VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
UNIVERSITY OF SCIENCE

NHẬP MÔN DỮ LIỆU LỚN

INTRODUCTION TO BIG DATA ANALYSIS

Lecturers

Dr. Nguyễn Ngọc Thảo | nnthao@fit.hcmus.edu.vn
Dr. Lê Ngọc Thành | lnthanh@fit.hcmus.edu.vn

Lab instructor

Huỳnh Lâm Hải Đăng | hlhdang@fit.hcmus.edu.vn

Lab 04: Spark Streaming[1]

1. Statements

In this Lab, you and your team is going to implement a pipeline of Extract-Transform-Load that perform streamed analysis on time-series data of cryptocurrencies' prices, in particular the price of symbol BTCUSDT from Binance trading platform. Similar to the Lab 1, this lab is also explorative of the frameworks for processing big data, thus you may encounter many unseen issues during the setup as well as the processing steps. You are recommended to provide detailed description of these issues and how to solve them in your report.

Disclaimer:

This exercise is for educational and technical demonstration purposes only. It is hereby declared that no encouragement, endorsement, or recommendation for engaging in cryptocurrency trading, investment, or speculative activities is provided.

1.1. Extract

In the extract stage, you will utilize Binance's APIs to crawl the time-series data about the symbol using **any** programming language that is capable of doing the task, meaning that you are NOT restricted to Python or Scala for this stage. This data is then published through Kafka to the transform stage.

Requirements: Implement a Kafka producer that

- Fetches the price of the symbol from Binance API which contains a floating-point value representing the symbol's price.
- Upon receiving a response from the API, checks if the received JSON conforms the following output format, based on the documentation of Binance:

```
{
  "symbol": <a string>,
  "price": <a floating-point value>
}
```

- Inserts event-time information:

[1] Credit: this labwork's statement and problems are also co-authored by Vũ Công Thành.

    - Add another field to the above JSON that denotes the timestamp associated with this response, in other words, this is the timestamp that your crawler received this JSON response.
    - Refer to *ISO8601 standards* for detailed format of the timestamp, you are recommended to use your language's time-related libraries for processing these timestamps. The precise decimal of sub-second information is determined according to the running frequency that your team has chosen, for example with a frequency of once per 100 miliseconds then your sub-second digits should be rounded down to *s.s00* (.e.g 30.475 to 30.400) form, or *s.s50-s.s00* (.e.g 30.475 to 30.450 and 30.419 to 30.400) if the frequency is twice per 100 miliseconds instead, or *s.s20* (.e.g 19.750 to 19.740 and 20.261 to 20.260) for five times per 100 miliseconds. This precision will be your event_time in other sections of this lab so this rounding rule will help your programs map the events' times more easily.
- Runs with a frequency of at least once per 100 miliseconds.
- Push those records to a topic in Kafka named **btc-price**.
- You and your team should take screenshots of significant steps that you and the team did, then put them into your report with detailed explanation.

Binance API:

- Reference: https://developers.binance.com/docs/binance-spot-api-docs/rest-api/market-data-endpoints
- API: api.binance.com/api/v3/ticker/price?symbol=BTCUSDT

1.2. Transform

The transform stage has two steps that involves Kafka publications and subscriptions. Specifically, the first step requires the calculation of moving average and moving standard deviation within specified sliding windows while the second one computes the Z-scores of latest price against those windows' moving average and standard deviation.

Allowed programming language(s): *Java*, *Python*, and *Scala*. NOTE: your implementations should handle late data with a tolerance of up to **10 seconds** late time.

Requirements:

- Implement a program using Spark Structure Streaming, also including Spark SQL, to:
    - Subscribe to the **btc-price** topic from Kafka of the extract stage.

- Use event-time processing to group the listened messages into sliding windows of the following lengths: *30s* (30 seconds), *1m* (1 minute or 60 seconds), *5m* (5 minutes), *15m* (15 minutes), *30m* (30 minutes), and *1h* (1 hour or 60 minutes).
- Compute the moving averages and moving standard deviations by calculating average and standard deviation of prices per window.
- You should also handle edge cases with your own rules and definitions, then output the results in the following format:

```
{
  "timestamp": <ISO8601 UTC timestamp>,
  "symbol": <a string>,
  "windows": [
    {
      "window": <a string among 30s, 1m, 5m, 15m, 30m, 1h>,
      "avg_price": <a floating-point value>,
      "std_price": <a floating-point value>
    },
    ... # Repeat until avg and std of all windows are provided
  ]
}
```

- Publish the results to another Kafka topic called **btc-price-moving** with append mode.

- Implement a program using Spark Structure Streaming, also including Spark SQL, to:
  - Listen to both of the following Kafka topics: **btc-price** and **btc-price-moving**.
  - With one record read from **btc-price** and another one from **btc-price-moving** that share the same `timestamp` information, computes the Z-score of the price with respect to each sliding window given in the moving statistics record's information.
  - After handling the edge cases (if any, with your own rules and definitions), output the results in the following format:

```
{
  "timestamp": <ISO8601 UTC timestamp>,
  "symbol": <a string>,
  "zscores": [
    {
      "window": <a string among 30s, 1m, 5m, 15m, 30m, 1h>,
      "zscore_price": <a floating-point value>
    },
    ... # Repeat until Z-scores of all windows are provided
```

```
    ]
}
```

  - Publish this result to a new Kafka topic called **btc-price-zscore** by append mode similar to prior tasks.
- You and your team should take screenshots of significant steps that you and the team did, then put them into your report with detailed explanation.

References:

- Z-score: also known as *standard score*, refer to Standard score - Wikipedia for more details.

Hint: these are *just* hints to ease your work, they are by NO means a restriction on what you can do and what you cannot, thus it is up to you and your team to consider making use of these hints or to find your team's own way towards the accomplishment.

- (1) As your event-time has been rounded down by earlier extract stage, you can possibly apply classic *sliding* window operation (no tumbling) of Apache Spark's Structured Stream to computes the required statistics.
- (2) You should use Apache Spark's stream-stream inner-join operations with watermarking to join the streams used in second transform program.

1.3. Load

In the load stage, you and your team will use Spark Structured Stream and MongoDB Spark Connector to store the calculated data as a collection under the streaming mode.

Allowed programming language(s): *Java*, *Python*, and *Scala*. NOTE: your implementations should handle late data with a tolerance of up to **10 seconds** late time.

Requirements:

- Setup a MongoDB for persistently storing the computed data, you are free to choose where to install the database management system as well as the method of installation.
- Subscribe to the **btc-price-zscore** Kafka topic from the transform stage and create a Spark Structured Stream from it.
- Write this stream to MongoDB collections named **btc-price-zscore-<window>** where **<window>** encodes the interval associated with the sliding window (*30s*, *1m*,

5m, *15m*, *30m*, *1h*). You are free to define the schema of these collections but be sure to denotes it in the final report.

- You and your team should take screenshots of significant steps that you and the team did, then put them into your report with detailed explanation.

References:

- MongoDB Documentation: https://www.mongodb.com/docs/manual/introduction/

1.4. Bonus

This is a bonus section to the transform stage where shortest windows of negative outcome is found for each price record published in the Kafka topic **btc-price**. There would be 2 such windows: one for the lower prices and another for the higher ones.

Allowed programming language(s): *Java*, *Python*, and *Scala*. NOTE: your implementations should handle late data with a tolerance of up to **10 seconds** late time.

Requirements:

- Implement a program using Spark Structure Streaming, also including Spark SQL, to:
  - Listen to the Kafka topic **btc-price** published in the extract stage.
  - For each price record *p* received from that topic with an event-time timestamp *t*, you will need to find at most two records in the 20-second interval after it, .i.e $(t, t + 20]$, where one is the first encountered message with price higher than that of *p* and the other is the first with price lower than that.
  - Calculate the time difference between each of the found records with *t*, in floating-point seconds, then publish the results to Kafka topics of **btc-price-higher** and **btc-price-lower**, respectively. For the case where no window of higher (or lower) price is found within this 20-second interval, the program must publish a placeholder record filling the field of length with 20.0.
  - The published JSON should have the following format:

```
{
  "timestamp": <ISO8601 UTC timestamp>,
  "<higher/lower>_window": <a floating-point value>
}
```

  - The Kafka topics' publications are in append mode.
- You and your team should take screenshots of significant steps that you and the team did, then put them into your report with detailed explanation.

Hint: You may need to employ some form of stateful operations for this part of the lab.