floating-point number. Records with invalid price formats are skipped.
- Producing to Kafka: Sends the processed data (symbol, price, timestamp) as a JSON message to the designated Kafka topic.
- Rate Limiting: Ensures that API calls are made at a controlled interval (FETCH_INTERVAL_SECONDS).

- Error Management: Catches and logs errors related to API connectivity, unexpected data formats, and Kafka operations, allowing the script to continue running or retry.

Results

First 30 lines of JSON formatted returned:



```
"symbol": "BTCUSDT", "price": 99190.48, "timestamp": "2025-05-08T04:12:38.352401+00:00"}
{"symbol": "BTCUSDT", "price": 99190.48, "timestamp": "2025-05-08T04:12:38.661959+00:00"}
{"symbol": "BTCUSDT", "price": 99190.48, "timestamp": "2025-05-08T04:12:38.936682+00:00"}
{"symbol": "BTCUSDT", "price": 99190.47, "timestamp": "2025-05-08T04:12:39.243282+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:39.447530+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:39.652760+00:00"}
{"symbol": "BTCUSDT", "price": 99198.0, "timestamp": "2025-05-08T04:12:39.855878+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:40.058993+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:40.264203+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:40.473190+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:40.673908+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:40.904977+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:41.189080+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:41.394052+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:41.689587+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:41.905965+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:42.213270+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:42.431873+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:42.664054+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:42.827317+00:00"}
{"symbol": "BTCUSDT", "price": 99197.99, "timestamp": "2025-05-08T04:12:43.066277+00:00"}
{"symbol": "BTCUSDT", "price": 99198.0, "timestamp": "2025-05-08T04:12:43.339707+00:00"}
{"symbol": "BTCUSDT", "price": 99198.0, "timestamp": "2025-05-08T04:12:43.648065+00:00"}
{"symbol": "BTCUSDT", "price": 99199.99, "timestamp": "2025-05-08T04:12:43.973257+00:00"}
{"symbol": "BTCUSDT", "price": 99199.99, "timestamp": "2025-05-08T04:12:44.303328+00:00"}
{"symbol": "BTCUSDT", "price": 99199.99, "timestamp": "2025-05-08T04:12:44.506696+00:00"}
{"symbol": "BTCUSDT", "price": 99199.99, "timestamp": "2025-05-08T04:12:44.773172+00:00"}
{"symbol": "BTCUSDT", "price": 99200.0, "timestamp": "2025-05-08T04:12:44.978128+00:00"}
{"symbol": "BTCUSDT", "price": 99200.0, "timestamp": "2025-05-08T04:12:45.285278+00:00"}
{"symbol": "BTCUSDT", "price": 99213.17, "timestamp": "2025-05-08T04:12:45.632659+00:00"}
```

6 Transform

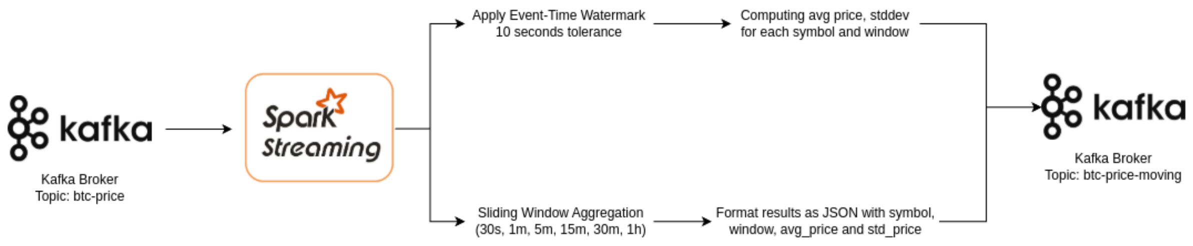
6.1 Moving Average & Standard Deviation

Objective: Consumes BTC price data from one Kafka topic, calculates various moving window statistics (average and standard deviation of price) over different timeframes, and then publishes these aggregated statistics to another Kafka topic.

Kafka Configuration:

- Kafka Topics:
  - Input: `btc-price`
  - Output: `btc-price-moving`

Program Architect:

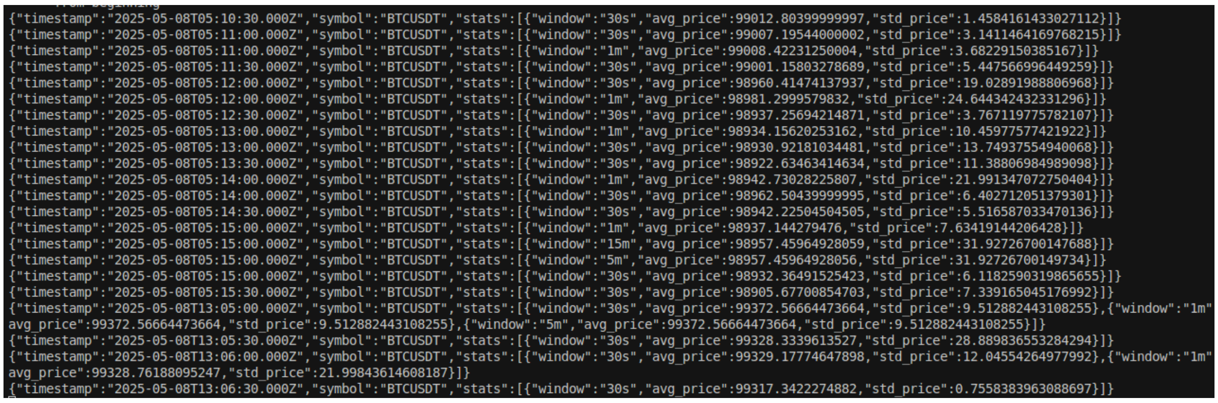

kafka
Kafka Broker
Topic: btc-price
Spark Streaming
Apply Event-Time Watermark
10 seconds tolerance
Computing avg price, stddev
for each symbol and window
Sliding Window Aggregation
(30s, 1m, 5m, 15m, 30m, 1h)
Format results as JSON with symbol,
window, avg_price and std_price
kafka
Kafka Broker
Topic: btc-price-moving

Components:

- Language: Python
- Framework: Pyspark (Structured Streaming)
- Libraries Used:
  - `pyspark.sql.SparkSession`: To initialize and configure the Spark application.
  - `pyspark.sql.functions`: For aggregation functions such as `avg`, `stddev`, and JSON formatting.
  - `pyspark.sql.types`: For schema definitions.

Implemented Methods (Core Logic):
- Stream Ingestion: Reads a continuous stream of BTC price updates from Kafka.
- Data Transformation: Parses JSON, casts types, and prepares data for windowing.
- Watermarking: Manages out-of-order data based on event timestamps.
- Multi-Window Statistical Calculation: Computes average and standard deviation of prices over multiple, concurrent time windows (30s, 1m, 5m, 15m, 30m, 1h).
- Stream Union: Merges the results from different window calculations into one stream.
- Final Result Aggregation: Consolidates statistics for each symbol and timestamp, presenting all calculated windowed metrics together.
- Stream Output: Publishes the aggregated statistics in JSON format to a new Kafka topic.

Result:



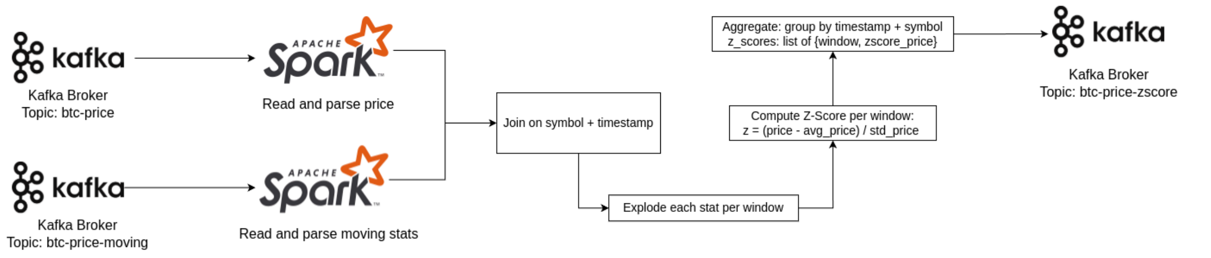
```
{"timestamp":"2025-05-08T05:10:30.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":99012.80399999997,"std_price":1.4584161433027112}]}
{"timestamp":"2025-05-08T05:11:00.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":99007.19544000002,"std_price":3.141146416976821 5}]}
{"timestamp":"2025-05-08T05:11:00.000Z","symbol":"BTCUSDT","stats":[{"window":"1m","avg_price":99008.42231250004,"std_price":3.68229150385167}]}
{"timestamp":"2025-05-08T05:11:30.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":99001.15803278689,"std_price":5.447566996449259}]}
{"timestamp":"2025-05-08T05:12:00.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":98960.41474137937,"std_price":19.02891988806968}]}
{"timestamp":"2025-05-08T05:12:00.000Z","symbol":"BTCUSDT","stats":[{"window":"1m","avg_price":98981.2999579832,"std_price":24.644342432331296}]}
{"timestamp":"2025-05-08T05:12:30.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":98937.25694214871,"std_price":3.767119775782107}]}
{"timestamp":"2025-05-08T05:13:00.000Z","symbol":"BTCUSDT","stats":[{"window":"1m","avg_price":98934.15620253162,"std_price":10.45977577421922}]}
{"timestamp":"2025-05-08T05:13:00.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":98930.92181034481,"std_price":13.74937554940068}]}
{"timestamp":"2025-05-08T05:13:30.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":98922.63463414634,"std_price":11.38806984989098}]}
{"timestamp":"2025-05-08T05:14:00.000Z","symbol":"BTCUSDT","stats":[{"window":"1m","avg_price":98942.73028225807,"std_price":21.991347072750404}]}
{"timestamp":"2025-05-08T05:14:00.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":98962.50439999995,"std_price":6.402712051379301}]}
{"timestamp":"2025-05-08T05:14:30.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":98942.22504504505,"std_price":5.516587033470136}]}
{"timestamp":"2025-05-08T05:15:00.000Z","symbol":"BTCUSDT","stats":[{"window":"1m","avg_price":98937.144279476,"std_price":7.63419144206428}]}
{"timestamp":"2025-05-08T05:15:00.000Z","symbol":"BTCUSDT","stats":[{"window":"15m","avg_price":98957.45964928059,"std_price":31.9272670014768 8}]}
{"timestamp":"2025-05-08T05:15:00.000Z","symbol":"BTCUSDT","stats":[{"window":"5m","avg_price":98957.45964928056,"std_price":31.92726700149734}]}
{"timestamp":"2025-05-08T05:15:00.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":98932.36491525423,"std_price":6.1182590319865655}]}
{"timestamp":"2025-05-08T05:15:30.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":98905.67700854703,"std_price":7.339165045176992}]}
{"timestamp":"2025-05-08T13:05:00.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":99372.56664473664,"std_price":9.512882443108255},{"window":"1m"
avg_price":99372.56664473664,"std_price":9.512882443108255},{"window":"5m","avg_price":99372.56664473664,"std_price":9.512882443108255}]}
{"timestamp":"2025-05-08T13:05:30.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":99328.3339613527,"std_price":28.889836553284294}]}
{"timestamp":"2025-05-08T13:06:00.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":99329.17774647898,"std_price":12.04554264977992},{"window":"1m"
avg_price":99328.76188095247,"std_price":21.99843614608187}]}
{"timestamp":"2025-05-08T13:06:30.000Z","symbol":"BTCUSDT","stats":[{"window":"30s","avg_price":99317.3422274882,"std_price":0.7558383963088697}]}
```

6.2 Z-Score Computation

Objective: Calculate the Z-score of current price against each statistical window.

Kafka Configuration:
- Kafka Topics:
  - Input: `btc-price`, `btc-price-moving`
  - Output: `btc-price-zscore`

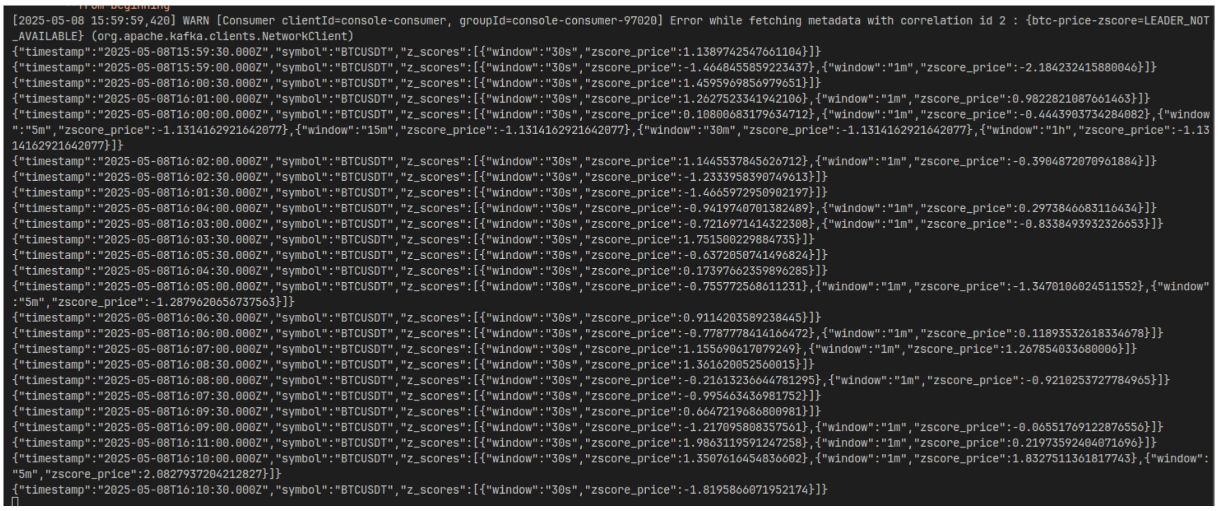
Program Architect:


kafka
Kafka Broker
Topic: btc-price
Spark
Read and parse price
kafka
Kafka Broker
Topic: btc-price-moving
Spark
Read and parse moving stats
Join on symbol + timestamp
Explode each stat per window
Compute Z-Score per window:
z = (price - avg_price) / std_price
Aggregate: group by timestamp + symbol
z_scores: list of {window, zscore_price}
kafka
Kafka Broker
Topic: btc-price-zscore

Implementation:

- Framework: Pyspark (Structured Streaming)
- Libraries Used:
  - `pyspark.sql.SparkSession`: to initialize and configure the Spark application.
  - `pyspark.sql.functions`: for aggregation functions such as `avg`, `stddev`, and JSON formatting.
  - `pyspark.sql.types`: for schema definitions.
- This phase reads two streams from Kafka:
  - btc-price – raw BTC price records with timestamps.
  - btc-price-moving – moving average and standard deviation data per time window.
- Each stream is parsed from JSON and structured using predefined schemas. The two data streams are then joined on both timestamp and symbol to ensure accurate pairing of real-time and aggregated data.
- For each window in the aggregated statistics, the Z-score is computed as:
  - `Z = (price - avg_price) / std_price`
- This value shows how many standard deviations the current price is from the average. The output is then restructured into a new JSON format containing an array of Z-scores for each window.
- Finally, the resulting stream is written to a new Kafka topic btc-price-zscore, ready to be consumed or stored.

Result:



```
    --from-beginning
[2025-05-08 15:59:59,420] WARN [Consumer clientId=console-consumer, groupId=console-consumer-97020] Error while fetching metadata with correlation id 2 : {btc-price-zscore=LEADER_NOT
_AVAILABLE} (org.apache.kafka.clients.NetworkClient)
{"timestamp":"2025-05-08T15:59:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.1389742547661104}]}
{"timestamp":"2025-05-08T15:59:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-1.4648455859223437},{"window":"1m","zscore_price":-2.184232415880046}]}
{"timestamp":"2025-05-08T16:00:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.4595969856979651}]}
{"timestamp":"2025-05-08T16:01:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.2627523341942106},{"window":"1m","zscore_price":0.9822821087661463}]}
{"timestamp":"2025-05-08T16:00:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":0.10800683179634712},{"window":"1m","zscore_price":-0.4443903734284082},{"window
":"5m","zscore_price":-1.1314162921642077},{"window":"15m","zscore_price":-1.1314162921642077},{"window":"30m","zscore_price":-1.1314162921642077},{"window":"1h","zscore_price":-1.13
14162921642077}]}
{"timestamp":"2025-05-08T16:02:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.1445537845626712},{"window":"1m","zscore_price":-0.3904872070961884}]}
{"timestamp":"2025-05-08T16:02:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-1.2333958390749613}]}
{"timestamp":"2025-05-08T16:01:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-1.4665972950902197}]}
{"timestamp":"2025-05-08T16:04:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-0.9419740701382489},{"window":"1m","zscore_price":0.2973846683116434}]}
{"timestamp":"2025-05-08T16:03:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-0.7216971414322308},{"window":"1m","zscore_price":-0.8338493932326653}]}
{"timestamp":"2025-05-08T16:03:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.751500229884735}]}
{"timestamp":"2025-05-08T16:05:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-0.6372050741496824}]}
{"timestamp":"2025-05-08T16:04:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":0.17397662359896285}]}
{"timestamp":"2025-05-08T16:05:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-0.755772568611231},{"window":"1m","zscore_price":-1.3470106024511552},{"window"
:"5m","zscore_price":-1.2879620656737563}]}
{"timestamp":"2025-05-08T16:06:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":0.9114203589238445}]}
{"timestamp":"2025-05-08T16:06:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-0.7787778414166472},{"window":"1m","zscore_price":0.11893532618334678}]}
{"timestamp":"2025-05-08T16:07:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.155690617079249},{"window":"1m","zscore_price":1.267854033680006}]}
{"timestamp":"2025-05-08T16:08:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.361620052560015}]}
{"timestamp":"2025-05-08T16:08:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-0.21613236644781295},{"window":"1m","zscore_price":-0.9210253727784965}]}
{"timestamp":"2025-05-08T16:07:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-0.995463436981752}]}
{"timestamp":"2025-05-08T16:09:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":0.6647219686800981}]}
{"timestamp":"2025-05-08T16:09:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-1.217095808357561},{"window":"1m","zscore_price":-0.06551769122876556}]}
{"timestamp":"2025-05-08T16:11:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.9863119591247258},{"window":"1m","zscore_price":0.21973592404071696}]}
{"timestamp":"2025-05-08T16:10:00.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":1.3507616454836602},{"window":"1m","zscore_price":1.8327511361817743},{"window":
"5m","zscore_price":2.0827937204212827}]}
{"timestamp":"2025-05-08T16:10:30.000Z","symbol":"BTCUSDT","z_scores":[{"window":"30s","zscore_price":-1.8195866071952174}]}
```

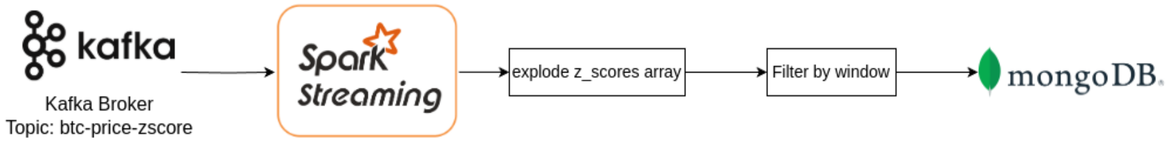
7 Load

Objective: Load the computed Z-scores into MongoDB collections for long-term storage and query.

Configuration:

- MongoDB Configuration:
  - URI: `mongodb://mongo:27017`
  - Database: btc
- Kafka configuration
  - Input topic: `btc-price-zscore`

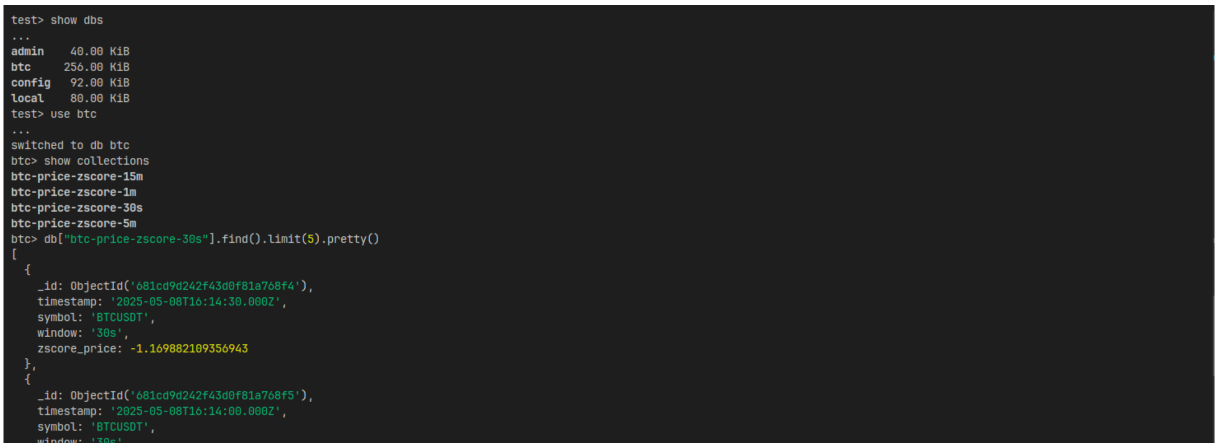
Program Architect:



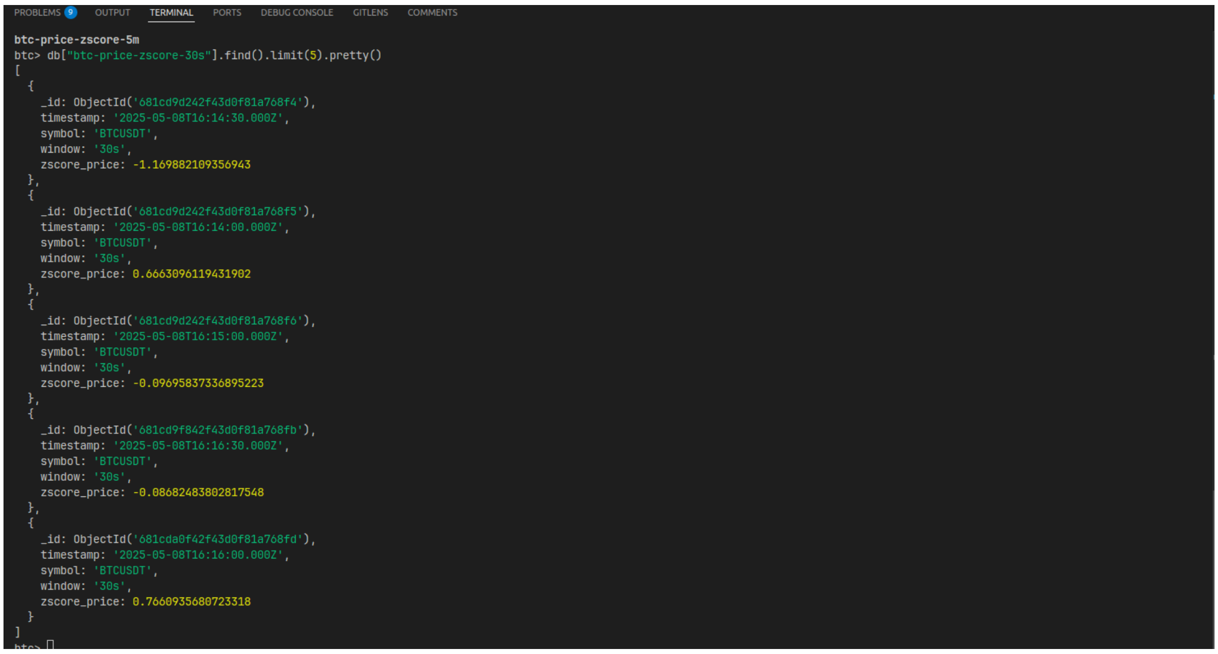
Implementation:

- Framework: PySpark + MongoDB Spark Connector
- Libraries Used:
  - pyspark.sql.SparkSession and functions
  - MongoDB Spark Connector to write data to collections
- First, the stream from Kafka topic `btc-price-zscore` is read and then the JSON is deserialized into structured rows. It then explodes the array of z__scores so each score is written to a dedicated MongoDB collection.
- Each collection corresponds to one window size, e.g., btc-price-zscore-30s, btc-price-zscore-1m, etc. These collections are dynamically filtered and written in parallel using writeStream.

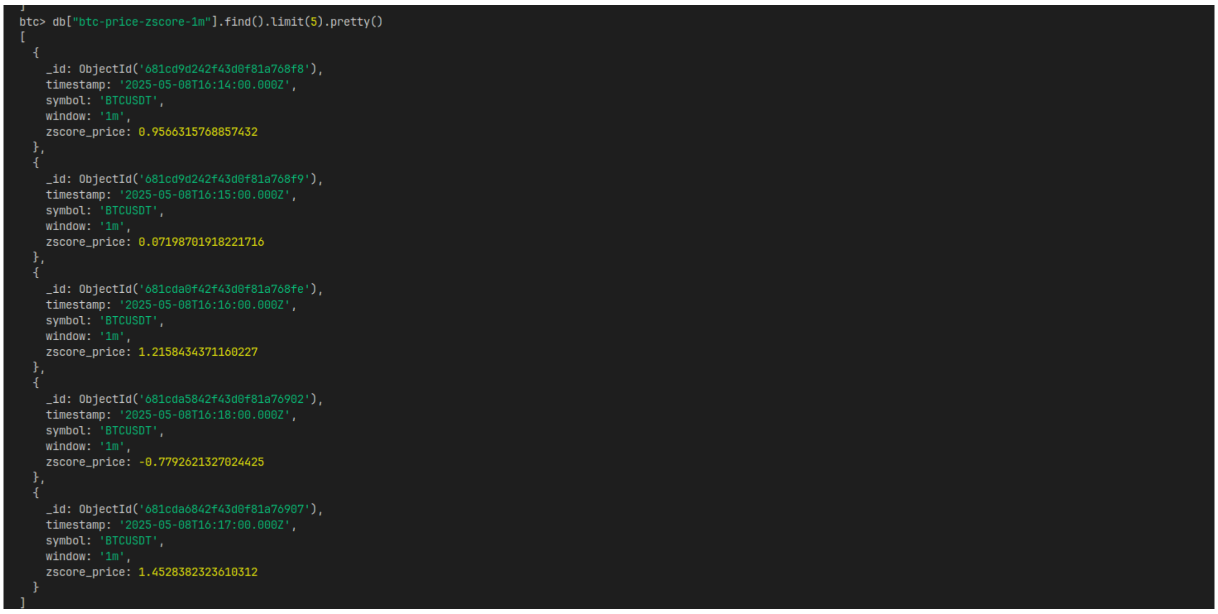
Result:



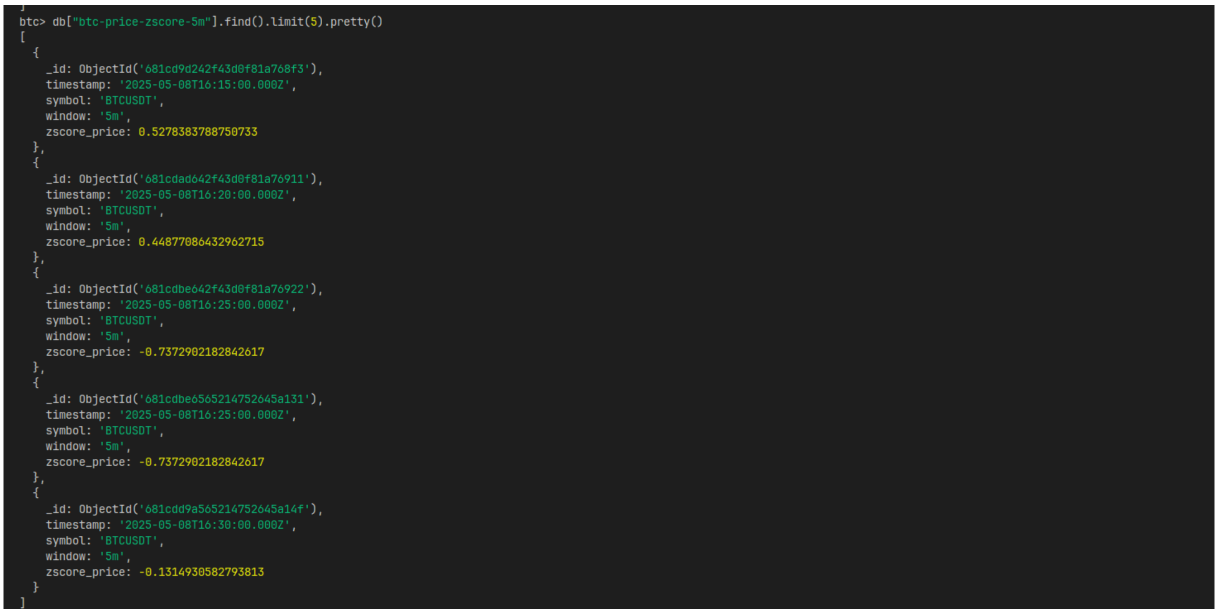
```
test> show dbs
...
admin    40.00 KiB
btc     256.00 KiB
config   92.00 KiB
local    80.00 KiB
test> use btc
...
switched to db btc
btc> show collections
btc-price-zscore-15m
btc-price-zscore-1m
btc-price-zscore-30s
btc-price-zscore-5m
btc> db["btc-price-zscore-30s"].find().limit(5).pretty()
[
  {
    _id: ObjectId('681cd9d242f43d0f81a768f4'),
    timestamp: '2025-05-08T16:14:30.000Z',
    symbol: 'BTCUSDT',
    window: '30s',
    zscore_price: -1.169882109356943
  },
  {
    _id: ObjectId('681cd9d242f43d0f81a768f5'),
    timestamp: '2025-05-08T16:14:00.000Z',
    symbol: 'BTCUSDT',
    window: '30s'
```



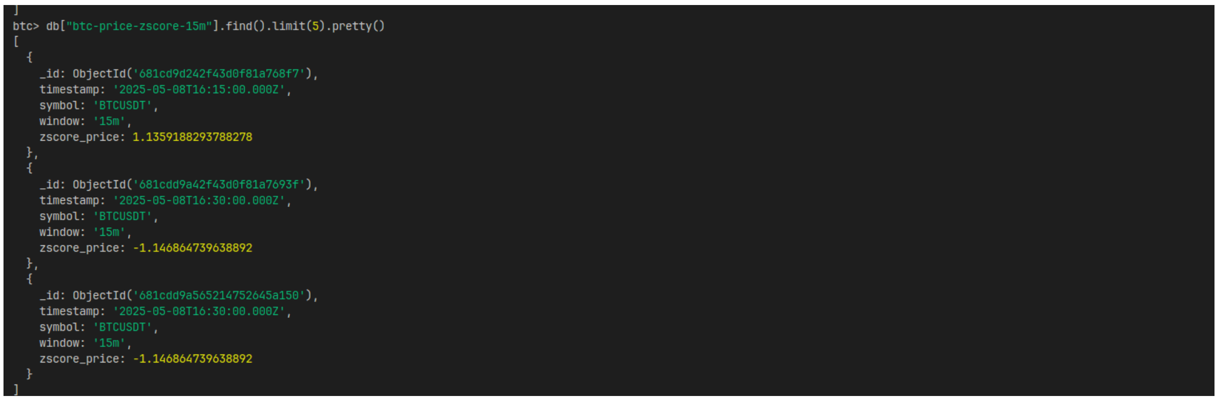
```
btc-price-zscore-5m
btc> db["btc-price-zscore-30s"].find().limit(5).pretty()
[
  {
    _id: ObjectId('681cd9d242f43d0f81a768f4'),
    timestamp: '2025-05-08T16:14:30.000Z',
    symbol: 'BTCUSDT',
    window: '30s',
    zscore_price: -1.169882109356943
  },
  {
    _id: ObjectId('681cd9d242f43d0f81a768f5'),
    timestamp: '2025-05-08T16:14:00.000Z',
    symbol: 'BTCUSDT',
    window: '30s',
    zscore_price: 0.6663096119431902
  },
  {
    _id: ObjectId('681cd9d242f43d0f81a768f6'),
    timestamp: '2025-05-08T16:15:00.000Z',
    symbol: 'BTCUSDT',
    window: '30s',
    zscore_price: -0.09695837336895223
  },
  {
    _id: ObjectId('681cd9f842f43d0f81a768fb'),
    timestamp: '2025-05-08T16:16:30.000Z',
    symbol: 'BTCUSDT',
    window: '30s',
    zscore_price: -0.08682483802817548
  },
  {
    _id: ObjectId('681cda0f42f43d0f81a768fd'),
    timestamp: '2025-05-08T16:16:00.000Z',
    symbol: 'BTCUSDT',
    window: '30s',
    zscore_price: 0.7660935680723318
  }
]
btc> 
```



```
btc> db["btc-price-zscore-1m"].find().limit(5).pretty()
[
  {
    _id: ObjectId('681cd9d242f43d0f81a768f8'),
    timestamp: '2025-05-08T16:14:00.000Z',
    symbol: 'BTCUSDT',
    window: '1m',
    zscore_price: 0.9566315768857432
  },
  {
    _id: ObjectId('681cd9d242f43d0f81a768f9'),
    timestamp: '2025-05-08T16:15:00.000Z',
    symbol: 'BTCUSDT',
    window: '1m',
    zscore_price: 0.07198701918221716
  },
  {
    _id: ObjectId('681cda0f42f43d0f81a768fe'),
    timestamp: '2025-05-08T16:16:00.000Z',
    symbol: 'BTCUSDT',
    window: '1m',
    zscore_price: 1.2158434371160227
  },
  {
    _id: ObjectId('681cda5842f43d0f81a76902'),
    timestamp: '2025-05-08T16:18:00.000Z',
    symbol: 'BTCUSDT',
    window: '1m',
    zscore_price: -0.7792621327024425
  },
  {
    _id: ObjectId('681cda6842f43d0f81a76907'),
    timestamp: '2025-05-08T16:17:00.000Z',
    symbol: 'BTCUSDT',
    window: '1m',
    zscore_price: 1.4528382323610312
  }
]
```



```
btc> db["btc-price-zscore-5m"].find().limit(5).pretty()
[
  {
    _id: ObjectId('681cd9d242f43d0f81a768f3'),
    timestamp: '2025-05-08T16:15:00.000Z',
    symbol: 'BTCUSDT',
    window: '5m',
    zscore_price: 0.5278383788750733
  },
  {
    _id: ObjectId('681cdad642f43d0f81a76911'),
    timestamp: '2025-05-08T16:20:00.000Z',
    symbol: 'BTCUSDT',
    window: '5m',
    zscore_price: 0.44877086432962715
  },
  {
    _id: ObjectId('681cdbe642f43d0f81a76922'),
    timestamp: '2025-05-08T16:25:00.000Z',
    symbol: 'BTCUSDT',
    window: '5m',
    zscore_price: -0.7372902182842617
  },
  {
    _id: ObjectId('681cdbe6565214752645a131'),
    timestamp: '2025-05-08T16:25:00.000Z',
    symbol: 'BTCUSDT',
    window: '5m',
    zscore_price: -0.7372902182842617
  },
  {
    _id: ObjectId('681cdd9a565214752645a14f'),
    timestamp: '2025-05-08T16:30:00.000Z',
    symbol: 'BTCUSDT',
    window: '5m',
    zscore_price: -0.1314930582793813
  }
]
```



```
]
btc> db["btc-price-zscore-15m"].find().limit(5).pretty()
[
  {
    _id: ObjectId('681cd9d242f43d0f81a768f7'),
    timestamp: '2025-05-08T16:15:00.000Z',
    symbol: 'BTCUSDT',
    window: '15m',
    zscore_price: 1.1359188293788278
  },
  {
    _id: ObjectId('681cdd9a42f43d0f81a7693f'),
    timestamp: '2025-05-08T16:30:00.000Z',
    symbol: 'BTCUSDT',
    window: '15m',
    zscore_price: -1.146864739638892
  },
  {
    _id: ObjectId('681cdd9a565214752645a150'),
    timestamp: '2025-05-08T16:30:00.000Z',
    symbol: 'BTCUSDT',
    window: '15m',
    zscore_price: -1.146864739638892
  }
]
```

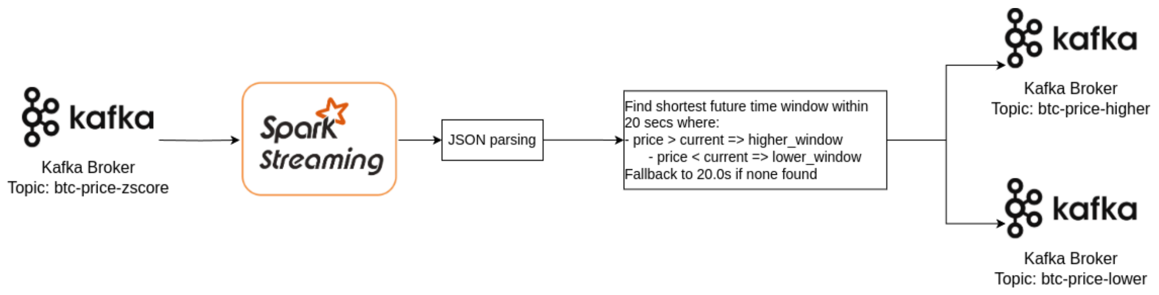
8 Bonus

Objective: For every price update from the Kafka topic btc-price, compute the shortest time window within the next 20 seconds where the price is higher or lower than the current price. This real-time detection helps in short-term trend monitoring. The results are published back to Kafka topics.

Configuration:

- Kafka Configuration:
  - Input topic: btc-price
  - Output topics: btc-price-higher, btc-price-lower

Program Architect:


kafka
Kafka Broker
Topic: btc-price-zscore
Spark Streaming
JSON parsing
Find shortest future time window within 20 secs where:
- price > current => higher_window
- price < current => lower_window
Fallback to 20.0s if none found
kafka
Kafka Broker
Topic: btc-price-higher
kafka
Kafka Broker
Topic: btc-price-lower

Implementation:

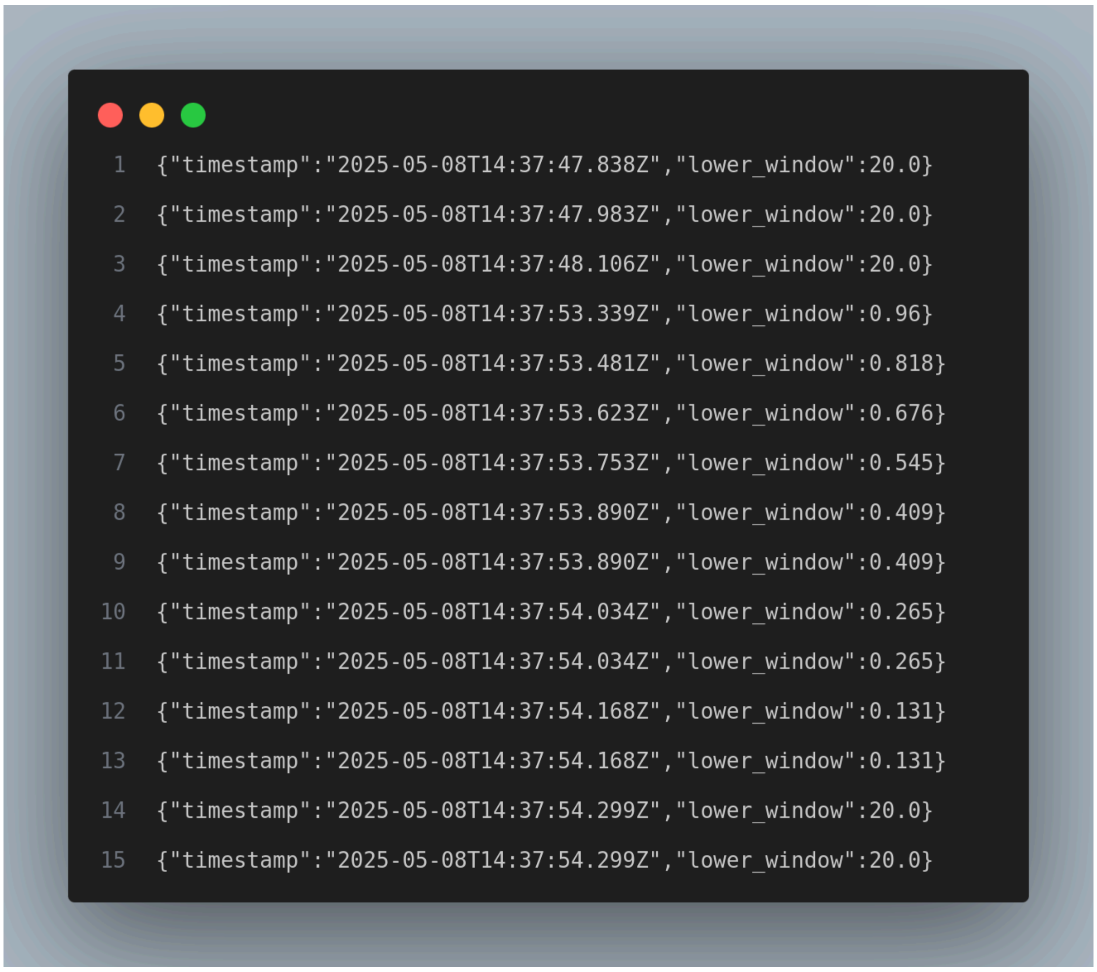
- Framework: PySpark Structured Streaming
- Libraries Used:
  - `pyspark.sql.SparkSession`: to initialize and configure the Spark application.
  - `pyspark.sql.functions`: for aggregation functions such as `avg`, `stddev`, and JSON formatting.
  - `pyspark.sql.types`: for schema definitions.
- Kafka Integration:
  - Input stream from Kafka topic `btc-price` using Spark's Kafka connector.
  - The message payloads are deserialized from JSON format into structured records using a defined schema (symbol, price, timestamp).
- Stateful Logic:
  - Maintains a temporary in-memory state of price updates.

  - For each batch, the code finds the shortest future window (≤ 20s) where:
  - A higher price than the current one appears.
  - A lower price than the current one appears
  - If no such price is found in 20 seconds, it's marked with a default 20.0 seconds window

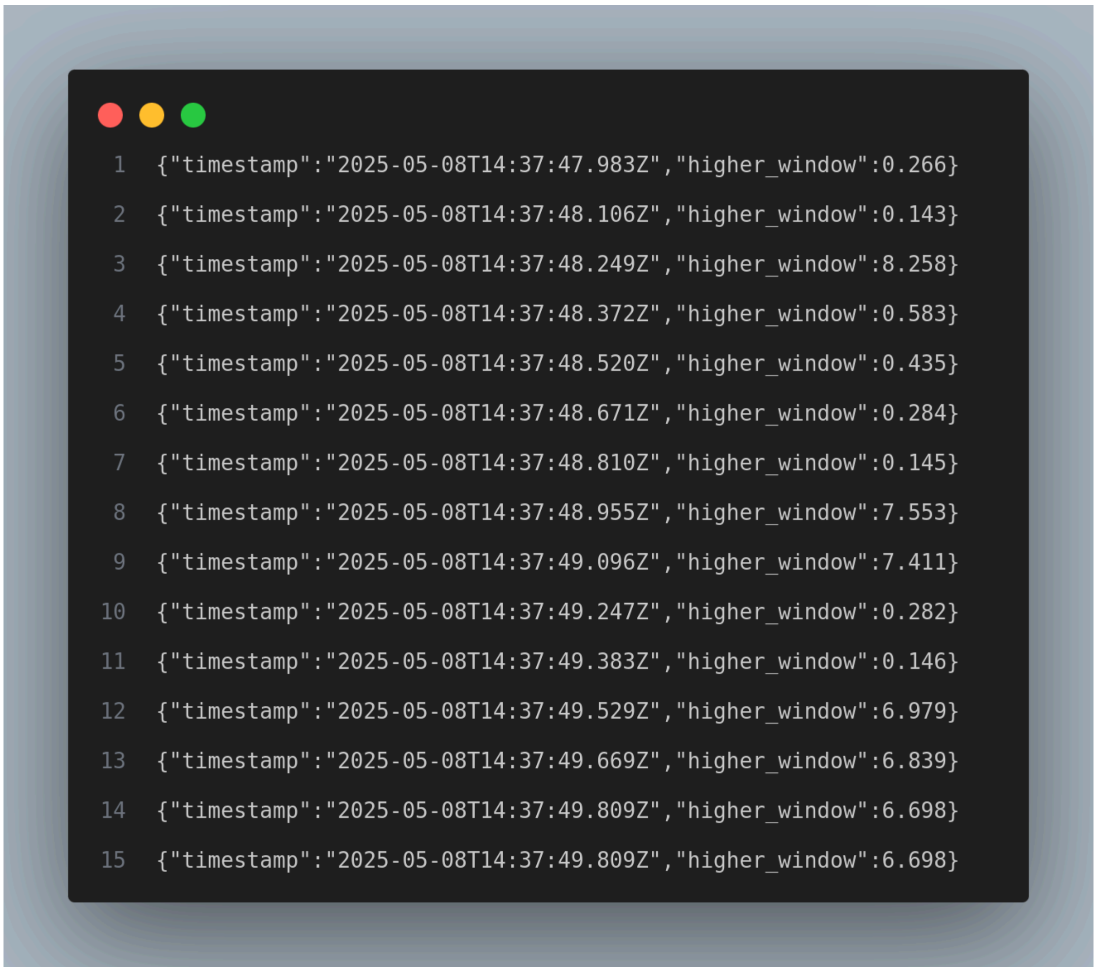
- Batch Processing:
  - Uses foreachBatch for micro-batch processing to apply custom logic.
  - Collects all incoming records in the batch, combines them with retained state, and sorts them.
  - Computes future windows by iterating through timestamps.
  - Dynamically filters results into two separate DataFrames for higher_window and lower_window.

- Kafka Output:
  - Results are formatted to JSON and published to likewise Kafka topics: btc-price-higher and btc-price-lower.
  - Output mode is append, and checkpoint is enabled for fault-tolerance.

Result:
For each BTC price update:
- The system outputs the minimal future time (up to 20s) when the price increases or decreases.
- Published as JSON messages to Kafka topics btc-price-higher and btc-price-lower:



```
{"timestamp":"2025-05-08T14:37:47.838Z","lower_window":20.0}
{"timestamp":"2025-05-08T14:37:47.983Z","lower_window":20.0}
{"timestamp":"2025-05-08T14:37:48.106Z","lower_window":20.0}
{"timestamp":"2025-05-08T14:37:53.339Z","lower_window":0.96}
{"timestamp":"2025-05-08T14:37:53.481Z","lower_window":0.818}
{"timestamp":"2025-05-08T14:37:53.623Z","lower_window":0.676}
{"timestamp":"2025-05-08T14:37:53.753Z","lower_window":0.545}
{"timestamp":"2025-05-08T14:37:53.890Z","lower_window":0.409}
{"timestamp":"2025-05-08T14:37:53.890Z","lower_window":0.409}
{"timestamp":"2025-05-08T14:37:54.034Z","lower_window":0.265}
{"timestamp":"2025-05-08T14:37:54.034Z","lower_window":0.265}
{"timestamp":"2025-05-08T14:37:54.168Z","lower_window":0.131}
{"timestamp":"2025-05-08T14:37:54.168Z","lower_window":0.131}
{"timestamp":"2025-05-08T14:37:54.299Z","lower_window":20.0}
{"timestamp":"2025-05-08T14:37:54.299Z","lower_window":20.0}
```



```
{"timestamp":"2025-05-08T14:37:47.983Z","higher_window":0.266}
{"timestamp":"2025-05-08T14:37:48.106Z","higher_window":0.143}
{"timestamp":"2025-05-08T14:37:48.249Z","higher_window":8.258}
{"timestamp":"2025-05-08T14:37:48.372Z","higher_window":0.583}
{"timestamp":"2025-05-08T14:37:48.520Z","higher_window":0.435}
{"timestamp":"2025-05-08T14:37:48.671Z","higher_window":0.284}
{"timestamp":"2025-05-08T14:37:48.810Z","higher_window":0.145}
{"timestamp":"2025-05-08T14:37:48.955Z","higher_window":7.553}
{"timestamp":"2025-05-08T14:37:49.096Z","higher_window":7.411}
{"timestamp":"2025-05-08T14:37:49.247Z","higher_window":0.282}
{"timestamp":"2025-05-08T14:37:49.383Z","higher_window":0.146}
{"timestamp":"2025-05-08T14:37:49.529Z","higher_window":6.979}
{"timestamp":"2025-05-08T14:37:49.669Z","higher_window":6.839}
{"timestamp":"2025-05-08T14:37:49.809Z","higher_window":6.698}
{"timestamp":"2025-05-08T14:37:49.809Z","higher_window":6.698}
```

9 References

- Binance API Docs
- Apache Spark Structured Streaming
- Kafka Python
- MongoDB Spark Connector