

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN
NHẬP MÔN DỮ LIỆU LỚN



BÁO CÁO BÀI TẬP MÔN HỌC
NỘI DUNG: LAB04 - SPARK STREAMING

Giảng viên hướng dẫn

:Nguyễn Ngọc Thảo

Lớp

:CQ2022/21

Sinh viên thực hiện

:Trương Tiến Anh- 22120017

Đoàn Minh Cường- 22120043

Đinh Viết Lợi-22120188

Nguyễn Trần Lợi-22120190

Hồ Chí Minh, ngày 15 tháng 6 năm 2025

MỤC LỤC

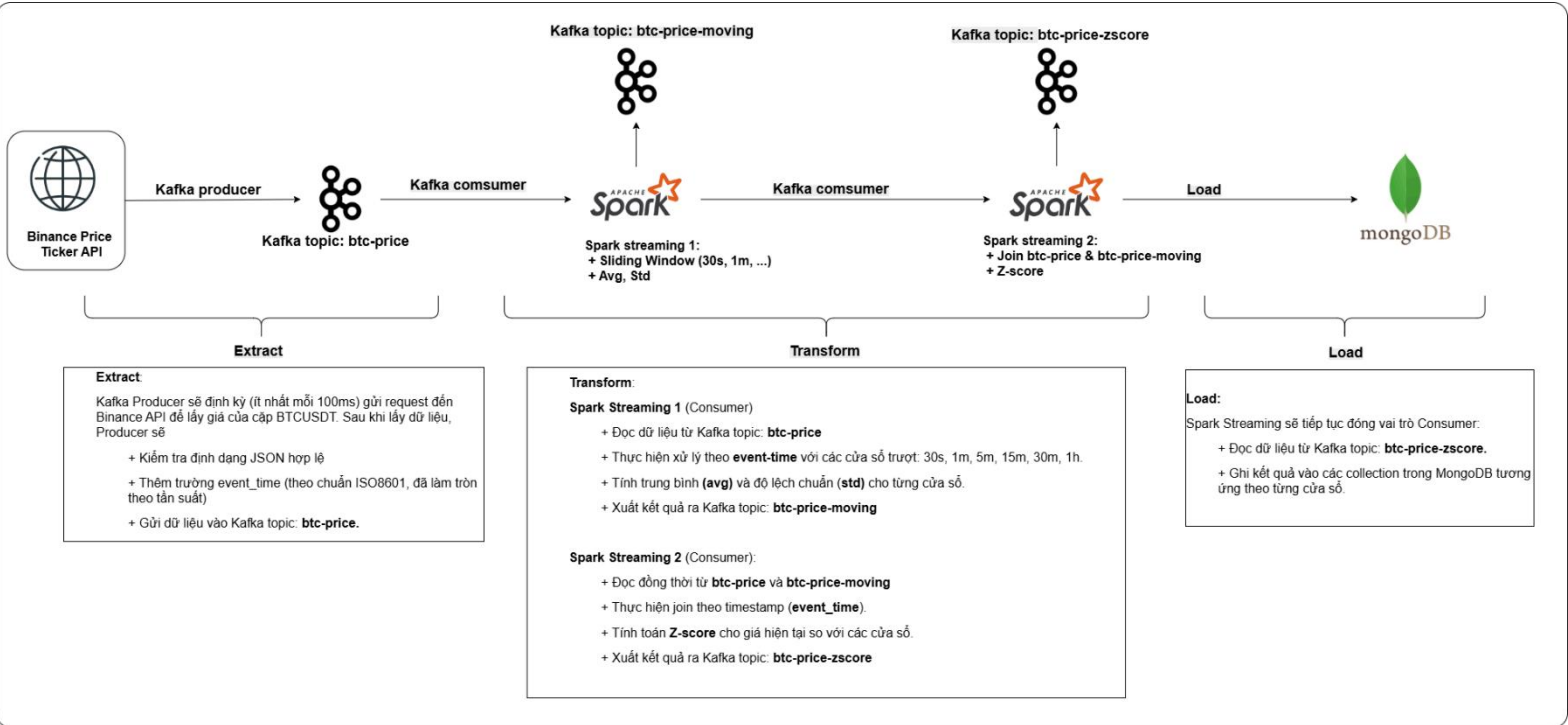
| | |
|---|-----------|
| MỤC LỤC..... | 1 |
| PHẦN 1: BÁO CÁO BÀI TẬP | 2 |
| I. Cấu trúc bài làm | 2 |
| II. Extract Stage..... | 3 |
| 1. Ý tưởng giải quyết..... | 3 |
| 2. Chi tiết thực thi..... | 3 |
| III. Transform Moving Stage..... | 5 |
| 1. Ý tưởng giải quyết..... | 5 |
| 2. Chi tiết thực thi..... | 5 |
| IV. Transform Z-score Stage | 9 |
| 1. Ý tưởng giải quyết..... | 9 |
| 2. Chi tiết thực thi..... | 9 |
| V. Load Stage..... | 12 |
| 1. Ý tưởng giải quyết..... | 12 |
| 2. Chi tiết thực thi..... | 12 |
| VI. Bonus Stage | 15 |
| 1. Ý tưởng giải quyết..... | 15 |
| 2. Chi tiết thực thi..... | 16 |
| PHẦN 2: BÁO CÁO NHÓM..... | 18 |

PHẦN 1: BÁO CÁO BÀI TẬP

I. Cấu trúc bài làm

- **src/Extract/**
 - Thực hiện thu thập dữ liệu giá BTCUSDT từ Binance API.
 - Thêm trường timestamp (event_time) vào mỗi bản ghi.
 - Gửi dữ liệu đã chuẩn hoá lên Kafka topic btc-price với tần suất $\geq 100\text{ms}$.
 - Bao gồm:
 - CQ05.py: file thực thi thu thập dữ liệu
- **src/Transform/**
 - Tính toán các thống kê trượt (moving statistics) theo sliding window:
 - Các khung thời gian: 30s, 1m, 5m, 15m, 30m, 1h.
 - Tính avg_price và std_price cho từng khung.
 - Gửi kết quả sang Kafka topic btc-price-moving.
 - Tính toán Z-score từ giá gốc và thống kê vừa tính:
 - Join hai topic btc-price và btc-price-moving bằng event_time.
 - Gửi kết quả sang Kafka topic btc-price-zscore.
 - Bao gồm:
 - CQ05_moving.py: chương trình tính moving avg/std.
 - CQ05_zscore.py: chương trình tính z-score từ kết quả thống kê.
- **src/Load/**
 - Đọc dữ liệu từ Kafka topic btc-price-zscore.
 - Ghi kết quả theo từng khung thời gian vào MongoDB collections tên dạng: btc-price-zscore-<window>.
 - Xử lý dữ liệu đến trễ với watermark 10s.
 - Bao gồm:
 - CQ05.py: chương trình thực thi giai đoạn Load
- **src/Bonus/**
 - Đọc dữ liệu từ Kafka topic btc-price.
 - Với mỗi bản ghi tại thời điểm t, tìm trong khoảng (t, t+20]:
 - Giá cao hơn \rightarrow ghi thời gian chênh lệch vào Kafka topic btc-price-higher.
 - Giá thấp hơn \rightarrow ghi vào Kafka topic btc-price-lower.
 - Nếu không có, ghi "<higher/lower>_window": 20.0.
 - Kết quả ở định dạng JSON có timestamp + độ dài cửa sổ.
 - Bao gồm:
 - CQ05.py: file thực thi tính thời điểm tăng/ giảm của một timestamp.
 - Thư mục btc_state_buffer: Lưu lại các bản ghi giá BTC gần đây để dùng cho batch sau.
 - Thư mục main_checkpoint: Lưu lại thông tin tiến độ đọc offset từ Kafka và các metadata khác của streaming query.

• Pipeline chương trình:



II. Extract Stage

1. Ý tưởng giải quyết

- Làm tròn thời gian đến mức độ chính xác là 100 ms và trả về thời gian đã làm tròn dưới dạng chuỗi ISO8601 với múi giờ UTC (+00.00).
- Khi vào quá trình fetch, chương trình sẽ bắt đầu đo thời gian và gọi API. Sau đó xác thực dữ liệu nhận được có định dạng chuẩn không rồi gửi đến topic 'btc-price'.
- Để thời gian đảm bảo liên tục và đều đặn, nhóm sử dụng chạy song song đa luồng. Với mỗi 100 ms, chương trình sẽ chạy giai đoạn fetch riêng biệt.

2. Chi tiết thực thi

- Chạy Zookeeper với dòng lệnh bên dưới:

```
~/kafka/kafka_2.12-3.5.2$ bin/zookeeper-server-start.sh config/zookeeper.properties
```

- Chạy Kafka với dòng lệnh bên dưới:

```
~/kafka/kafka_2.12-3.5.2$ bin/kafka-server-start.sh config/server.properties
```

- Tạo topic 'btc-price':

```
hadoop@NTL:~/kafka/kafka_2.12-3.5.2$ bin/kafka-topics.sh --create --topic btc-price \
--bootstrap-server localhost:9092 \
--partitions 1 \
--replication-factor 1
```

- Khởi động Kafka với port 9092 và tham số *value_serializer* định nghĩa cách dữ liệu (giá trị của bản ghi) được tuần tự hóa (serialized) trước khi gửi đến Kafka.

```
# Khởi tạo Kafka Producer
producer = KafkaProducer(
    bootstrap_servers=['localhost:9092'],
    value_serializer=lambda v: json.dumps(v).encode('utf-8')
)
```

- Hàm làm tròn thời gian với mức độ chính xác là 100 ms. '*timestamp_ms // precision_ms*' thực hiện phép chia nguyên để làm tròn xuống thời gian đến bội số gần nhất của *precision_ms*. Nhân lại với *precision_ms* để lấy thời gian đã làm tròn, loại bỏ các mili-giây dư thừa nhỏ hơn 100ms. Sau đó, chuyển về mili giây và lấy timezone là UTC.

```
def round_event_time(now:datetime, precision_ms=100):
    timestamp_ms = int(now.timestamp() * 1000)
    rounded_ms = (timestamp_ms // precision_ms) * precision_ms
    rounded_dt = datetime.fromtimestamp(rounded_ms / 1000, tz=timezone.utc)
    return rounded_dt.isoformat(timespec='milliseconds')
```

- Hàm fetch và gửi dữ liệu đến topic 'btc-price'. Gọi API và kiểm tra nếu phản hồi thành công thì sẽ tiến hành thêm "event_time" vào dữ liệu và gửi đến topic 'btc-price'.

```
def fetch_and_send(event_time):
    try:
        response = requests.get('https://api.binance.com/api/v3/ticker/price?symbol=BTCUSDT', timeout=2)
        if response.status_code == 200:
            data = response.json()
            if "symbol" in data and "price" in data:
                data["event_time"] = event_time
                data["price"] = float(data["price"])
                producer.send("btc-price", value=data)
                print("Sent:", data)
    except Exception as e:
        print(f"Lỗi: {e}")
```

- Hàm main. Đặt khoảng thời gian giữa các lần thu thập dữ liệu là 0.1 giây (100ms). Tối đa 10 luồng xử lý song song. Lưu thời điểm bắt đầu để tính toán chính xác thời gian cần ngủ ở mỗi vòng lặp, đảm bảo tần suất cố định. Lấy thời gian hiện tại để làm tròn và chuẩn hóa timestamp sau đó đưa hàm fetch và gửi dữ liệu vào hàng đợi luồng. Tính toán sleep_time dựa trên target_time, đảm bảo vòng lặp duy trì tốc độ đều đặn 100ms/lần. Nếu thời gian còn lại trước lần kế tiếp là dương thì nghỉ đúng thời gian cần thiết.

```
def main():
    interval = 0.1 # 100ms
    max_workers = 10

    executor = ThreadPoolExecutor(max_workers=max_workers)

    # Ghi nhận thời điểm bắt đầu
    target_time = time.time()

    try:
        while True:
            now = datetime.now()
            event_time = round_event_time(now)
            executor.submit(fetch_and_send, event_time)

            # Tính thời gian tiếp theo cần chờ
            target_time += interval
            sleep_time = target_time - time.time()
            if sleep_time > 0:
                time.sleep(sleep_time)
```

- Chạy chương trình.

```
hadoop@NTL:~/Extract$ python3 CQ05.py
Sent: {'symbol': 'BTCUSDT', 'price': 105471.27, 'event_time': '2025-06-15T16:40:48.400+00:00'}
Sent: {'symbol': 'BTCUSDT', 'price': 105471.27, 'event_time': '2025-06-15T16:40:48.500+00:00'}
Sent: {'symbol': 'BTCUSDT', 'price': 105471.27, 'event_time': '2025-06-15T16:40:48.600+00:00'}
Sent: {'symbol': 'BTCUSDT', 'price': 105471.26, 'event_time': '2025-06-15T16:40:48.700+00:00'}
Sent: {'symbol': 'BTCUSDT', 'price': 105471.26, 'event_time': '2025-06-15T16:40:48.800+00:00'}
Sent: {'symbol': 'BTCUSDT', 'price': 105471.26, 'event_time': '2025-06-15T16:40:48.900+00:00'}
Sent: {'symbol': 'BTCUSDT', 'price': 105471.26, 'event_time': '2025-06-15T16:40:49.000+00:00'}
Sent: {'symbol': 'BTCUSDT', 'price': 105479.8, 'event_time': '2025-06-15T16:40:49.100+00:00'}
```

III. Transform Moving Stage

1. Ý tưởng giải quyết

- Lắng nghe topic Kafka btc-price, đây là nơi chứa dữ liệu đã được chuẩn hóa ở giai đoạn Extract.
- Dữ liệu được xử lý theo event-time và phân chia thành các cửa sổ trượt với các kích thước khác nhau: 30s, 1m, 5m, 15m, 30m, 1h.
- Với mỗi cửa sổ trên ta sẽ tính giá trị trung bình (avg_price) và độ lệch chuẩn (std_price)
- Dữ liệu đầu ra sẽ được ghi vào topic Kafka btc-price-moving. Và cần phải xử lý độ trễ khoảng 10s.

2. Chi tiết thực thi

- Tạo topic 'btc-price-moving'.

```
hadoop@NTL:~/kafka/kafka_2.12-3.5.2$ bin/kafka-topics.sh --create --topic btc-price-moving \
--bootstrap-server localhost:9092 \
--partitions 1 \
--replication-factor 1
```

- Config thông số Kafka: định nghĩa topic Kafka nguồn và đích, địa chỉ bootstrap servers.

```
CONFIG = {
    "KAFKA_TOPIC_PRICE": "btc-price",
    "KAFKA_TOPIC_MOVING": "btc-price-moving",
    "KAFKA_BOOTSTRAP_SERVERS": "localhost:9092"
}
```

- Khởi tạo SparkSession: **splitThreshold** giúp tăng giới hạn chia nhỏ codegen khi tối ưu hóa query (xử lý codegen cho batch lớn). **timeZone="UTC"** quan trọng khi xử lý dữ liệu thời gian thực (giúp đồng bộ thời gian chuẩn UTC).

```
def init_spark():
    spark = SparkSession.builder \
        .appName("BTC Price Transform") \
        .config("spark.sql.codegen.methods.splitThreshold", "100000") \
        .config("spark.sql.session.timeZone", "UTC") \
        .getOrCreate()
    return spark
```

- Đọc dữ liệu từ Kafka (streaming source): Đọc dữ liệu streaming từ Kafka topic "btc-price". **startingOffsets = earliest**: đọc từ đầu topic (tốt cho testing) nhưng thực tế thì nên chạy lastest để không đọc lại dữ liệu cũ.

```
def read_topic_kafka(spark):
    kafka_topic = CONFIG["KAFKA_TOPIC_PRICE"]
    kafka_bootstrap_servers = CONFIG["KAFKA_BOOTSTRAP_SERVERS"]

    df = spark.readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", kafka_bootstrap_servers) \
        .option("subscribe", kafka_topic) \
        .option("startingOffsets", "earliest") \
        .load()
    return df
```

- Parse dữ liệu JSON + watermark: Parse chuỗi JSON từ Kafka. Chuyển event_time thành timestamp Spark. **Watermark 10 giây**: cho phép trễ tối đa 10s khi thực hiện window aggregation.

```
def parse_and_watermark(df):
    raw_schema = StructType([
        StructField("symbol", StringType(), True),
        StructField("price", DoubleType(), True),
        StructField("event_time", StringType(), True)
    ])
    df_parsed = (
        df.selectExpr("CAST(value AS STRING)")
        .select(from_json(col("value"), raw_schema).alias("data"))
        .select("data.*")
        .withColumn("timestamp", to_timestamp(col("event_time"), "yyyy-MM-dd'T'HH:mm:ss.SSSXXX"))
        .filter(col("price").isNotNull() & (col("price") > 0))
        .withWatermark("timestamp", "10 seconds")
    )
    return df_parsed
```

- Tính toán các thống kê trên nhiều window: Tính trung bình (avg) và độ lệch chuẩn (stddev) cho các cửa sổ 30s và 1 phút. Dùng sliding window (cửa sổ trượt) mỗi 0.1 giây → cho kết quả gần như real-time.

```
def compute_all_stats(df):
    WINDOW_SPECS = [
        ("30s", "30 seconds"),
        ("1m", "1 minute"),
        ("5m", "5 minutes"),
        ("15m", "15 minutes"),
        ("30m", "30 minutes"),
        ("1h", "1 hour"),
    ]
    SLIDE = "0.1 seconds"
    stats_dfs = []
    for abbr, duration in WINDOW_SPECS:
        stats = df.groupBy(window(col("timestamp"), duration, SLIDE), col("symbol"))
        stats_1 = stats.agg(
            avg("price").alias(f"avg_{abbr}"),
            stddev("price").alias(f"std_{abbr}")
        )
        stats_2 = stats_1.select(
            col("symbol"),
            col("window.end").alias("timestamp"),
            col(f"avg_{abbr}"),
            col(f"std_{abbr}")
        )
        stats_dfs.append((abbr, stats_2))
    return stats_dfs
```

- Join nhiều kết quả window: Join các bảng thống kê lại với nhau theo symbol và timestamp đồng bộ. Gộp kết quả thành 1 bản ghi duy nhất cho mỗi timestamp.


```
def join_and_format(stats_dfs):
    # Tách tuple (abbr, df)
    abbrs = [abbr for abbr, df in stats_dfs]
    dfs = [df for abbr, df in stats_dfs]

    joined = dfs[0]
    for i in range(1, len(dfs)):
        joined = joined.join(dfs[i], on=["symbol", "timestamp"], how="inner")

    windows_arr = array(
        *[
            struct(
                lit(abbr).alias("window"),
                col(f"avg_{abbr}").alias("avg_price"),
                col(f"std_{abbr}").alias("std_price"),
            )
            for abbr in abbrs
        ]
    )
    result = joined.select(
        col("timestamp"),
        col("symbol"),
        windows_arr.alias("windows")
    )
    return result
```

- Format dữ liệu về đúng dạng và ghi vào kafka đích.

```
def format_output(df):
    return df.select(
        col("symbol").cast("string").alias("key"),
        to_json(struct("timestamp", "symbol", "windows")).alias("value"),
    )

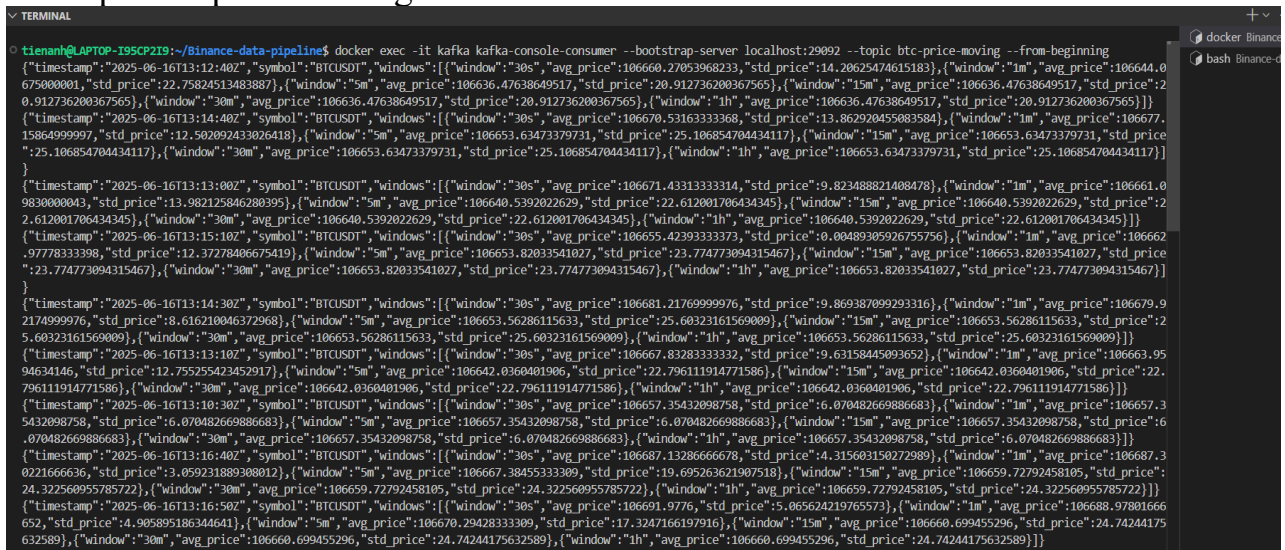
def write_to_kafka(df):
    kafka_topic = CONFIG["KAFKA_TOPIC_MOVING"]
    kafka_bootstrap_servers = CONFIG["KAFKA_BOOTSTRAP_SERVERS"]

    query = df.writeStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", kafka_bootstrap_servers) \
        .option("topic", kafka_topic) \
        .option("checkpointLocation", "tmp/checkpoint_price_moving") \
        .outputMode("append") \
        .start()
    return query
```

- Gọi main để chạy các function theo thứ tự.

```
if __name__ == "__main__":
    spark = init_spark()
    spark.sparkContext.setLogLevel("WARN")
    raw_kafka_df = read_topic_kafka(spark)
    parsed_df = parse_and_watermark(raw_kafka_df)
    stats_dfs = compute_all_stats(parsed_df)
    joined_df = join_and_format(stats_dfs)
    formatted_df = format_output(joined_df)
    kafka_query = write_to_kafka(formatted_df)
    kafka_query.awaitTermination()
```

- Chuyển kết quả thành JSON chứa các trường: timestamp (UTC ISO8601), symbol, và windows (mảng các avg, std theo từng cửa sổ: window, avg_price, std_price). Ghi kết quả vào topic btc-price-moving.



```

{"timestamp": "2025-06-16T13:12:40Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106660.27053968233, "std_price": 14.20625474615183}, {"window": "1m", "avg_price": 106644.0675000001, "std_price": 22.75824513483887}, {"window": "5m", "avg_price": 106636.47638649517, "std_price": 20.912736200367565}, {"window": "15m", "avg_price": 106636.47638649517, "std_price": 20.912736200367565}, {"window": "30m", "avg_price": 106636.47638649517, "std_price": 20.912736200367565}, {"window": "1h", "avg_price": 106636.47638649517, "std_price": 20.912736200367565}], {"timestamp": "2025-06-16T13:14:40Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106670.53163333368, "std_price": 13.862920455083584}, {"window": "1m", "avg_price": 106677.15864999997, "std_price": 12.582092433026418}, {"window": "5m", "avg_price": 106653.63473379731, "std_price": 25.106854704434117}, {"window": "15m", "avg_price": 106653.63473379731, "std_price": 25.106854704434117}, {"window": "30m", "avg_price": 106653.63473379731, "std_price": 25.106854704434117}, {"window": "1h", "avg_price": 106653.63473379731, "std_price": 25.106854704434117}], {"timestamp": "2025-06-16T13:13:00Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106671.43313333314, "std_price": 9.823488821408478}, {"window": "1m", "avg_price": 106661.09830000043, "std_price": 13.982125846280395}, {"window": "5m", "avg_price": 106640.5392022629, "std_price": 22.612001706434345}, {"window": "15m", "avg_price": 106640.5392022629, "std_price": 22.612001706434345}, {"window": "30m", "avg_price": 106640.5392022629, "std_price": 22.612001706434345}, {"window": "1h", "avg_price": 106640.5392022629, "std_price": 22.612001706434345}], {"timestamp": "2025-06-16T13:15:10Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106655.42393333373, "std_price": 0.00489305926755756}, {"window": "1m", "avg_price": 106662.9777833398, "std_price": 12.37278406675419}, {"window": "5m", "avg_price": 106653.82033541027, "std_price": 23.774773094315467}, {"window": "15m", "avg_price": 106653.82033541027, "std_price": 23.774773094315467}, {"window": "30m", "avg_price": 106653.82033541027, "std_price": 23.774773094315467}, {"window": "1h", "avg_price": 106653.82033541027, "std_price": 23.774773094315467}], {"timestamp": "2025-06-16T13:14:30Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106681.21769999976, "std_price": 9.869387099293316}, {"window": "1m", "avg_price": 106679.92174999976, "std_price": 8.616210046372968}, {"window": "5m", "avg_price": 106653.56286115633, "std_price": 25.60323161569009}, {"window": "15m", "avg_price": 106653.56286115633, "std_price": 25.60323161569009}, {"window": "30m", "avg_price": 106653.56286115633, "std_price": 25.60323161569009}, {"window": "1h", "avg_price": 106653.56286115633, "std_price": 25.60323161569009}], {"timestamp": "2025-06-16T13:13:10Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106667.83283333332, "std_price": 9.63158445093652}, {"window": "1m", "avg_price": 106663.9594634146, "std_price": 12.75525423452917}, {"window": "5m", "avg_price": 106642.0360401906, "std_price": 22.796111914771586}, {"window": "15m", "avg_price": 106642.0360401906, "std_price": 22.796111914771586}, {"window": "30m", "avg_price": 106642.0360401906, "std_price": 22.796111914771586}, {"window": "1h", "avg_price": 106642.0360401906, "std_price": 22.796111914771586}], {"timestamp": "2025-06-16T13:10:30Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106657.35432098758, "std_price": 6.070482669886683}, {"window": "1m", "avg_price": 106657.35432098758, "std_price": 6.070482669886683}, {"window": "5m", "avg_price": 106657.35432098758, "std_price": 6.070482669886683}, {"window": "15m", "avg_price": 106657.35432098758, "std_price": 6.070482669886683}, {"window": "30m", "avg_price": 106657.35432098758, "std_price": 6.070482669886683}, {"window": "1h", "avg_price": 106657.35432098758, "std_price": 6.070482669886683}], {"timestamp": "2025-06-16T13:16:40Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106687.13286666678, "std_price": 4.315603150272989}, {"window": "1m", "avg_price": 106687.30216666636, "std_price": 3.059231889308012}, {"window": "5m", "avg_price": 106667.38455333309, "std_price": 10.695263621907518}, {"window": "15m", "avg_price": 106659.72792458105, "std_price": 24.322560955785722}, {"window": "30m", "avg_price": 106659.72792458105, "std_price": 24.322560955785722}, {"window": "1h", "avg_price": 106659.72792458105, "std_price": 24.322560955785722}], {"timestamp": "2025-06-16T13:16:50Z", "symbol": "BTCUSD", "windows": [{"window": "30s", "avg_price": 106691.9776, "std_price": 5.065624219765573}, {"window": "1m", "avg_price": 106688.97801666652, "std_price": 4.905895186344641}, {"window": "5m", "avg_price": 106670.29428333309, "std_price": 17.3247166197916}, {"window": "15m", "avg_price": 106660.699455296, "std_price": 24.74244175632589}, {"window": "30m", "avg_price": 106660.699455296, "std_price": 24.74244175632589}, {"window": "1h", "avg_price": 106660.699455296, "std_price": 24.74244175632589}]}

```

IV. Transform Z-score Stage

1. Ý tưởng giải quyết

- Ta sẽ lắng nghe đồng thời 2 topic Kafka: **btc-price** và **btc-price-moving**.
- Thực hiện inner join hai luồng dữ liệu này dựa trên timestamp giống nhau, kết hợp với watermarking để đảm bảo đồng bộ hóa theo thời gian sự kiện.
- Với mỗi bản ghi (cùng timestamp), chương trình sẽ tính Z-score cho từng cửa sổ và sử dụng công thức:

$$Z = \frac{x - \mu}{\sigma}$$

- Kết quả đầu ra sẽ được đẩy vào topic Kafka mới tên **btc-price-zscore**
- ### 2. Chi tiết thực thi
- Cấu hình cách biến mỗi trường để dễ triển khai

```
CONFIG = {  
    "KAFKA_TOPIC_PRICE": "btc-price",  
    "KAFKA_TOPIC_MOVING": "btc-price-moving",  
    "KAFKA_TOPIC_ZSCORE": "btc-price-zscore",  
    "KAFKA_BOOTSTRAP_SERVERS": "localhost:9092"  
}
```

- Khởi tạo SparkSession: Tạo một SparkSession với cấu hình UTC để thống nhất timezone.

```
def init_spark():  
    spark = SparkSession.builder \  
        .appName("BTC Price Z-Score") \  
        .config("spark.sql.codegen.methods.splitThreshold", "10000") \  
        .config("spark.sql.session.timeZone", "UTC") \  
        .getOrCreate()  
    spark.sparkContext.setLogLevel("WARN")  
    return spark
```

- Đọc dữ liệu từ Kafka:
 - Đọc song song hai topic Kafka: btc-price: chứa thông tin giá BTC tại các thời điểm. btc-price-moving: chứa thông tin trung bình động và độ lệch chuẩn tương ứng với từng cửa sổ thời gian
 - Cả hai stream đều được xử lý dạng structured streaming từ Kafka, và đều chuyển event_time hoặc timestamp thành kiểu TimestampType để xử lý theo event-time

```
def read_kafka_topic(spark, topic):  
    df = spark.readStream \  
        .format("kafka") \  
        .option("kafka.bootstrap.servers", CONFIG["KAFKA_BOOTSTRAP_SERVERS"]) \  
        .option("subscribe", topic) \  
        .option("startingOffsets", "earliest") \  
        .load()  
    return df  
  
spark = init_spark()  
  
price_raw_df = read_kafka_topic(spark, CONFIG["KAFKA_TOPIC_PRICE"])  
moving_raw_df = read_kafka_topic(spark, CONFIG["KAFKA_TOPIC_MOVING"])
```

- Xử lý stream data từ topic Kafka btc-price:
 - Parse JSON từ topic btc-price thành DataFrame có các cột: symbol, price, event_time.
 - Dùng to_timestamp để chuyển đổi thời gian, thêm watermark 10s để xử lý dữ liệu đến trễ

```
def parse_price_stream(df):
    schema = StructType([
        StructField("symbol", StringType(), True),
        StructField("price", DoubleType(), True),
        StructField("event_time", StringType(), True)
    ])
    df_parsed = (
        df.selectExpr("CAST(value AS STRING)")
        .select(from_json(col("value"), schema).alias("data"))
        .select("data.*")
        .withColumn("timestamp", to_timestamp(col("event_time"), "yyyy-MM-dd'T'HH:mm:ss.SSSXXX"))
        .filter(col("price").isNotNull() & (col("price") > 0))
        .withWatermark("timestamp", "10 seconds")
    )
    return df_parsed
```

- Xử lý stream data từ topic Kafka btc-price-moving:
 - Parse JSON từ topic btc-price-moving, định nghĩa schema ArrayType(window_info).
 - Áp dụng withWatermark("timestamp", "10 seconds") để hỗ trợ inner join theo event-time.

```
def parse_moving_stream(df):
    window_schema = StructType([
        StructField("window", StringType(), True),
        StructField("avg_price", DoubleType(), True),
        StructField("std_price", DoubleType(), True)
    ])
    schema = StructType([
        StructField("timestamp", StringType(), True),
        StructField("symbol", StringType(), True),
        StructField("windows", ArrayType(window_schema), True)
    ])
    df_parsed = (
        df.selectExpr("CAST(value AS STRING)")
        .select(from_json(col("value"), schema).alias("data"))
        .select("data.*")
        .withColumn("timestamp", to_timestamp(col("timestamp"), "yyyy-MM-dd'T'HH:mm:ss.SSSXXX"))
        .filter(col("windows").isNotNull())
        .withWatermark("timestamp", "10 seconds")
    )
    return df_parsed
```

- Tính toán Z-score:
 - Join hai stream (price_df và moving_df) dựa trên timestamp và symbol.
 - Sử dụng explode(windows) để phân rã mảng thống kê theo từng cửa sổ.
 - Với mỗi cửa sổ, áp dụng công thức:

$$Z = \frac{x - \mu}{\sigma}$$

- (Nếu std_price = 0, gán zscore = 0.0 để tránh chia cho 0)
- Nhóm kết quả lại theo timestamp, symbol, và gom các Z-score lại thành một mảng.

```
def compute_zscores(price_df, moving_df):

    joined_df = price_df.join(
        moving_df,
        on=["timestamp", "symbol"],
        how="inner"
    )

    exploded_df = joined_df.select(
        col("timestamp"),
        col("symbol"),
        col("price"),
        explode(col("windows")).alias("window_stats")
    )

    zscore_df = exploded_df.select(
        col("timestamp"),
        col("symbol"),
        struct(
            col("window_stats.window").alias("window"),
            when(
                col("window_stats.std_price") == 0,
                lit(0.0)
            ).otherwise(
                (col("price") - col("window_stats.avg_price")) / col("window_stats.std_price")
            ).alias("zscore_price")
        ).alias("zscore")
    )

    # Group by timestamp and symbol to collect zscores into an array
    result_df = zscore_df.groupBy("timestamp", "symbol").agg(
        collect_list(col("zscore")).alias("zscores")
    )

    return result_df
```

- Chuyển kết quả thành JSON chứa các trường: timestamp (UTC ISO8601), symbol, và zscores (mảng các z-score theo từng cửa sổ). Ghi kết quả vào topic btc-price-zscore

```

> TERMINAL
tienanh@LAPTOP-T95CP219:~/Binance-data-pipeline$ docker exec -it kafka kafka-console-consumer --bootstrap-server localhost:29092 --topic btc-price-zscore --
from-beginning
{"timestamp":"2025-06-15T14:13:00.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":1.7852497628299193}, {"window":"1m","zscore_price":1.0531994794364707}, {"window":"5m","zscore_price":1.053631181404603}, {"window":"15m","zscore_price":1.0382594464038124}, {"window":"30m","zscore_price":1.0382594464038124}, {"window":"1h","zscore_price":1.0382594464038124}]}
{"timestamp":"2025-06-15T14:10:40.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":-1.6775075430369584}, {"window":"1m","zscore_price":-1.8098107741363685}, {"window":"5m","zscore_price":0.4394548690833019}, {"window":"15m","zscore_price":0.4394548690833019}, {"window":"30m","zscore_price":0.4394548690833019}, {"window":"1h","zscore_price":0.4394548690833019}]}
{"timestamp":"2025-06-15T14:10:50.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":-1.3958902696669648}, {"window":"1m","zscore_price":-1.727537147861195}, {"window":"5m","zscore_price":0.10599115074067463}, {"window":"15m","zscore_price":0.10599115074067463}, {"window":"30m","zscore_price":0.10599115074067463}, {"window":"1h","zscore_price":0.10599115074067463}]}
{"timestamp":"2025-06-15T14:12:10.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":1.6911641247728861}, {"window":"1m","zscore_price":1.38496986013445}, {"window":"5m","zscore_price":0.3393335498399105}, {"window":"15m","zscore_price":0.3393335498399105}, {"window":"30m","zscore_price":0.3393335498399105}, {"window":"1h","zscore_price":0.3393335498399105}]}
{"timestamp":"2025-06-15T14:14:00.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":-0.7245135161752148}, {"window":"1m","zscore_price":-1.2735618073120516}, {"window":"5m","zscore_price":-0.6361358990416175}, {"window":"15m","zscore_price":-0.045277964724536844}, {"window":"30m","zscore_price":-0.045277964724536844}, {"window":"1h","zscore_price":-0.045277964724536844}]}
{"timestamp":"2025-06-15T14:11:00.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":0.18312287842467623}, {"window":"1m","zscore_price":-0.8203713423042475}, {"window":"5m","zscore_price":0.3530157127525016}, {"window":"15m","zscore_price":0.3530157127525016}, {"window":"30m","zscore_price":0.3530157127525016}, {"window":"1h","zscore_price":0.3530157127525016}]}
{"timestamp":"2025-06-15T14:13:10.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":0.9026923699646114}, {"window":"1m","zscore_price":0.8397690878498448}, {"window":"5m","zscore_price":0.9899271222317824}, {"window":"15m","zscore_price":1.0070074468199566}, {"window":"30m","zscore_price":1.0070074468199566}, {"window":"1h","zscore_price":1.0070074468199566}]}
{"timestamp":"2025-06-15T14:09:40.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":1.2128457070315508}, {"window":"1m","zscore_price":1.964320495753373}, {"window":"5m","zscore_price":2.266161118492408}, {"window":"15m","zscore_price":2.266161118492408}, {"window":"30m","zscore_price":2.266161118492408}, {"window":"1h","zscore_price":2.266161118492408}]}
{"timestamp":"2025-06-15T14:14:10.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":-1.5353444321897949}, {"window":"1m","zscore_price":-1.486310679115348}, {"window":"5m","zscore_price":-1.2373302175416074}, {"window":"15m","zscore_price":-0.3768418484143149}, {"window":"30m","zscore_price":-0.3768418484143149}, {"window":"1h","zscore_price":-0.3768418484143149}]}
{"timestamp":"2025-06-15T14:11:30.000Z","symbol":"BTCUSDT","zscores":[{"window":"30s","zscore_price":-1.2969713435300332}, {"window":"1m","zscore_price":1.4498113095664145}, {"window":"5m","zscore_price":1.4498113095664145}, {"window":"15m","zscore_price":1.4498113095664145}, {"window":"30m","zscore_price":1.4498113095664145}, {"window":"1h","zscore_price":1.4498113095664145}]}

```

V. Load Stage

1. Ý tưởng giải quyết

- Nhóm đã thực hiện việc ghi dữ liệu từ topic Kafka btc-price-zscore (đã tính toán ở giai đoạn Transform) vào MongoDB bằng cách sử dụng Spark Structured Streaming kết hợp với MongoDB Spark Connector.

2. Chi tiết thực thi

- Khởi tạo SparkSession: nhóm em tạo một SparkSession với cấu hình để hoạt động trong múi giờ UTC và sử dụng gói MongoDB Spark Connector (mongo-spark-connector_2.12:10.3.0) để có thể ghi dữ liệu trực tiếp vào MongoDB từ Spark..

```
CONFIG = {
    "KAFKA_TOPIC_ZSCORE": "btc-price-zscore",
    "KAFKA_BOOTSTRAP_SERVERS": "localhost:9092",
    "MONGO_URI": "mongodb+srv://cq05:cq05@hcmus.nzorvvx.mongodb.net/?retryWrites=true&w=majority&appName=HCMUS",
    "CHECKPOINT_LOCATION": "tmp/checkpoint_zscore_mongo"
}

def init_spark():
    spark = SparkSession.builder \
        .appName("BTC Price Z-Score to MongoDB") \
        .config("spark.sql.session.timeZone", "UTC") \
        .config("spark.jars.packages", "org.mongodb.spark:mongo-spark-connector_2.12:10.3.0") \
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    return spark
```

- Đọc dữ liệu từ Kafka: đọc dữ liệu từ topic Kafka btc-price-zscore – là nơi chứa các bản ghi Z-score đã được tính toán ở bước Transform. Dữ liệu Kafka có định dạng JSON, vì vậy chúng ta cần xử lý bước phân tích cú pháp (parse) tiếp theo.

```
def read_kafka_topic(spark):
    df = spark.readStream \
        .format("kafka") \
        .option("kafka.bootstrap.servers", CONFIG["KAFKA_BOOTSTRAP_SERVERS"]) \
        .option("subscribe", CONFIG["KAFKA_TOPIC_ZSCORE"]) \
        .option("startingOffsets", "earliest") \
        .load()
    return df
```

- Phân tích và làm sạch dữ liệu đầu vào
- Dữ liệu được phân tích dựa trên schema bao gồm: timestamp: thời gian tính toán Z-score. symbol: cặp giao dịch (ví dụ: BTCUSDT). zscores: một mảng chứa các đối tượng {window, zscore_price} cho mỗi cửa sổ trượt.
- Sau khi parse JSON, chúng em sử dụng withColumn("timestamp", to_timestamp(...)) để chuyển chuỗi thời gian về kiểu timestamp chuẩn, đồng thời áp dụng withWatermark("timestamp", "10 seconds") nhằm đảm bảo có thể xử lý dữ liệu đến trễ trong vòng 10 giây.


```
def parse_zscore_stream(df):
    window_schema = StructType([
        StructField("window", StringType(), True),
        StructField("zscore_price", DoubleType(), True)
    ])
    schema = StructType([
        StructField("timestamp", StringType(), True),
        StructField("symbol", StringType(), True),
        StructField("zscores", ArrayType(window_schema), True)
    ])
    df_parsed = (
        df.selectExpr("CAST(value AS STRING)")
        .select(from_json(col("value"), schema).alias("data"))
        .select("data.*")
        .withColumn("timestamp", to_timestamp(col("timestamp"), "yyyy-MM-dd'T'HH:mm:ss.SSSXXX"))
        .filter(col("zscores").isNotNull())
        .withWatermark("timestamp", "10 seconds")
    )
    return df_parsed
```

- Ghi dữ liệu vào MongoDB: sau khi tách zscores, ta lọc từng dòng theo giá trị window và ghi vào các MongoDB collection tương ứng với tên
 - btc-price-zscore-30s
 - btc-price-zscore-1m
 - btc-price-zscore-5m
 - btc-price-zscore-15m
 - btc-price-zscore-30m
 - btc-price-zscore-1h

```
def write_to_mongodb(df):

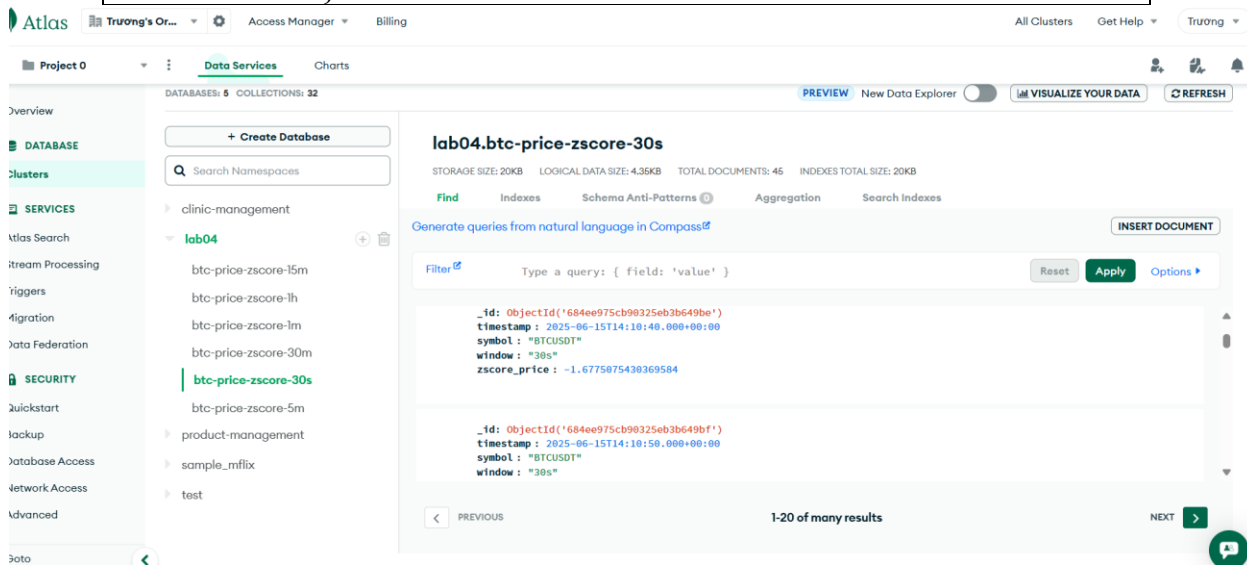
    exploded_df = df.select(
        col("timestamp"),
        col("symbol"),
        explode(col("zscores")).alias("zscore")
    ).select(
        col("timestamp"),
        col("symbol"),
        col("zscore.window").alias("window"),
        col("zscore.zscore_price").alias("zscore_price")
    )

    windows = ["30s", "1m", "5m", "15m", "30m", "1h"]

    for window in windows:
        window_df = exploded_df.filter(col("window") == window)
        query = window_df.writeStream \
            .format("mongodb") \
            .option("spark.mongodb.connection.uri", CONFIG["MONGO_URI"]) \
            .option("spark.mongodb.database", "lab04") \
            .option("spark.mongodb.collection", f"btc-price-zscore-{window}") \
            .option("checkpointLocation", f"{CONFIG['CHECKPOINT_LOCATION']}/{window}") \
            .outputMode("append") \
            .start()
        print(f"Started streaming query for collection btc-price-zscore-{window}")
```

- Dữ liệu sẽ được đẩy lên Mongo Atlas có cấu trúc như sau:

- {
- “timestamp”: <ISO8601 UTC timestamp>,
- “symbol”: <string>,
- “window”: <string>,
- “zscore_price”: <float>
- }



VI. Bonus Stage

1. Ý tưởng giải quyết

- Đề bài yêu cầu đọc dữ liệu từ topic btc-price để tìm cửa sổ (khoảng thời gian) ngắn nhất để xảy ra sự tăng/ giảm giá trị của đồng btc.
- Ta sẽ tiến hành đọc dữ liệu từ topic btc-price, với mỗi record p tại timestamp t ta sẽ tiến hành kiểm tra xem trong khoảng thời gian 20s tới, có record p1/p2 nào có sự tăng/giảm giá trị so với p thông qua phép so sánh.
- Vì dữ liệu đọc từ topic btc-price được đọc xuống theo từng batch, với mỗi batch tới ta sẽ chuyển đổi các dữ liệu này thành dataframe, lọc ra record p1/p2 có price đầu tiên theo timestamp lớn/nhỏ hơn record p đang xét, tính khoảng thời gian chênh lệch giữa 2 timestamp để xác độ dài của window.
- **Lưu ý 1:** vì dữ liệu được đọc theo từng batch, khả năng cao các record p đầu batch với timestamp t sẽ có đầy đủ thông tin về các record p' có t' nằm trong khoảng (t;t+20], tuy nhiên các record p' với timestamp t' về sau có khả năng thiếu đi độ dài cửa sổ tối đa, tức là bị thiếu đi thông tin về các record thuộc khoảng (t',t'+20] vì các record này thuộc về batch tiếp theo. Do đó chúng ta cần lưu trạng thái của các record chưa được xử lý vì thiếu thông tin về cửa sổ tối đa (+20) của nó để đợi batch tiếp theo. Kỹ thuật này được gọi chung là stateful operations.
- **Lưu ý 2:** Đề bài yêu cầu dữ liệu được gửi (publish) vào các topic Kafka theo chế độ append, tức là chỉ thêm mới vào cuối dòng dữ liệu, không chỉnh sửa hoặc xóa dữ liệu cũ. 'mapGroupsWithState' và 'flatMapGroupsWithState' là hai stateful operation giúp thay

đôi trạng thái của key, giúp ta kiểm tra xem record này đã đủ điều kiện để thực hiện tính cửa sổ ngắn nhất chưa (đã có thông tin về các record có t thuộc $(t; t+20]$). Tuy nhiên hai phương thức này chỉ hỗ trợ cho update mode. Để sử dụng stateful operation cho append mode, t chỉ có thể cắt phần dữ liệu chưa đủ điều kiện, lưu tạm thời và đợi xử lý chung với batch sau. Cắt các record đã được xử lý và publish lên topic tương ứng.

- **Ý tưởng:**

- Dữ liệu sẽ được gửi tới theo từng batch, với mỗi timestamp kiểm tra các timestamp trong vòng 20 giây sau để quyết định giá trị của window.
- Chuyển batch thành Pandas để xử lý dễ dàng.
- Tính thời gian ngắn nhất trong 20 giây tới mà giá cao hơn hoặc thấp hơn giá hiện tại.
- Gửi kết quả đến Kafka (btc-price-higher, btc-price-lower).
- Đối với dữ liệu trong 1 batch chưa có đủ thông tin về 20 giây sau thì sẽ cắt ra và lưu vào STATE_PATH.
- Khi sang batch mới sẽ gộp với dữ liệu từ các batch trước (đọc từ Parquet lưu ở STATE_PATH).

- **Hạn chế:**

- Chưa thể triển khai các stateful operations của spark do vấn đề xung đột phiên bản giữa các chương trình python, spark...
- Việc sử dụng pandas tuy dễ dàng viết mã nguồn nhưng hiện đang cho hiệu suất khá kém.
- Không thể xử lý lượng dữ liệu quá lớn và nhanh lấy từ phần Extract.
- Thường xuyên xảy ra hiện tượng nghẽn dữ liệu, thất cổ chai.

2. Chi tiết thực thi

- Cấu hình chung các thư mục, topic lưu trữ dữ liệu:

```
CONFIG = {  
    "KAFKA_BOOTSTRAP": "localhost:9092",  
    "KAFKA_TOPIC_SOURCE": "btc-price",  
    "KAFKA_TOPIC_HIGHER": "btc-price-higher",  
    "KAFKA_TOPIC_LOWER": "btc-price-lower",  
    "STATE_PATH": "./src/Bonus/btc_state_buffer",  
    "CHECKPOINT": "./src/Bonus/main_checkpoint"  
}
```

- Khởi tạo Spark:

```
def init_spark():
    os.environ["PYSPARK_PYTHON"] = "python"
    spark = SparkSession.builder \
        .appName("KafkaBTCCConsumerSimple") \
        .config("spark.sql.shuffle.partitions", "2") \
        .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.5") \
        .getOrCreate()
    spark.sparkContext.setLogLevel("WARN")
    return spark
```

- Định nghĩa cấu trúc của dữ liệu: vì muốn giữ nguyên format của timestamp nên sẽ lấy xuống dạng chuỗi String.

```
schema = StructType([
    StructField("symbol", StringType()),
    StructField("price", FloatType()),
    StructField("timestamp", StringType())
])
```

- Hàm `compute_windows`: hàm chịu trách nhiệm chính cho việc tính toán của vấn đề, tìm ra độ dài cửa sổ `higher` và `lower`.
 - Đối với mỗi bản ghi, nhóm các bản ghi có timestamp trong khoảng $(t; t+20]$ để xác định price có `higher` hay `lower`, kết quả trả về là một list các bản ghi đủ điều kiện để xét.
 - Các bản ghi đủ điều kiện xét nhưng không có giá trị thỏa mãn sẽ trả về 20.
- Hàm `load_previous_state`: trả về các bản ghi thuộc về batch trước nhưng chưa đủ điều kiện để xét tìm cửa sổ thỏa mãn.
- Hàm `update_state_buffer`: Lưu lại những dữ liệu mới nhất trong 20 giây gần đây vào thư mục `STATE_PATH` (để sử dụng ở các batch sau). Tức là cập nhật lại các bản ghi sau mỗi batch xử lý, xem nên giữ lại thêm batch nào.
- Hàm `write_to_kafka`: dùng để đẩy các dữ liệu đã được xử lý lên topic tương ứng.

```
INFO:py4j.clientserver:Received command c on object id p0
INFO: __main__:Processing batch 2
INFO: __main__:Batch 2 rows after cleaning: 619
INFO: __main__:Wrote 619 higher records to Kafka
INFO: __main__:Wrote 619 lower records to Kafka
INFO: __main__:Updated state buffer with 201 records
25/06/17 21:13:16 WARN ProcessingTimeExecutor: Current batch is falling behind. The trigger interval is 5000 milliseconds, but spent 60665 milliseconds
INFO:py4j.clientserver:Received command c on object id p0
INFO: __main__:Processing batch 3
INFO: __main__:Batch 3 rows after cleaning: 607
[[Stage 30:> (0 + 12) / 12]
```

- Hàm `process_batch` -hàm chính xử lý mỗi batch dữ liệu Spark Structured Streaming nhận từ Kafka:
 - Ghi log thông báo bắt đầu xử lý batch.

- Chuyển Spark DataFrame thành Pandas DataFrame để dễ xử lý. Chuyển event-time thành datetime để xử lý logic.
- Ghi log số dòng còn lại sau khi làm sạch.
- Gọi hàm load_previous_state() để lấy dữ liệu đã lưu từ các batch trước (20 giây gần nhất).
- Gộp dữ liệu mới với dữ liệu cũ (batch chưa xử lý), sắp xếp lại timestamp.
- Gọi hàm compute_windows(...) để tính xem trong 20s sau, giá có tăng/giảm không, và sau bao lâu. Trả về 2 danh sách: higher_records, lower_records.
- Ghi danh sách kết quả higher/lower vào topic tương ứng.
- Gọi hàm update_state_buffer để **lưu lại các bản ghi mới nhất (trong 20s gần đây)** để phục vụ cho batch kế tiếp.

```
{"timestamp": "2025-06-17T21:11:57.820", "higher_window": 0.32}
{"timestamp": "2025-06-17T21:11:57.920", "higher_window": 0.22}
{"timestamp": "2025-06-17T21:11:58.040", "higher_window": 0.1}
{"timestamp": "2025-06-17T21:11:58.140", "higher_window": 0.1}
{"timestamp": "2025-06-17T21:11:58.240", "higher_window": 5.5}
{"timestamp": "2025-06-17T21:11:58.340", "higher_window": 5.4}
{"timestamp": "2025-06-17T21:11:58.420", "higher_window": 5.32}
{"timestamp": "2025-06-17T21:11:58.520", "higher_window": 5.22}
{"timestamp": "2025-06-17T21:11:58.640", "higher_window": 5.1}
{"timestamp": "2025-06-17T21:11:58.740", "higher_window": 5.0}
{"timestamp": "2025-06-17T21:11:58.820", "higher_window": 4.92}
{"timestamp": "2025-06-17T21:11:58.920", "higher_window": 4.82}
```

```
{"timestamp": "2025-06-17T21:22:37.440", "higher_window": 20.0}
{"timestamp": "2025-06-17T21:22:37.520", "higher_window": 20.0}
{"timestamp": "2025-06-17T21:22:37.620", "higher_window": 20.0}
{"timestamp": "2025-06-17T21:22:37.740", "higher_window": 20.0}
{"timestamp": "2025-06-17T21:22:37.840", "higher_window": 0.5}
{"timestamp": "2025-06-17T21:22:37.920", "higher_window": 0.42}
{"timestamp": "2025-06-17T21:22:38.040", "higher_window": 0.3}
{"timestamp": "2025-06-17T21:22:38.140", "higher_window": 0.2}
{"timestamp": "2025-06-17T21:22:38.220", "higher_window": 0.12}
{"timestamp": "2025-06-17T21:22:38.340", "higher_window": 20.0}
{"timestamp": "2025-06-17T21:22:38.440", "higher_window": 20.0}
{"timestamp": "2025-06-17T21:22:38.520", "higher_window": 20.0}
```

PHẦN 2: BÁO CÁO NHÓM

- Phương thức trao đổi và làm việc:
 - Tất cả các thành viên được yêu cầu phải tự trang bị các kiến thức cơ bản về lý thuyết cũng

nhu thực hành đơn giản với Kafka.

- Giai đoạn Extract yêu cầu tất cả các thành viên đều phải thực hiện, mục đích là để thiết lập các môi trường cần thiết cho việc sử dụng Kafka giải quyết bài toán.
- Giai đoạn Transform, Load, Bonus sẽ được thực hiện nhóm với một cá nhân cụ thể chịu trách nhiệm chính.
- Nhóm tổ chức họp nhóm trao đổi sau 2-3 ngày làm việc cá nhân kết hợp trao đổi liên tục hằng ngày trên các công cụ trao đổi trực tuyến.
- Phiên họp nhằm kiểm tra kết quả, ý tưởng giải quyết cho từng vấn đề.
- Giao công việc cho từng thành viên trên workspace của nhóm sau khi thống nhất kết quả.
- Bảng phân công công việc:

| Thành viên | Công việc | Lưu ý |
|-------------------|---|---|
| Trương Tiến Anh | Chịu trách nhiệm chính cho giai đoạn Transform, Load | Các thành viên vẫn thực hiện bài tập riêng lẻ nhưng kết hợp trao đổi để cải thiện phương pháp, đánh giá hiệu quả và tương thích |
| Nguyễn Minh Cường | Chịu trách nhiệm chính cho giai đoạn Extract, Load Tham gia hỗ trợ giai đoạn Transform | |
| Đinh Viết Lợi | Điều hành công việc nhóm, viết báo cáo. Chịu trách nhiệm chính cho giai đoạn Bonus | |
| Nguyễn Trần Lợi | Chịu trách nhiệm chính cho giai đoạn Transform, Load | |