Real-Time Market Data Streaming and Sentiment Analysis Platform

M314 – Big Data

Fall 2025

Submitted by

Ilyas Hakkou
Imane Rahali

Supervised by

Dr. Fahd Kalloubi

January 2, 2026

Abstract

This project presents a comprehensive real-time data processing pipeline for cryptocurrency market data and financial news sentiment analysis. The system leverages a modern big data architecture combining Apache NiFi for data ingestion, Apache Kafka for message streaming, Apache Spark for real-time analytics, Elasticsearch for storage, and Kibana for interactive visualization. The platform ingests live cryptocurrency trade data from Binance and FinnHub APIs, processes streaming data through windowed aggregations, and performs automated sentiment analysis on financial news articles using natural language processing techniques. The resulting dashboard provides real-time insights into market trends, trading patterns, and news sentiment, demonstrating the practical application of big data technologies in financial analytics.

Contents

1 Introduction

The rapid growth of cryptocurrency markets and the increasing volume of financial news have created a need for real-time data processing and analysis systems. This project implements an end-to-end streaming data pipeline that processes live market data and performs sentiment analysis on financial news articles.

The system architecture leverages several big data technologies working in concert:

- **Apache NiFi** for data ingestion from Binance and FinnHub APIs
- **Apache Kafka** for distributed message queuing and stream buffering
- **Apache Spark Streaming** for real-time analytics and data processing
- **Elasticsearch** for efficient storage and indexing
- **Kibana** for interactive data visualization and dashboard creation

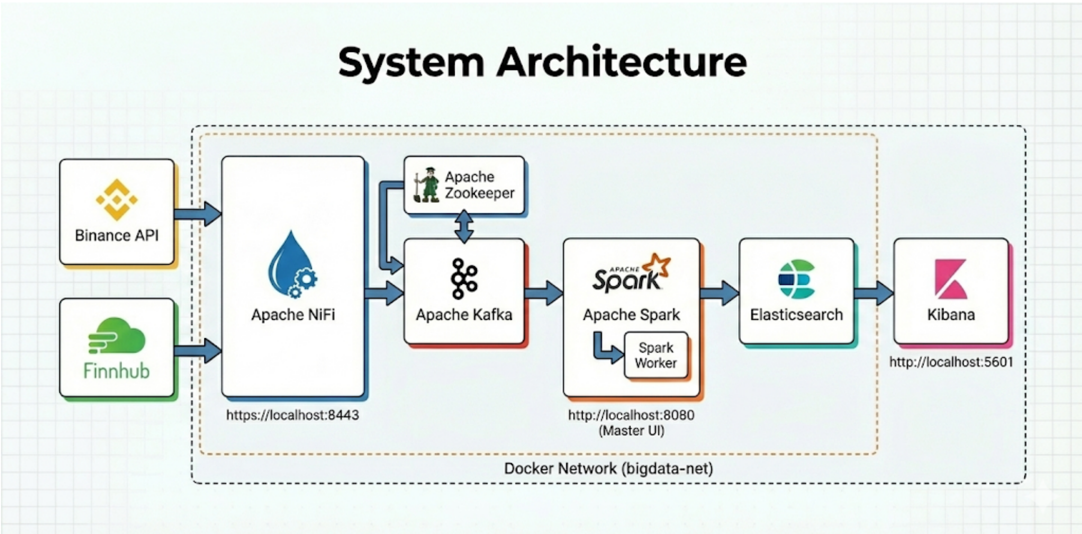

Figure 1: System Architecture Overview

The platform processes two primary data streams: cryptocurrency trades and financial news. Trade data undergoes windowed aggregation to compute metrics such as average price, trading volume, and trade count. News articles are analyzed using TextBlob, a natural language processing library, to determine sentiment polarity and categorize articles as positive, negative, or neutral.

2 Deployment and Configuration

This section provides a comprehensive guide to deploying and configuring the entire data pipeline.

2.1 Starting the Services

1. **Clone the Repository**

```
git clone https://github.com/ilyasishere/market-data-streaming-viz.git
cd market-data-streaming-viz
```

2. **Start Docker Containers**

```
docker compose up -d
```

3. **Access the Services**

- NiFi: https://localhost:8443/nifi (login with `admin` / `password12345678`)
- Note: Use `https` protocol, as `http` will not work
- NiFi Registry: http://localhost:18080/nifi-registry
- Kibana: http://localhost:5601

2.2 Importing the NiFi Workflow

2.2.1 Set Up NiFi Registry Client

In NiFi, create a new `NifiRegistryFlowRegistryClient` under Controller Services.

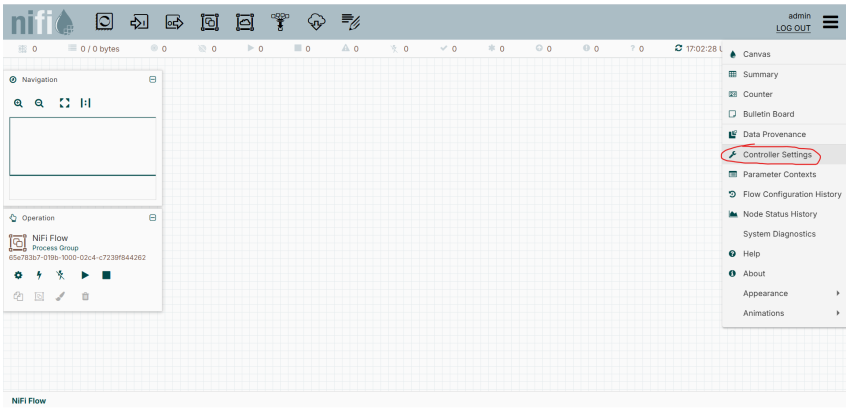

Figure 2: Creating NiFi Registry Client

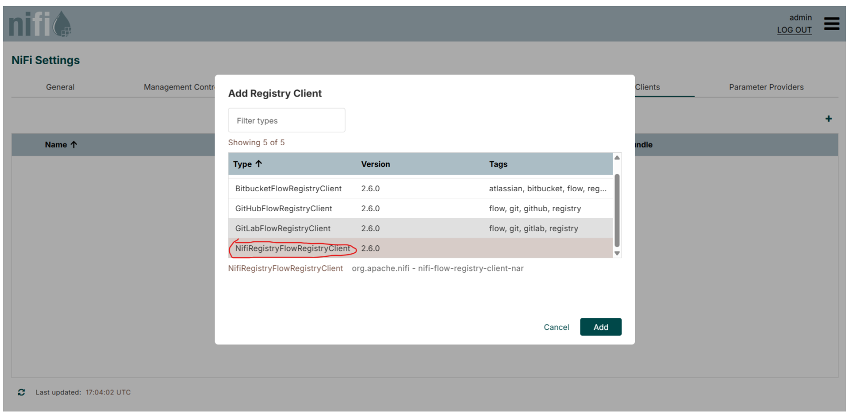

Figure 3: Registry Client Configuration

Set the URL to `http://nifi-registry:18080`.

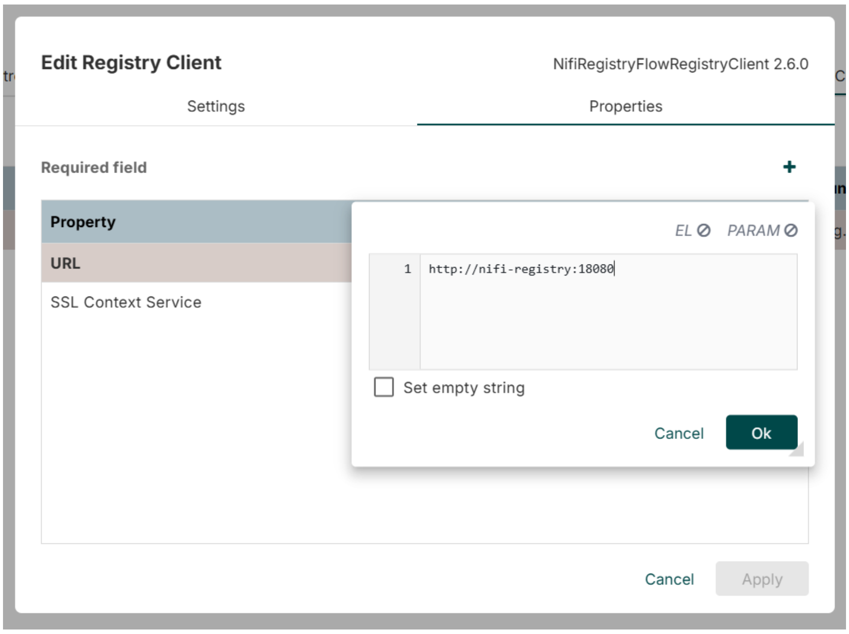

Figure 4: Setting Registry URL

2.2.2 Create a Bucket in NiFi Registry

Open the Registry UI at http://localhost:18080/nifi-registry, go to Settings, and create a bucket named `main-flows`.

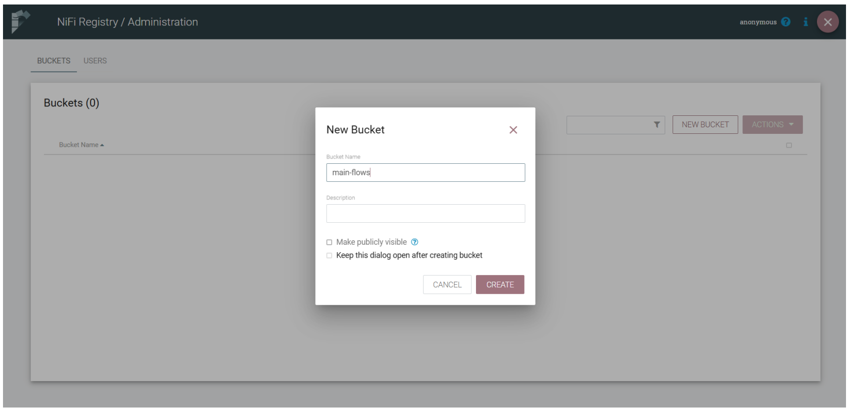

Figure 5: Creating Registry Bucket

2.2.3 Import the Workflow

On the Registry homepage, click **Import New Flow**, set the flow name to `main`, select the `main-flows` bucket, upload the `workflow.json` file, and click **Import**.

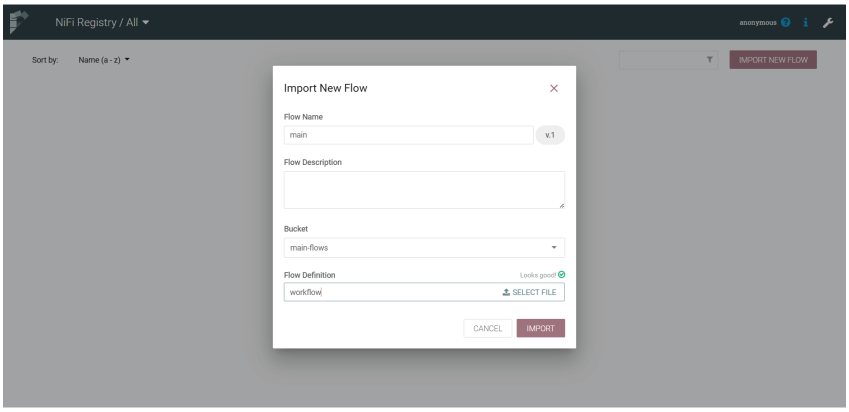

Figure 6: Importing Workflow from JSON

2.2.4 Deploy the Workflow in NiFi

In the NiFi UI, drag the **Import from Registry** icon onto the canvas.

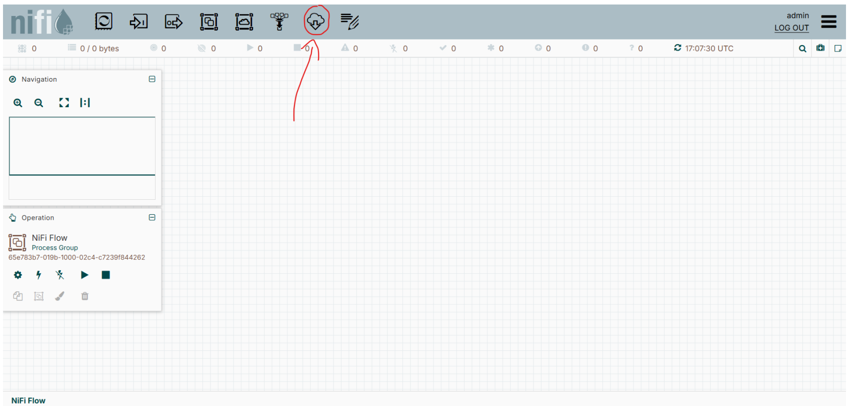

Figure 7: Import from Registry Icon

Select the `main-flows` bucket and the `main` flow, then click **Import**.

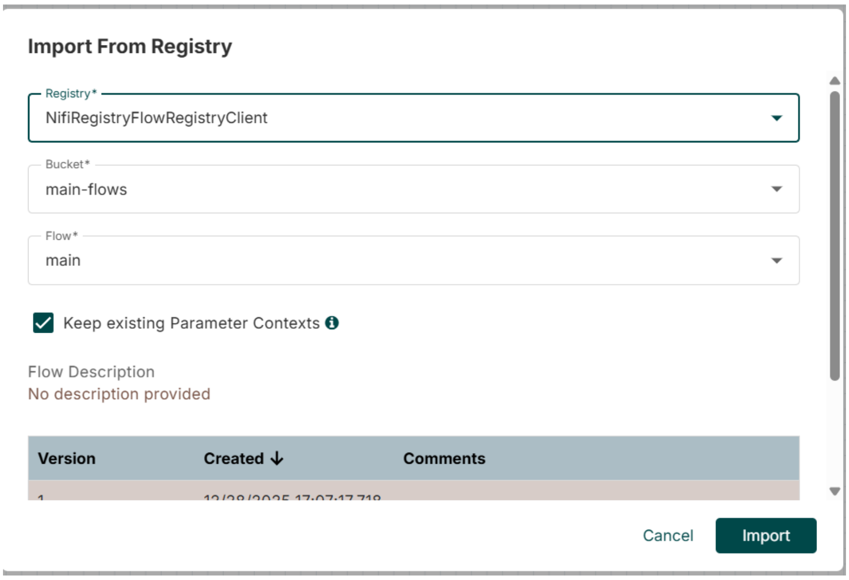

Figure 8: Selecting Flow to Import

2.2.5 Enable and Start the Workflow

Right-click the `main` process group and click **Enable all controller services**.

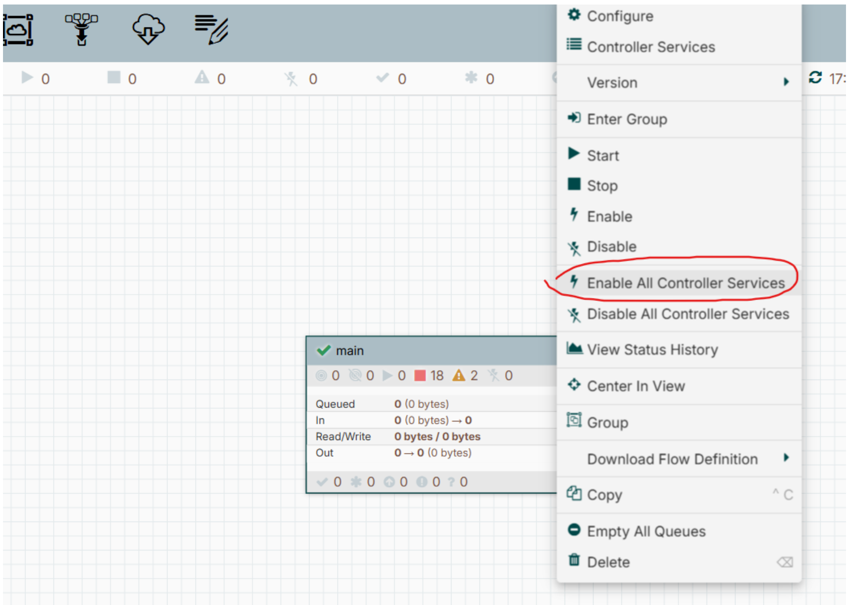

Figure 9: Enabling Controller Services

Right-click again and select **Start** to run the workflow.

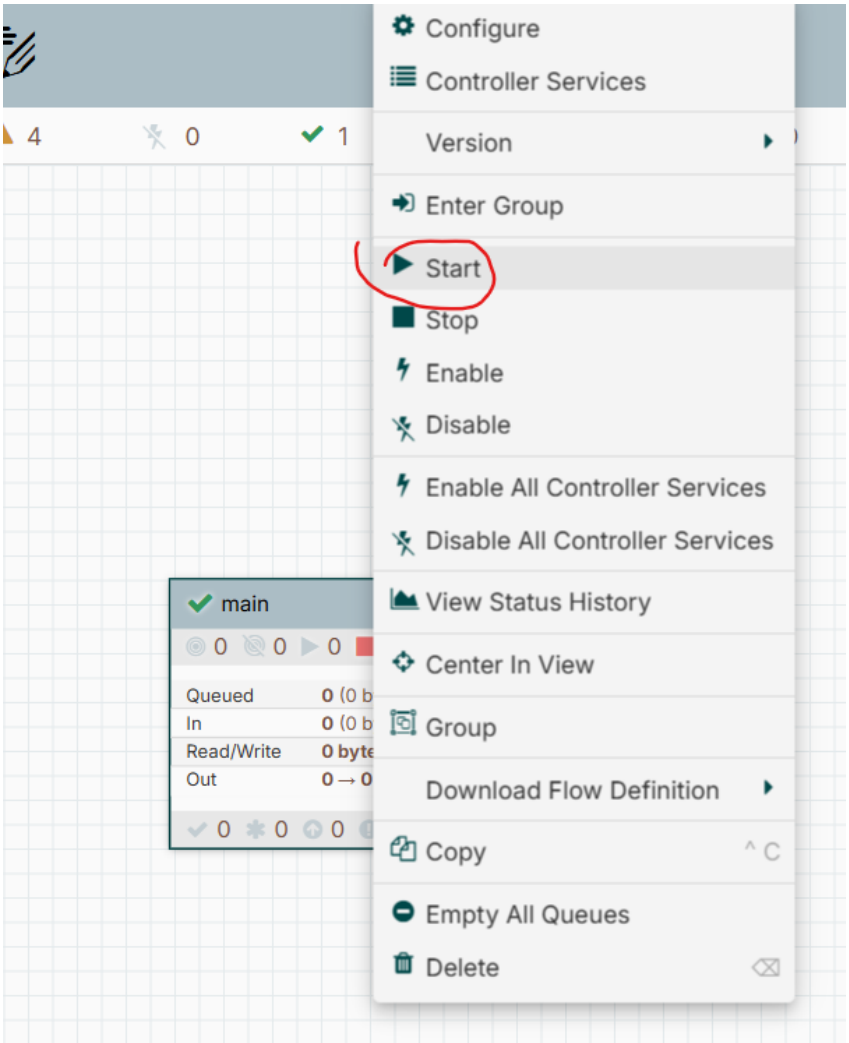

Figure 10: Starting the Workflow

2.3 Launching the Spark Streaming Jobs

Note: Before proceeding, ensure all Docker services (spark-master, spark-worker, kafka, nifi, elasticsearch, etc.) have fully started. Verify by checking `docker compose ps` to confirm all services show as "Up" or "healthy".

The Spark jobs process both trades and news data, performing analytics and sentiment analysis. Sentiment analysis is performed using TextBlob, a Python library that provides a pre-trained Naive Bayes classifier for determining whether text is positive, negative, or neutral. TextBlob is automatically installed when the Spark containers start (configured in `docker-compose.yaml` under both `spark-master` and `spark-worker` service commands).

2.3.1 Start Both Streaming Jobs

On Linux/Mac:

```
./start_processing.sh
```

On Windows:

```
bash ./start_processing.sh
```

Figure 11: Executing Spark Job Launch Script

The script will launch:

- **Trades Analytics Job**: Processes cryptocurrency trades and calculates 1-minute window aggregations (avg price, volume, trade count)
- **News Sentiment Job**: Analyzes news articles using TextBlob for sentiment scoring

2.3.2 Verify Jobs Are Running

Check the Spark UI at http://localhost:8080. You should see both jobs running, each allocated 1 core:

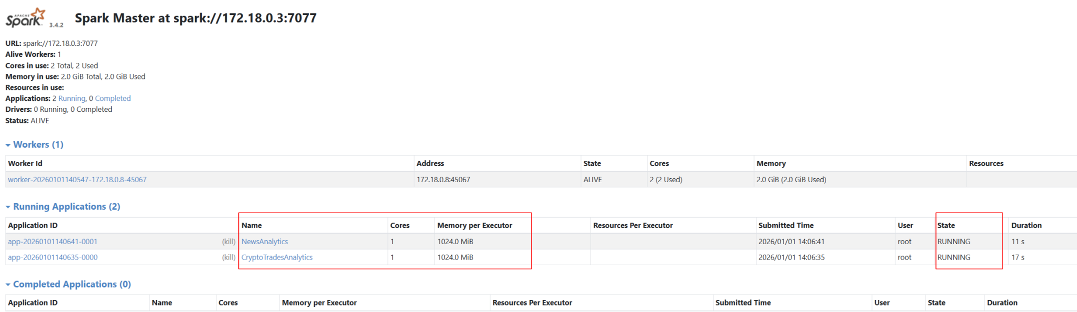

Figure 12: Spark Web UI Showing Running Jobs

Wait 1-2 minutes for data to start flowing, then check Elasticsearch:

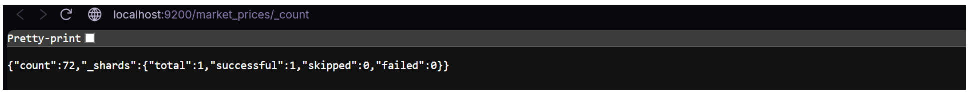

Figure 13: Verifying Market Prices Index

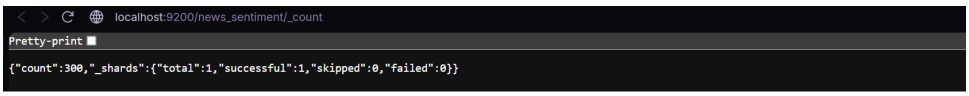

Figure 14: Verifying News Sentiment Index

2.4 Visualizing Data in Kibana

Once data is flowing through the pipeline, you can view the pre-built dashboard in Kibana.

2.4.1 Open Kibana

Navigate to `http://localhost:5601`

2.4.2 Import the Dashboard

Click the hamburger menu () → **Management** → **Stack Management**, then under "Kibana", click **Saved Objects**. Click **Import** and select the `kibana_dashboard.ndjson` file.

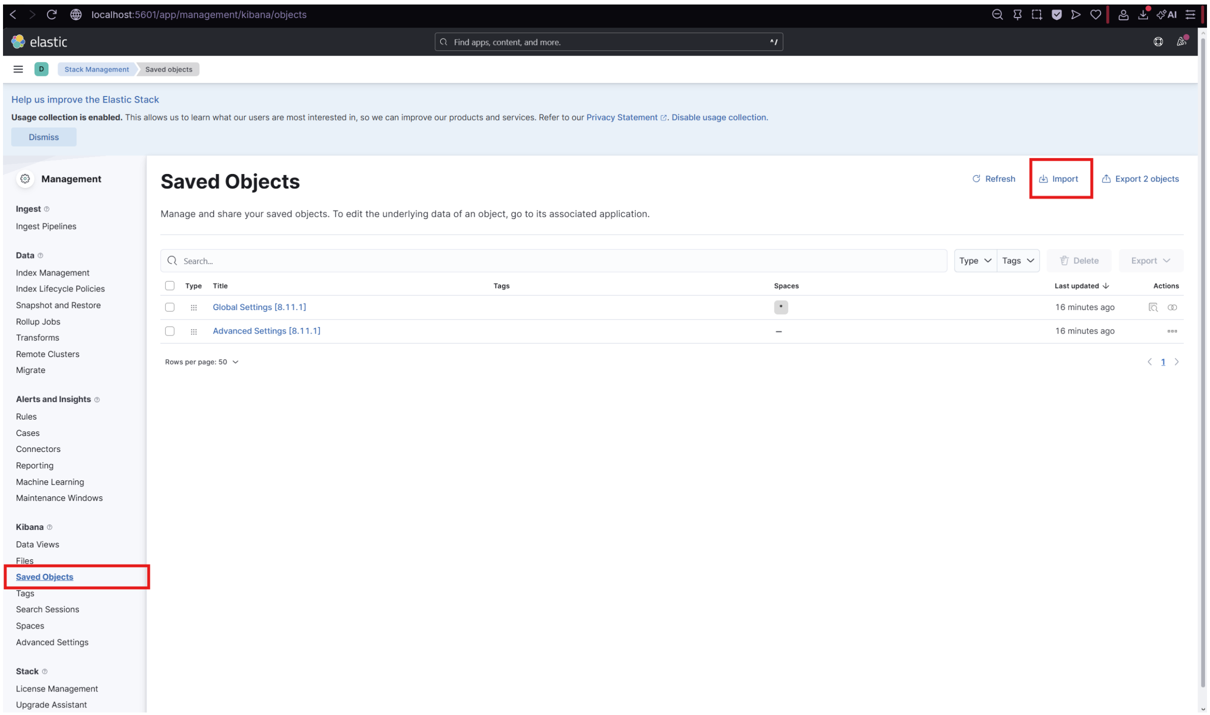

Figure 15: Kibana Import Dialog

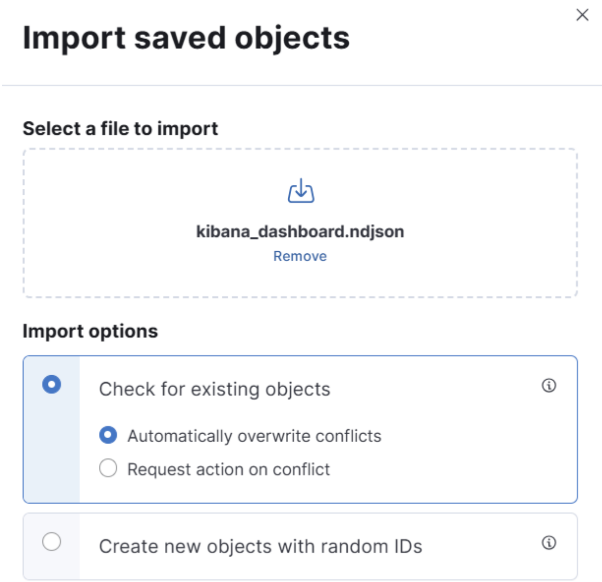

Figure 16: Import Dialog

2.4.3 View the Dashboard

Click the hamburger menu () → **Analytics** → **Dashboard** and select **Market Data Real-Time Dashboard**.

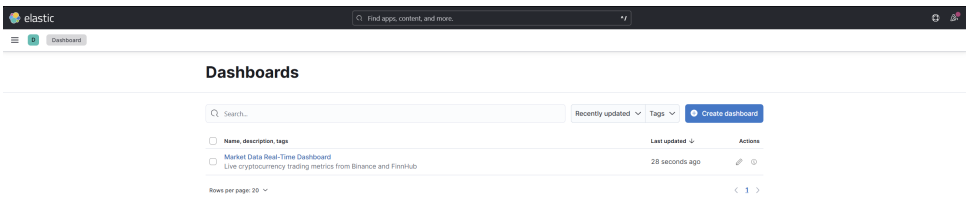

Figure 17: Dashboard Selection Screen

2.4.4 Configure Time Range and Auto-Refresh

Set the time range to **Last 15 minutes** in the top-right corner. Click the refresh icon and select an auto-refresh interval (e.g., 10 seconds).

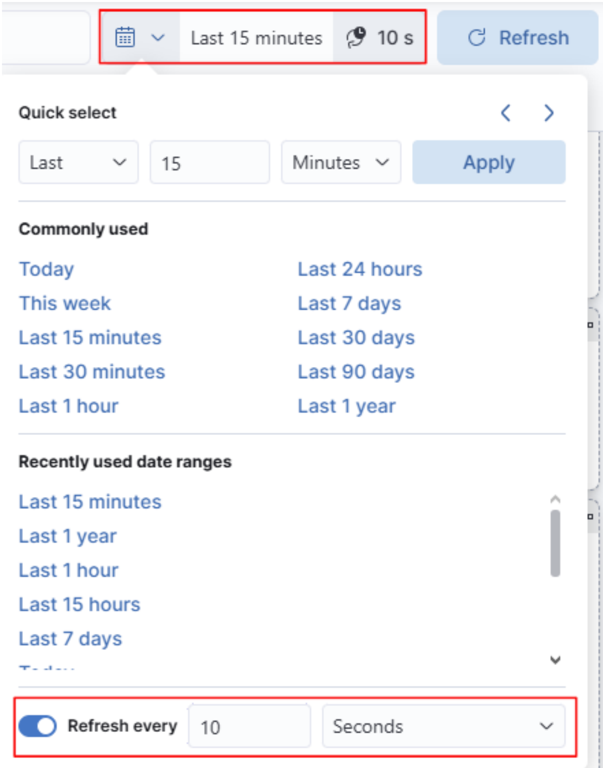

Figure 18: Time Range and Auto-Refresh Configuration

The dashboard updates automatically as new data streams through the pipeline.

2.4.5 Dashboard Overview

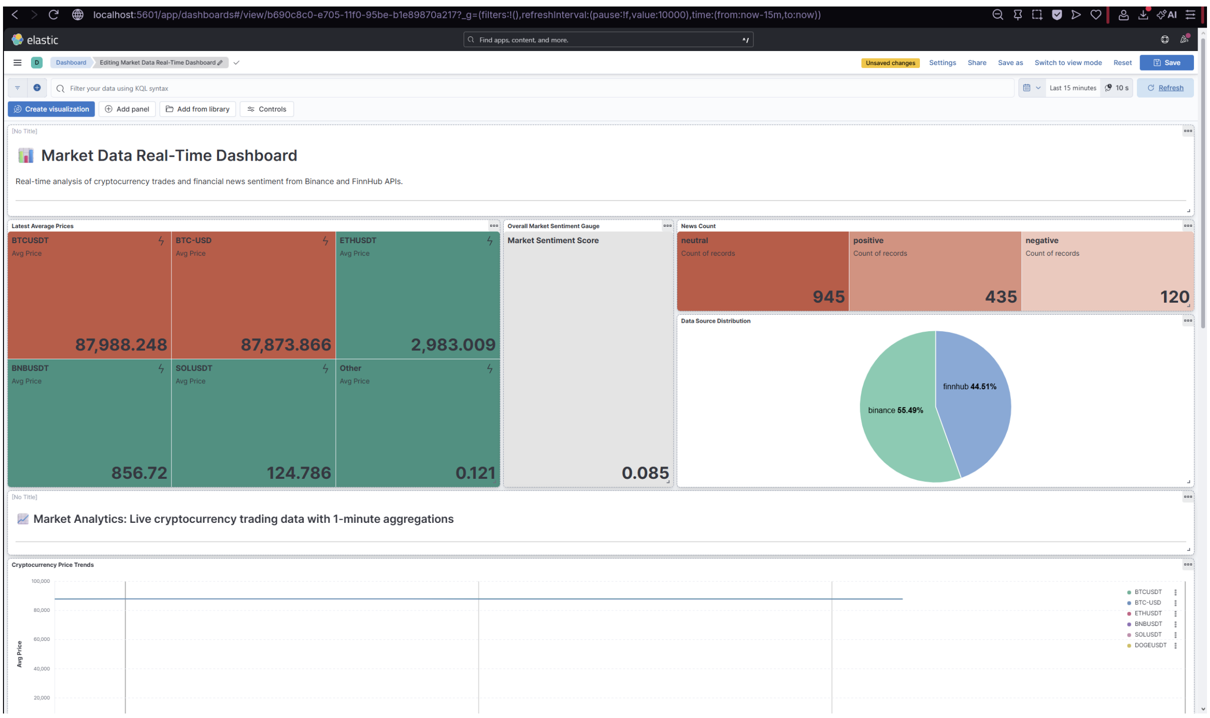

Figure 19: Complete Dashboard Overview

The dashboard is organized into three main sections:

Section 1: Key Metrics

- **Latest Average Prices**: Real-time average prices for each cryptocurrency symbol
- **Overall Market Sentiment Gauge**: Visual gauge showing the average sentiment score across all news articles, ranging from -1.0 (very negative) to +1.0 (very positive), calculated as the mean of all sentiment polarity scores
- **News Count**: Distribution of news articles by sentiment (positive, negative, neutral)
- **Data Source Distribution**: Breakdown of data by source (Binance, FinnHub)

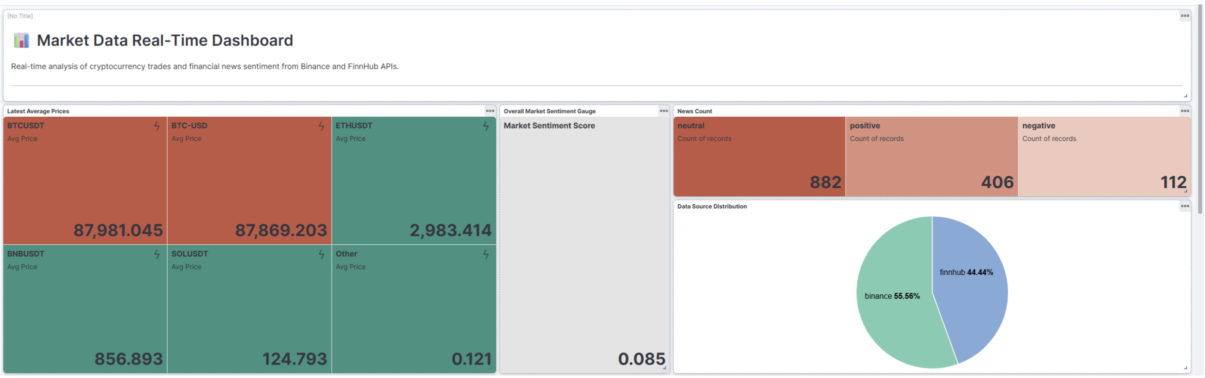

Figure 20: Key Metrics Section

Section 2: Market Analytics

- **Cryptocurrency Price Trends**: Line chart showing price movements over time
- **Trading Volume by Symbol**: Bar chart of trading volume for each cryptocurrency
- **Trading Activity Over Time**: Time-series visualization of trading activity
- **Trade Activity Heatmap**: Heat map showing trade intensity across symbols and time
- **Symbol Metrics Table**: Detailed table with avg price, volume, and trade count per symbol

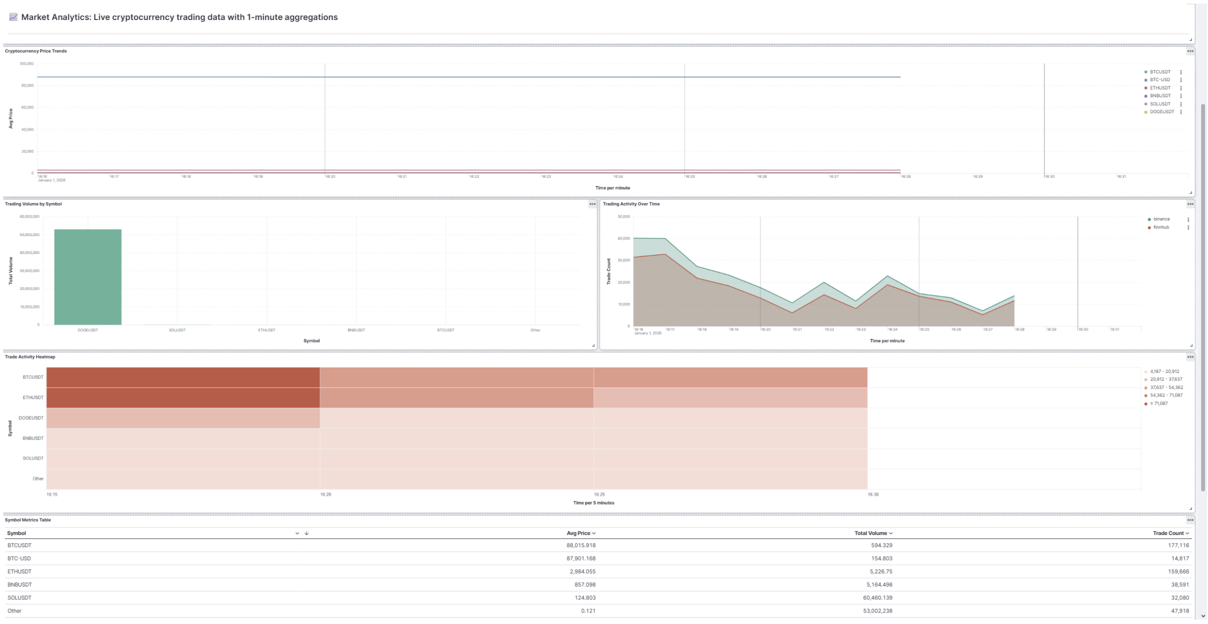

Figure 21: Market Analytics Section

Section 3: News Sentiment Analysis

- **News Sentiment Distribution**: Pie chart showing percentage of positive, negative, and neutral news
- **Sentiment Trends Over Time**: Line chart tracking sentiment score changes over time
- **Sentiment Score Distribution**: Histogram showing the distribution of sentiment scores

Figure 22: News Sentiment Analysis Section

3 System Architecture and Implementation

This section describes the technical implementation details of each component in the data pipeline.

3.1 Data Ingestion with Apache NiFi

Apache NiFi fetches real-time trade data from Binance and news data from FinnHub APIs, then pushes this data into Kafka topics.

The figure below shows the complete NiFi workflow. The workflow on the right is responsible for fetching trade data using websockets, while the left side handles news data fetching via REST API calls.

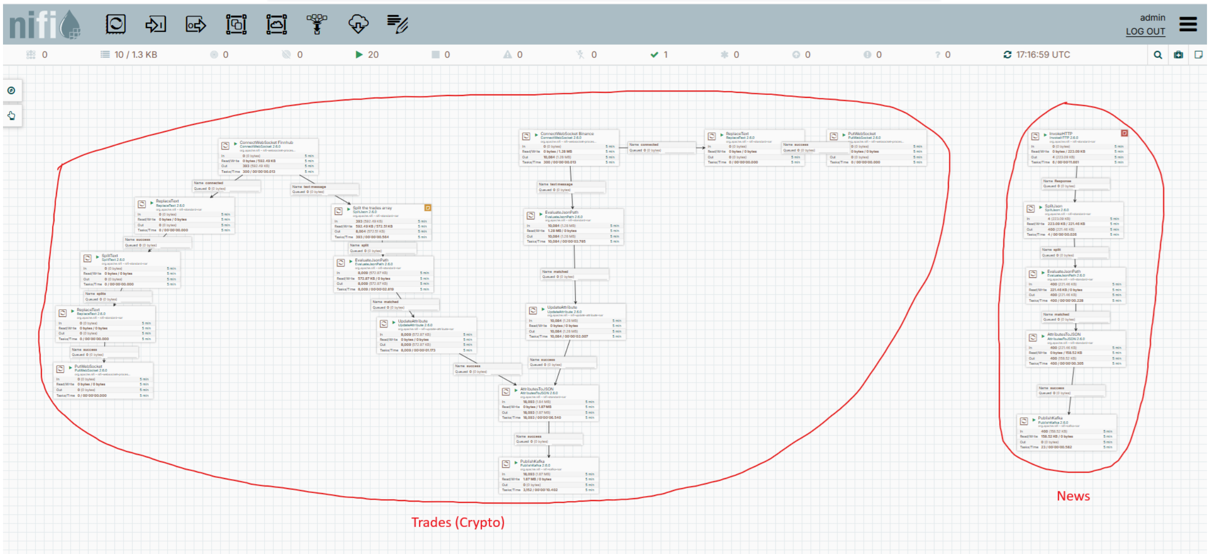

Figure 23: Complete NiFi Workflow

3.1.1 Trade Data Ingestion

For the ConnectWebSocket processor to work, we need to create a Jetty WebSocketClient Service. We then configure the WebSocket URI property to point to the FinnHub trades stream endpoint with the API key.

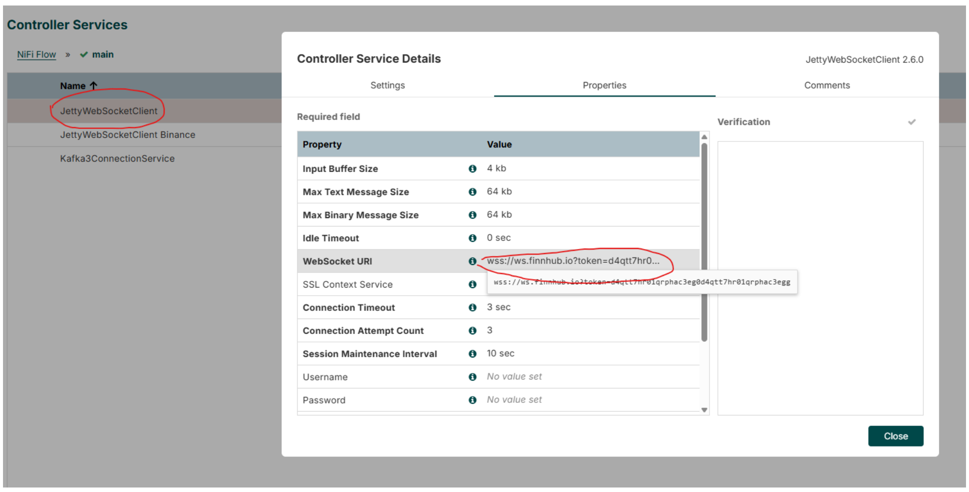

Figure 24: WebSocket Configuration

The left-most path sends a subscription message to the FinnHub API to start receiving trades data for the specified symbols.

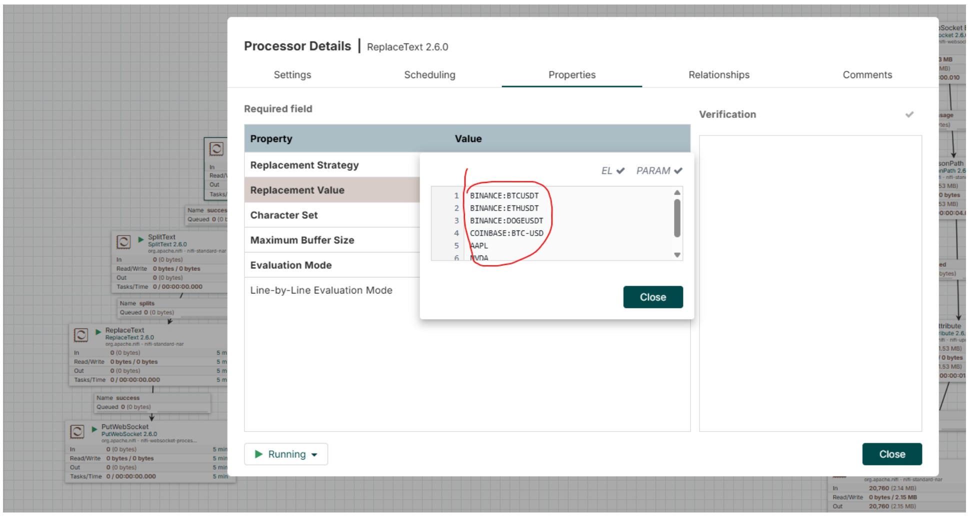

Figure 25: Subscription Message Configuration

The path next to it receives the incoming trades data, splits the JSON array into individual messages (because FinnHub sends trades in batches), then renames the attributes (e.g. changing "p" to "price") for better readability. Finally, the output is converted back into JSON and published to the Kafka topic `financial_trades`, using the PublishKafka NiFi processor and a Kafka3ConnectionService.

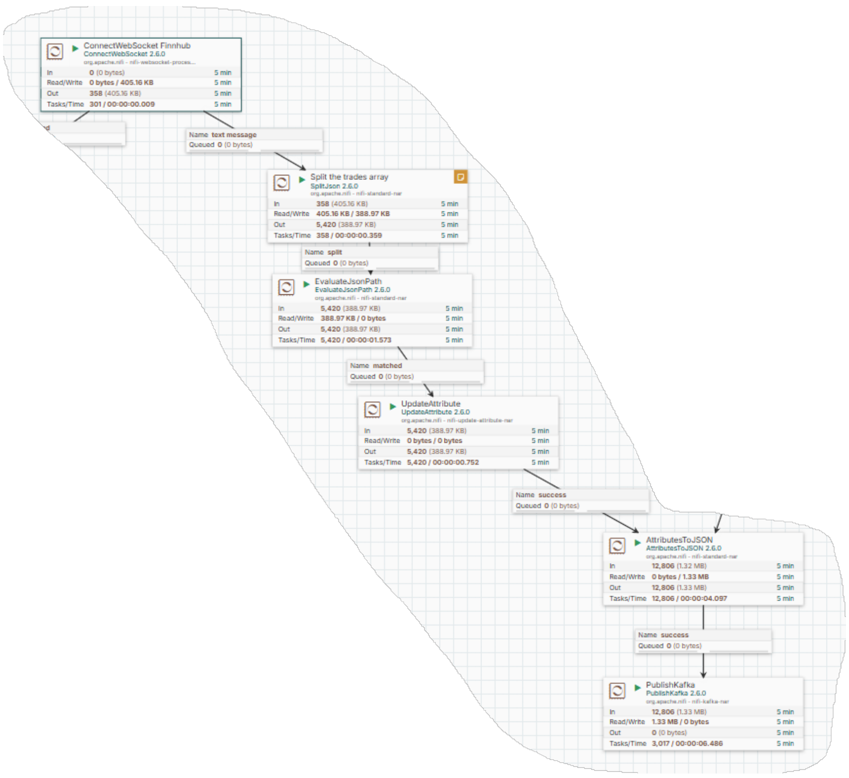

Figure 26: JSON Splitting Process

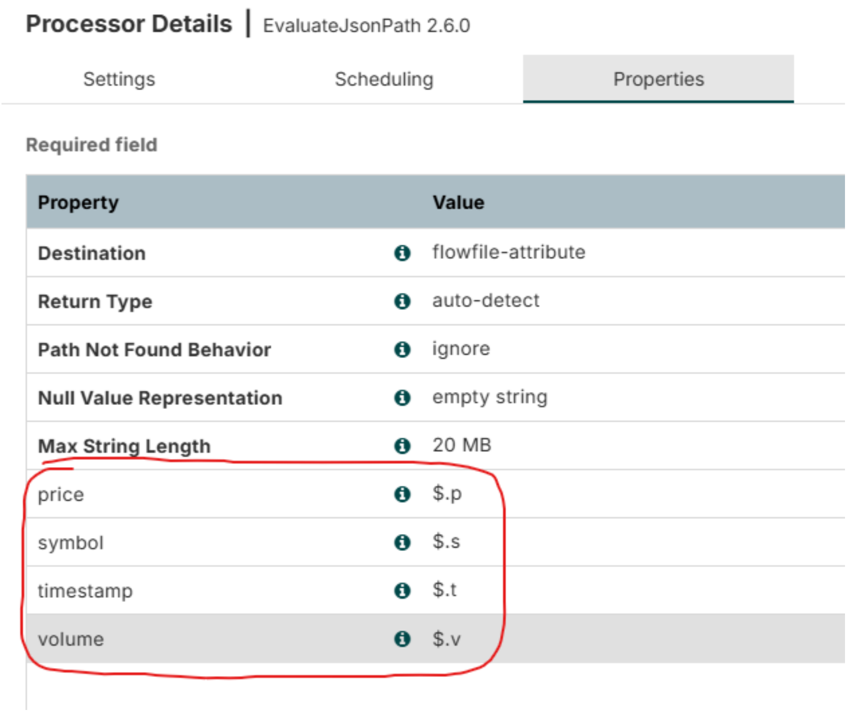

Figure 27: Attribute Renaming

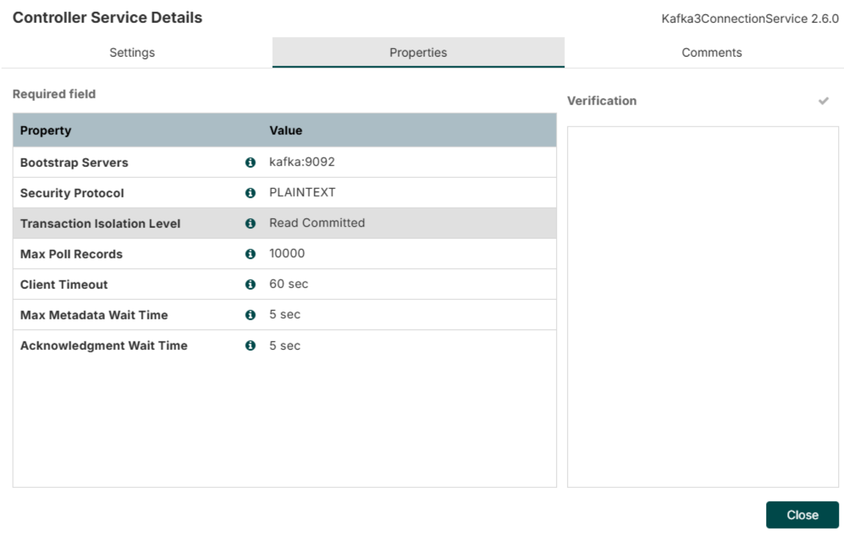

Figure 28: Kafka3ConnectionService Configuration

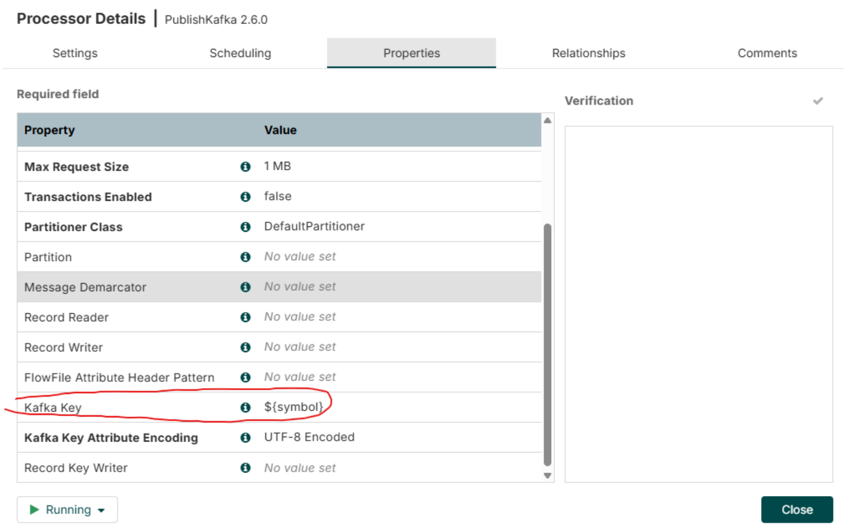

Figure 29: PublishKafka Processor: Using Symbol as Kafka Key

Figure 30: PublishKafka Processor: Setting Financial Trades Topic

Note: The same approach applies to the Binance API, except that it uses a different WebSocket endpoint and different message format.

3.1.2 News Data Ingestion

The news data ingestion flow is simpler, as it uses REST API calls every 60 seconds instead of websockets.

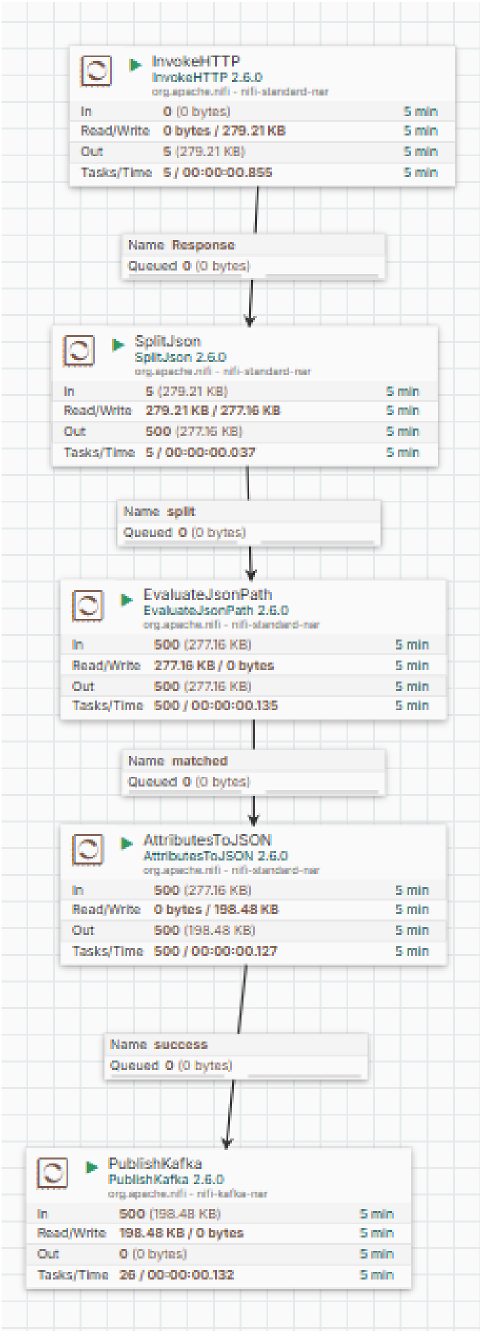

Figure 31: News Data Ingestion Flow

We use the InvokeHTTP NiFi processor to call the FinnHub news endpoint, setting the API key and other parameters in the URL.

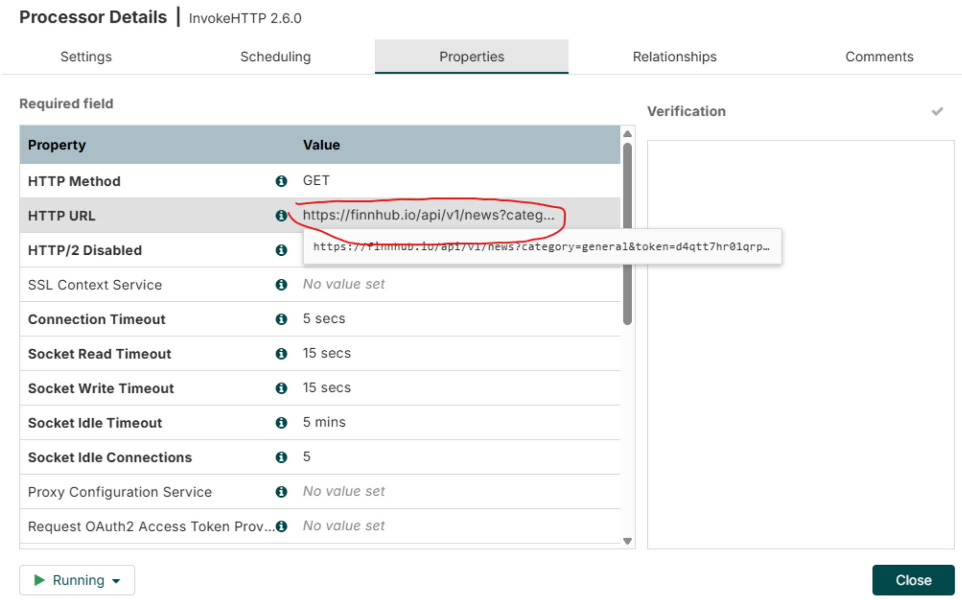

Figure 32: InvokeHTTP Configuration

The scheduler is set to run every 1 minute.

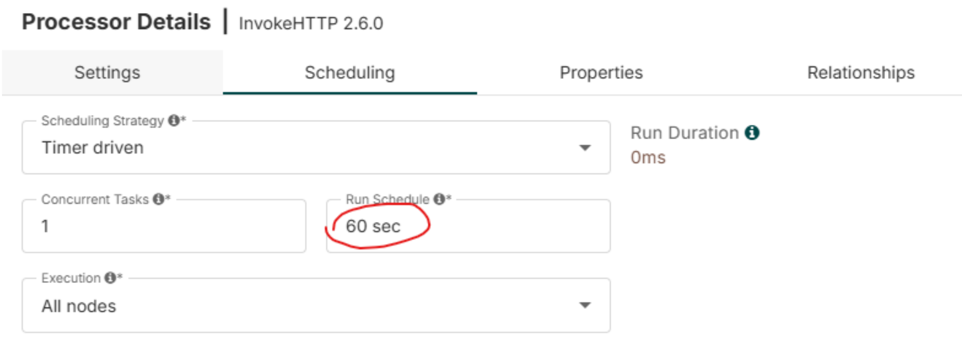

Figure 33: Scheduling Configuration

The SplitJson processor splits the returned JSON array of news articles into individual messages using the JsonPath expression $.

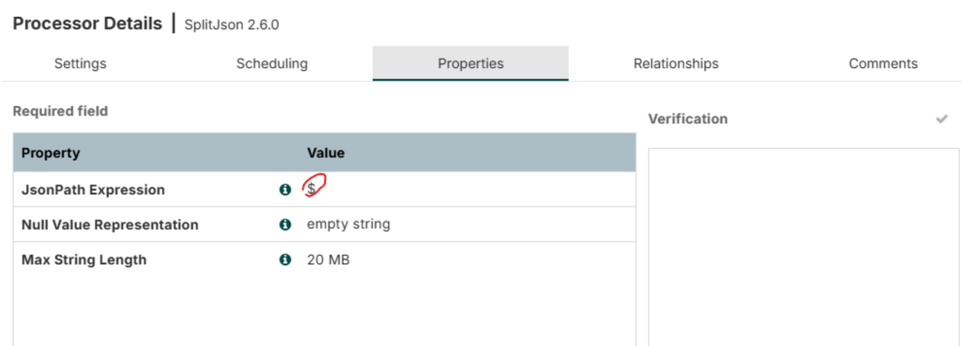

Figure 34: SplitJson Configuration

After that, we rename some attributes (e.g. url to article_url).

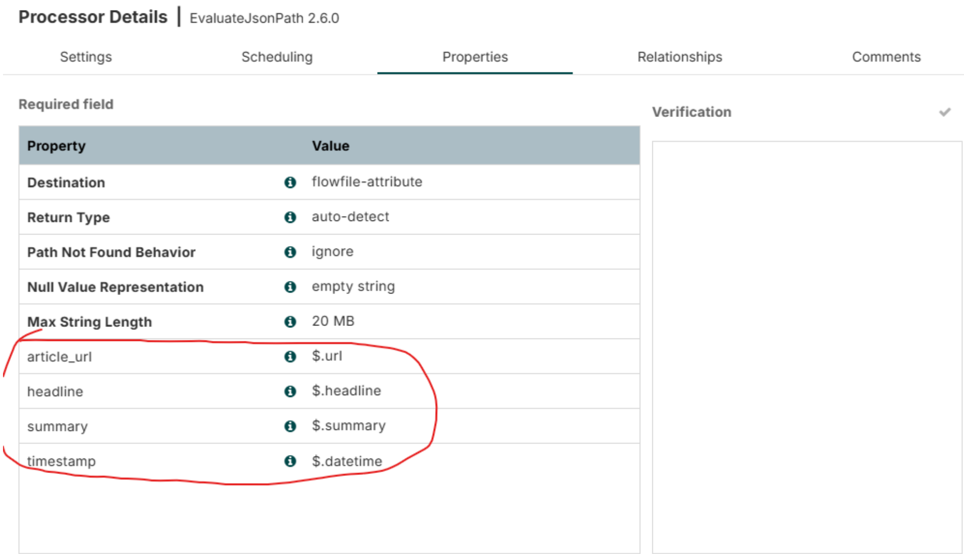

Figure 35: Attribute Renaming for News

Then, we set the schema and convert the data back to JSON format using the AttributesToJson processor.

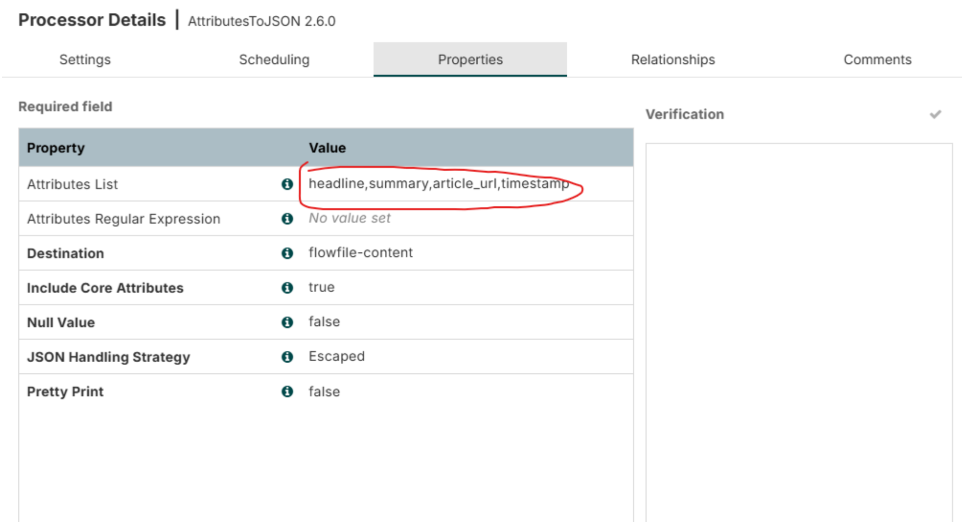

Figure 36: Converting Attributes to JSON

Finally, we publish the news data to the Kafka topic `financial_news` using the PublishKafka processor.

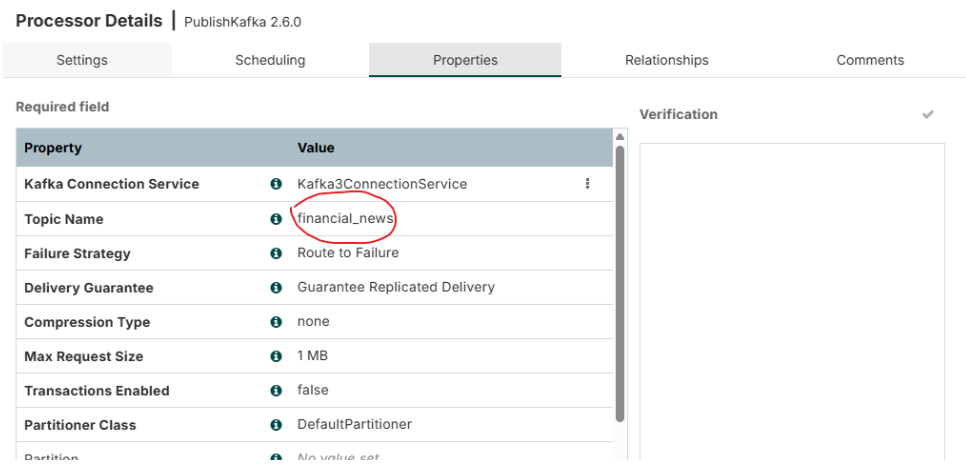

Figure 37: Publishing News to Kafka

3.2 Data Processing with Apache Spark Streaming

Spark Streaming consumes data from Kafka topics and performs real-time analytics. Two separate Spark jobs handle trades and news data independently.

3.2.1 Trades Analytics Job (`spark_trades.py`)

This job reads cryptocurrency trade data from the `financial_trades` Kafka topic, performs windowed aggregations, and writes results to Elasticsearch.

Key Steps:

1. **Read from Kafka**: Connect to the `financial_trades` topic with `startingOffsets: latest` (only processes new data)

2. **Parse JSON and Type Casting**: Parse the incoming JSON messages and cast fields to appropriate types:

```
df_parsed = df_kafka.select(
    from_json(col("value").cast("string"), schema).alias("data")
).select(
    col("data.symbol"),
    col("data.source"),
    col("data.price").cast(DoubleType()).alias("price"),
    col("data.volume").cast(DoubleType()).alias("volume"),
    (col("data.timestamp") / 1000).cast("timestamp").alias("
       event_time")
)
```

3. **Windowed Aggregation**: Group trades into 1-minute windows with a 10-second slide interval, calculating:

   - Average price per symbol
   - Total volume traded
   - Trade count

```
df_analytics = df_parsed \
    .withWatermark("event_time", "1 minute") \
    .groupBy(
        window(col("event_time"), "1 minute", "10 seconds"),
        col("symbol"),
        col("source")
    ) \
    .agg(
        avg("price").alias("avg_price"),
        sum("volume").alias("total_volume"),
        count("*").alias("trade_count")
    )
```

4. **Write to Elasticsearch**: Stream results to the `market_prices` index in append mode

Output Schema:

- `window_start`, `window_end`: Time boundaries of the aggregation window
- `symbol`: Cryptocurrency symbol (e.g., BTCUSDT, ETHUSDT)
- `source`: Data source (binance or finnhub)
- `avg_price`: Average trade price in the window
- `total_volume`: Total trading volume
- `trade_count`: Number of trades
- `processing_time`: Timestamp when Spark processed the data

3.2.2 News Sentiment Analysis Job (`spark_news.py`)

This job reads news articles from the `financial_news` Kafka topic, performs sentiment analysis using TextBlob, and writes results to Elasticsearch.

Key Steps:

1. **Read from Kafka**: Connect to the `financial_news` topic
2. **Parse JSON**: Extract headline, summary, article URL, and timestamp from incoming messages
3. **Sentiment Analysis with TextBlob**:
   - Create a User-Defined Function (UDF) that uses TextBlob for sentiment scoring
   - Analyze the article summary (or headline if summary is unavailable)
   - TextBlob returns a polarity score from -1.0 (very negative) to +1.0 (very positive)

```python
def analyze_sentiment_textblob(text):
    from textblob import TextBlob
    blob = TextBlob(text)
    sentiment_score = blob.sentiment.polarity
    return float(sentiment_score)

sentiment_score_udf = udf(analyze_sentiment_textblob, FloatType())
```

4. **Categorize Sentiment**: Convert the numerical score into categories:
   - **Positive**: score $\geq$ 0.15
   - **Negative**: score $\leq$ -0.15
   - **Neutral**: -0.15 $<$ score $<$ 0.15
5. **Write to Elasticsearch**: Stream results to the `news_sentiment` index

Output Schema:

- `headline`: News article headline
- `summary`: Article summary text
- `article_url`: Link to full article
- `news_timestamp`: When the article was published
- `sentiment_score`: Numerical sentiment score (-1.0 to 1.0)
- `sentiment`: Categorical sentiment (positive/negative/neutral)
- `processing_time`: When Spark processed the article

Note on TextBlob: TextBlob uses a pre-trained Naive Bayes classifier trained on movie reviews. It analyzes word patterns, adjectives, and contextual clues to determine sentiment polarity and subjectivity. The library is automatically installed on both Spark master and worker containers via the docker-compose configuration.

4 Conclusion

This project successfully demonstrates the implementation of a real-time big data pipeline for financial market analysis and sentiment detection. By integrating Apache NiFi, Kafka, Spark Streaming, Elasticsearch, and Kibana, we created a scalable system capable of processing continuous streams of cryptocurrency trades and financial news.

The platform showcases several key capabilities of modern big data technologies: real-time data ingestion from multiple sources, distributed stream processing, natural language processing for sentiment analysis, and interactive visualization. The modular architecture allows for easy extension to additional data sources or analytical capabilities.

Future enhancements could include machine learning models for price prediction, anomaly detection in trading patterns, more sophisticated sentiment analysis techniques, and integration with additional cryptocurrency exchanges and news sources.