



College of
Computing

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	3
2	Deployment and Configuration	3
2.1	Starting the Services	4
2.2	Importing the NiFi Workflow	4
2.2.1	Set Up NiFi Registry Client	4
2.2.2	Create a Bucket in NiFi Registry	5
2.2.3	Import the Workflow	5
2.2.4	Deploy the Workflow in NiFi	6
2.2.5	Enable and Start the Workflow	7
2.3	Launching the Spark Streaming Jobs	8
2.3.1	Start Both Streaming Jobs	8
2.3.2	Verify Jobs Are Running	9
2.4	Visualizing Data in Kibana	10
2.4.1	Open Kibana	10
2.4.2	Import the Dashboard	10
2.4.3	View the Dashboard	11
2.4.4	Configure Time Range and Auto-Refresh	11
2.4.5	Dashboard Overview	12
3	System Architecture and Implementation	14
3.1	Data Ingestion with Apache NiFi	15
3.1.1	Trade Data Ingestion	15
3.1.2	News Data Ingestion	18
3.2	Data Processing with Apache Spark Streaming	22
3.2.1	Trades Analytics Job (<code>spark_trades.py</code>)	22
3.2.2	News Sentiment Analysis Job (<code>spark_news.py</code>)	23
4	Conclusion	24

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

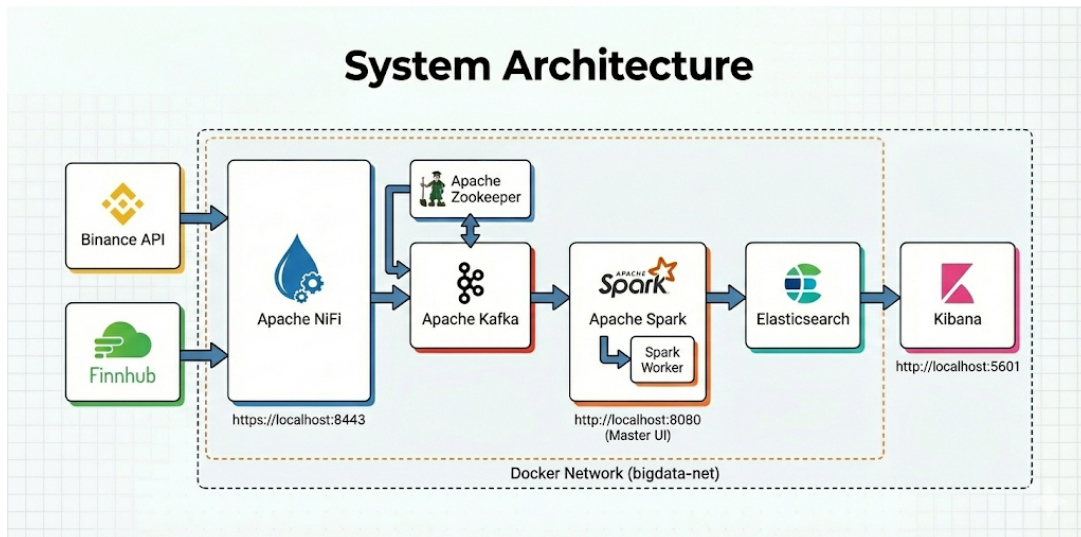


Figure 1: System Architecture Overview

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

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

2. Start Docker Containers

```
1 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.

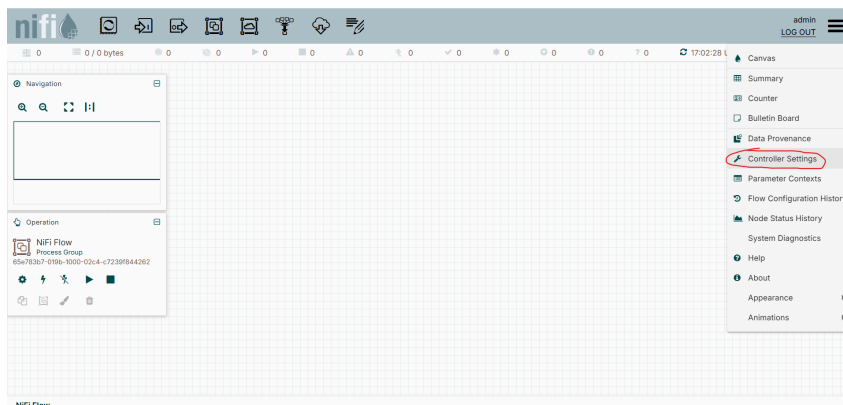


Figure 2: Creating NiFi Registry Client

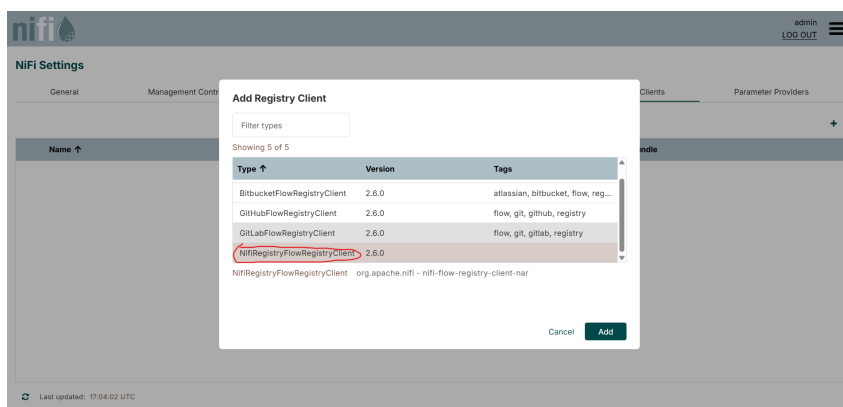


Figure 3: Registry Client Configuration

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

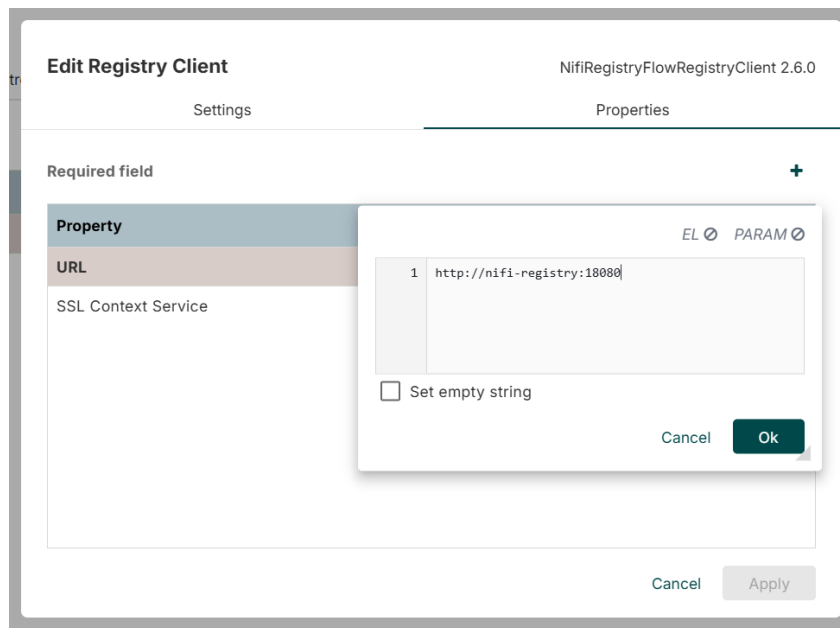


Figure 4: Setting Registry URL

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`.

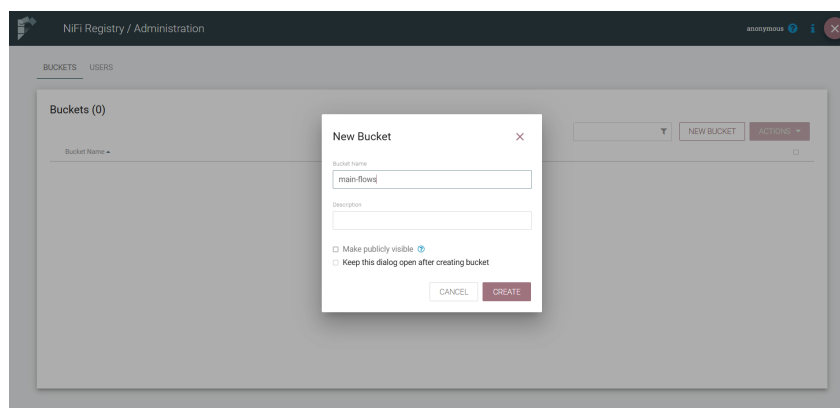


Figure 5: Creating Registry Bucket

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**.

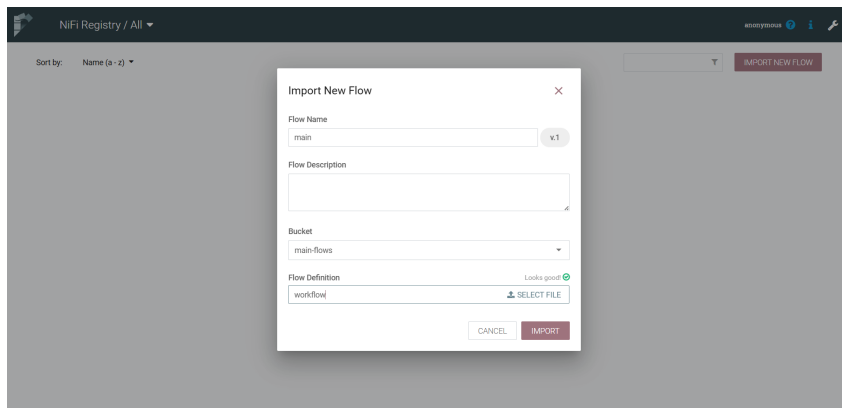


Figure 6: Importing Workflow from JSON

2.2.4 Deploy the Workflow in NiFi

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

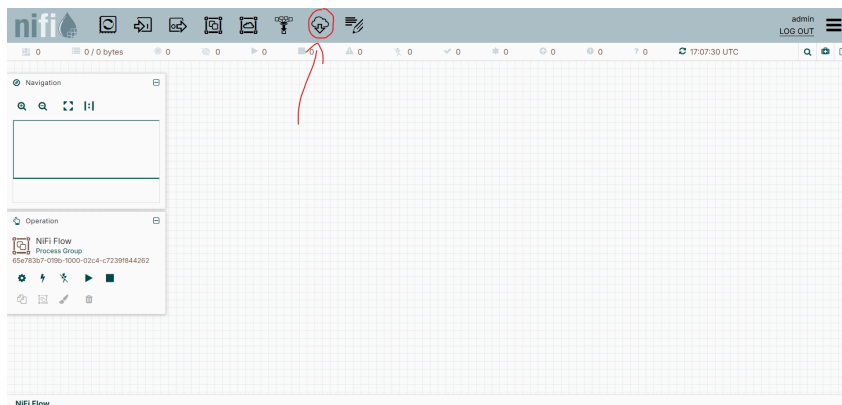


Figure 7: Import from Registry Icon

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

Import From Registry

Registry*
NifiRegistryFlowRegistryClient

Bucket*
main-flows

Flow*
main

☒ Keep existing Parameter Contexts ⓘ

Flow Description
No description provided

Version	Created ↓	Comments
1	12/22/2025 17:07:17	

Cancel 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**.

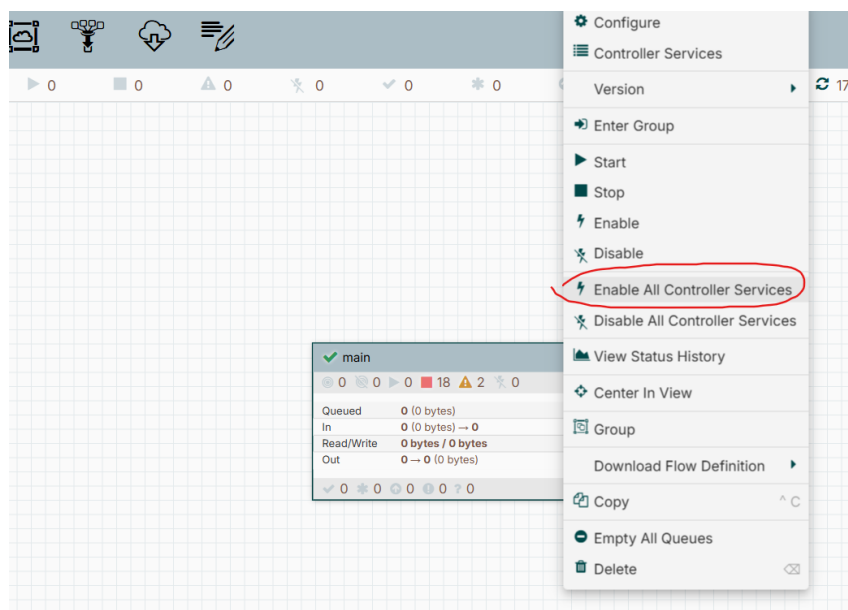


Figure 9: Enabling Controller Services

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

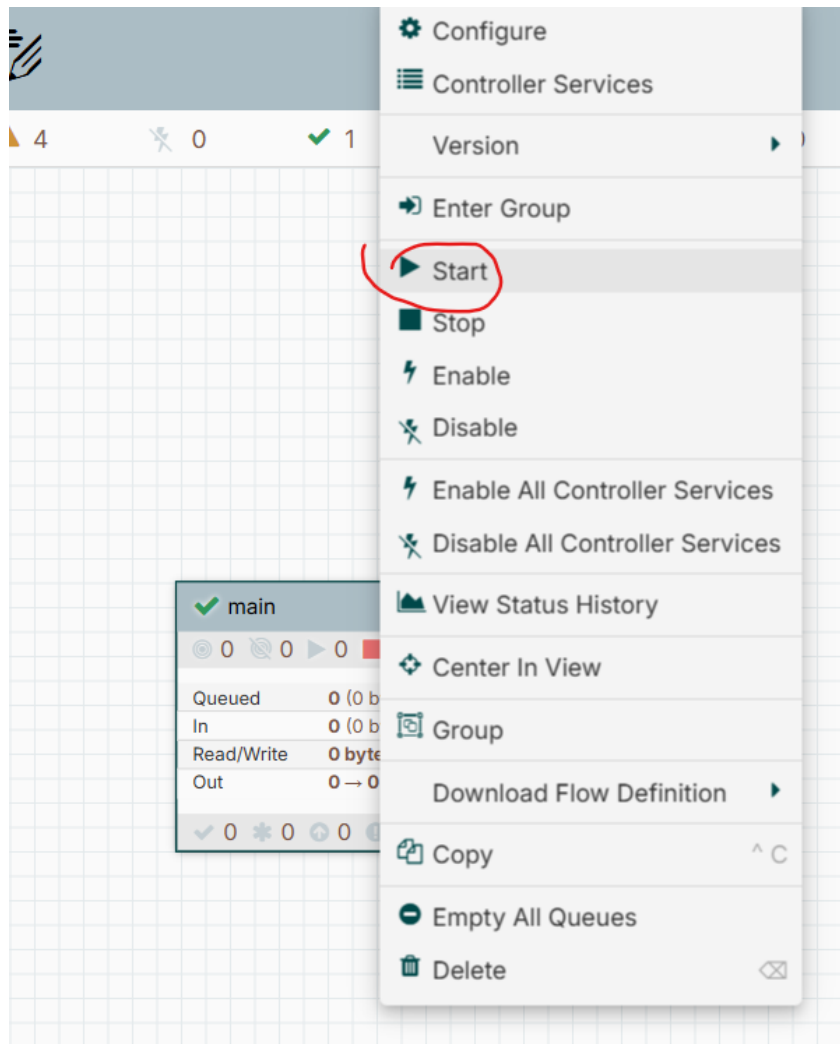


Figure 10: Starting the Workflow

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:

```
1 ./start_processing.sh
```

On Windows:

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

```
PS C:\Users\mycro\Desktop\market-data-streaming-viz> bash ./start_processing.sh
■ Creating Elasticsearch index templates...
■ Index templates created.
■ Submitting TRADES Job (limited to 1 core)...
■ Submitting NEWS Job (limited to 1 core)...
■ Jobs submitted in background.

Monitor with:
- Spark UI: http://localhost:8080
- Spark logs: docker logs spark-master -f
- Check data: curl http://localhost:9200/market_prices/_count 56 curl http://localhost:9200/news_sentiment/_count
PS C:\Users\mycro\Desktop\market-data-streaming-viz>
```

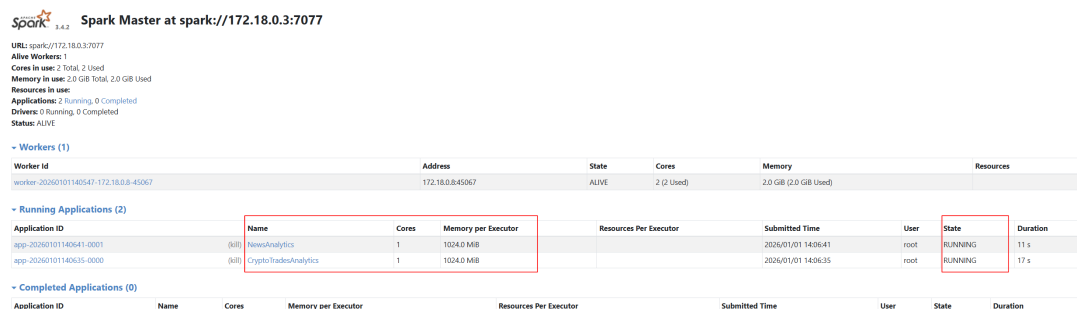
Figure 11: Executing Spark Job Launch Script

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:



The screenshot shows the Spark Master at spark://172.18.0.3:7077. It displays the status of workers and running applications. Two applications are running: 'NewsAnalytics' and 'CryptoTradesAnalytics', both with a state of 'RUNNING'.

Worker ID	Address	State	Cores	Memory	Resources
worker-20260701140547-172.18.0.8-45067	172.18.0.8:45067	ALIVE	2 (2 Used)	2.0 GB (2.0 GB Used)	

Application ID	Name	Cores	Memory per Executor	Resources Per Executor	Submitted Time	User	State	Duration
app-20260701140641-0001	NewsAnalytics	1	1024.0 MB		2026/01/01 14:06:41	root	RUNNING	11 s
app-20260701140635-0000	CryptoTradesAnalytics	1	1024.0 MB		2026/01/01 14:06:35	root	RUNNING	17 s

Figure 12: Spark Web UI Showing Running Jobs

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

```
< > localhost:9200/market_prices/_count
Pretty-print
{"count":72,"_shards":{"total":1,"successful":1,"skipped":0,"failed":0}}
```

Figure 13: Verifying Market Prices Index

```
< > localhost:9200/news_sentiment/_count
Pretty-print
{"count":300,"_shards":{"total":1,"successful":1,"skipped":0,"failed":0}}
```

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.

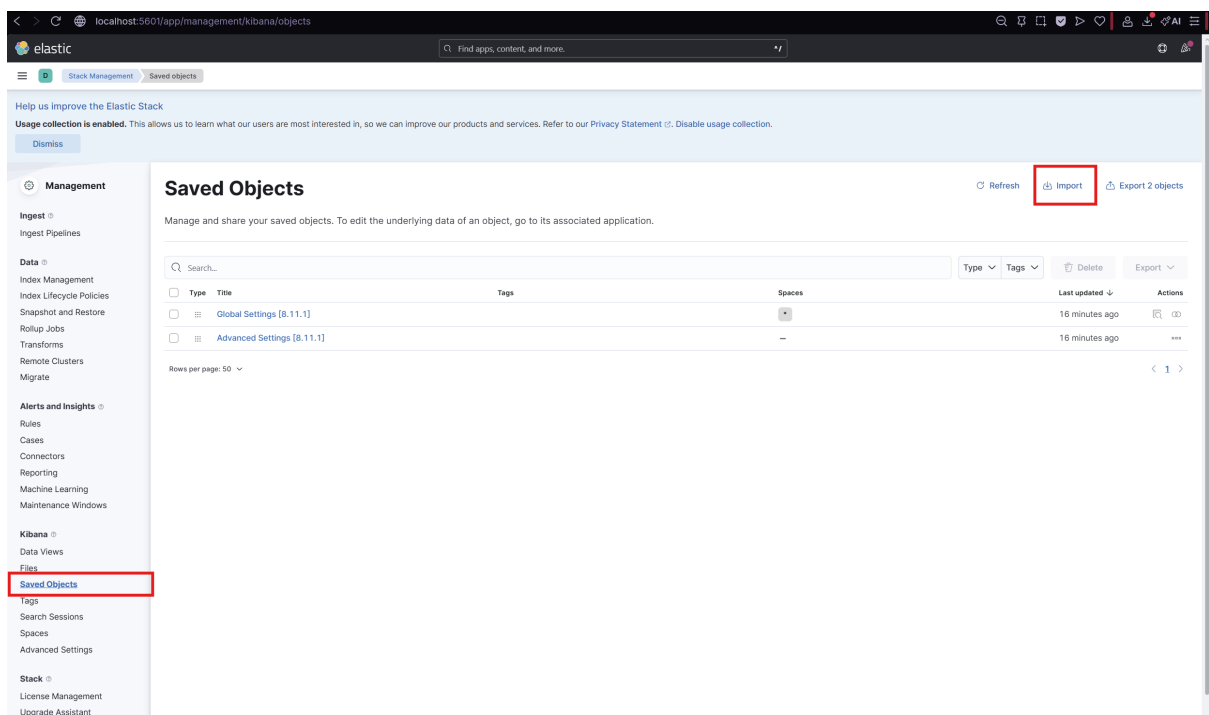


Figure 15: Kibana Import Dialog

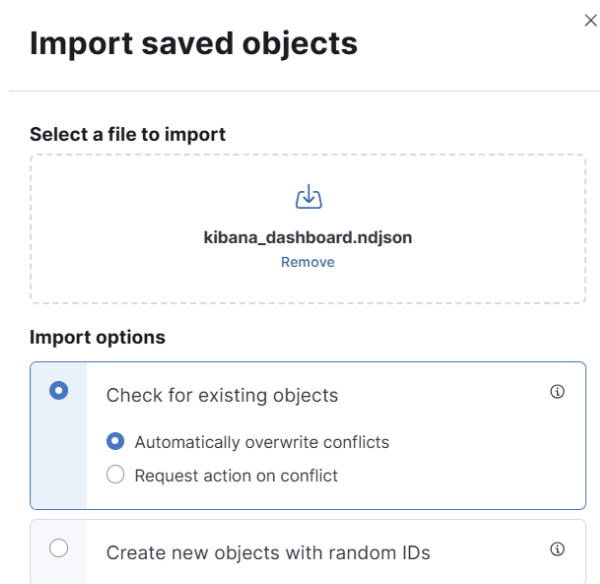


Figure 16: Import Dialog

2.4.3 View the Dashboard

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

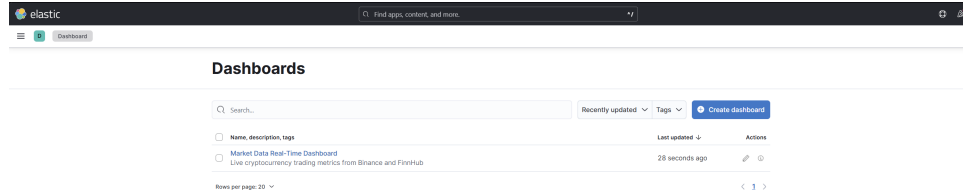


Figure 17: Dashboard Selection Screen

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).

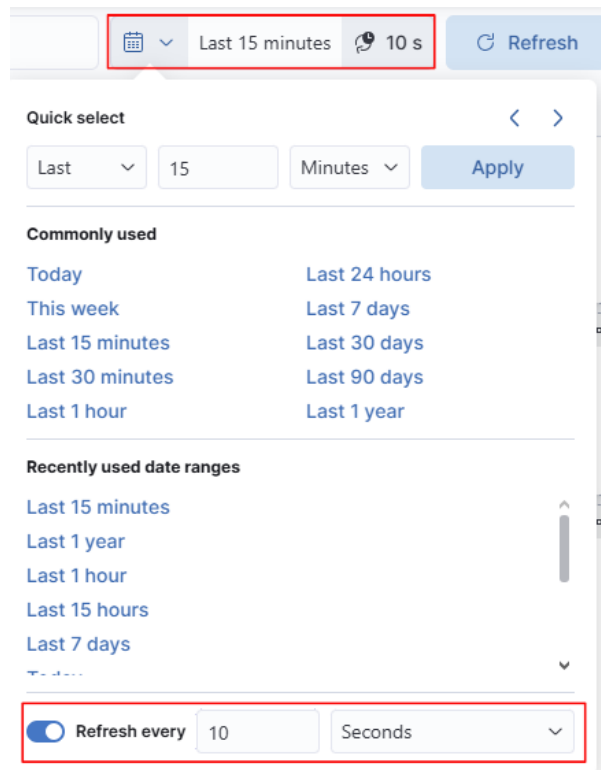


Figure 18: Time Range and Auto-Refresh Configuration

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

2.4.5 Dashboard Overview

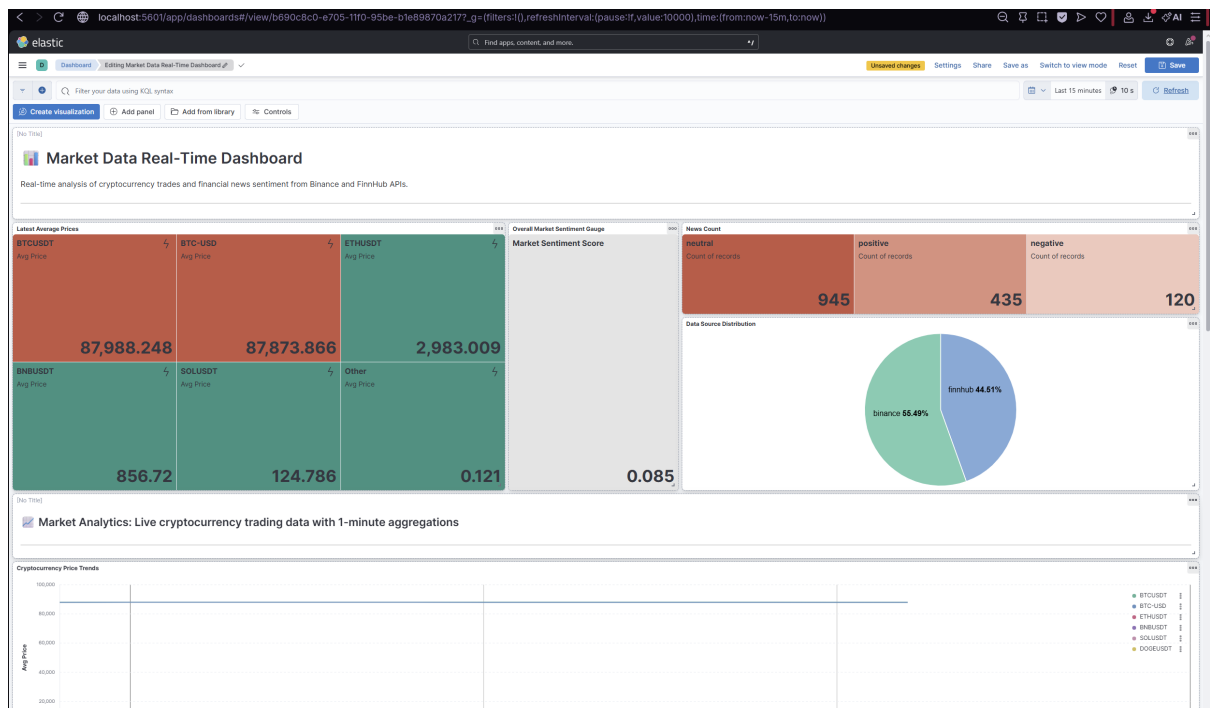


Figure 19: Complete Dashboard Overview

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)

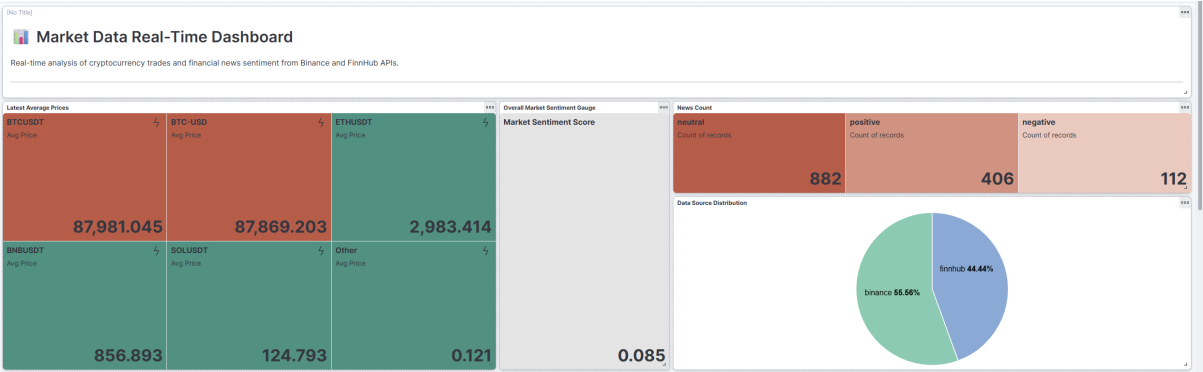


Figure 20: Key Metrics Section

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

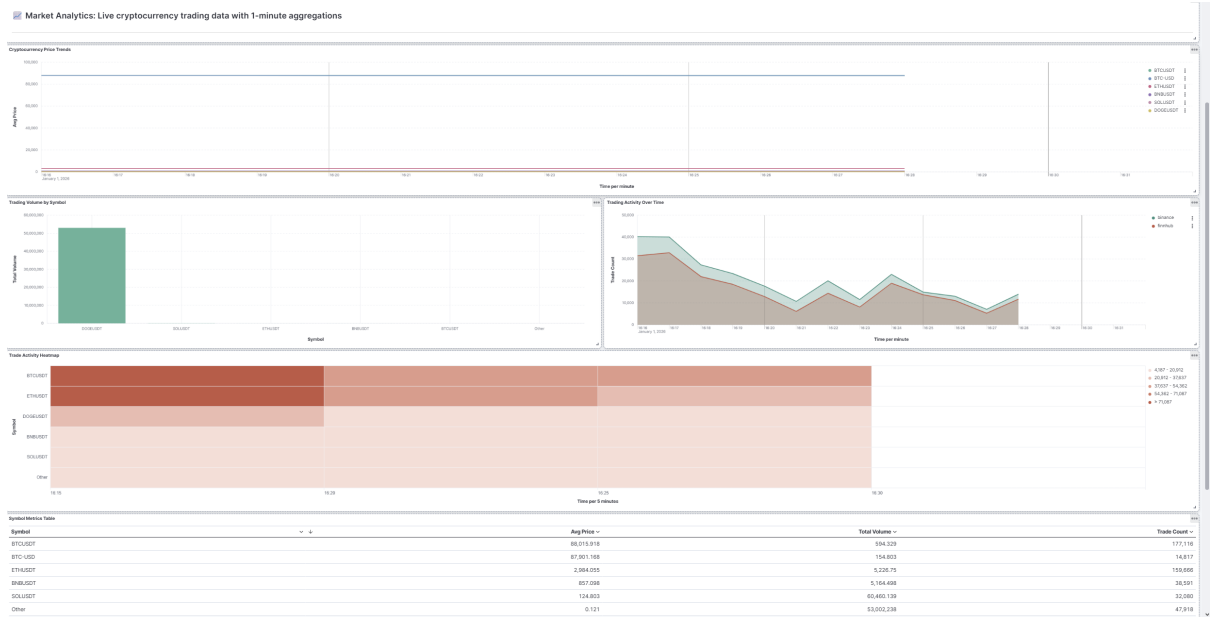


Figure 21: Market Analytics Section

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.

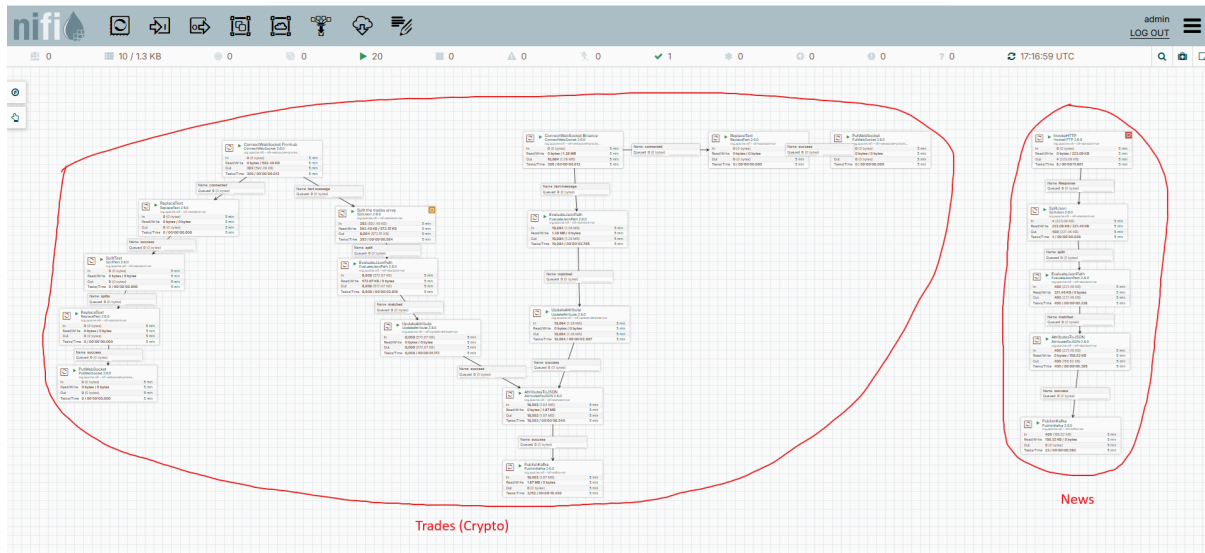


Figure 23: Complete NiFi Workflow

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.

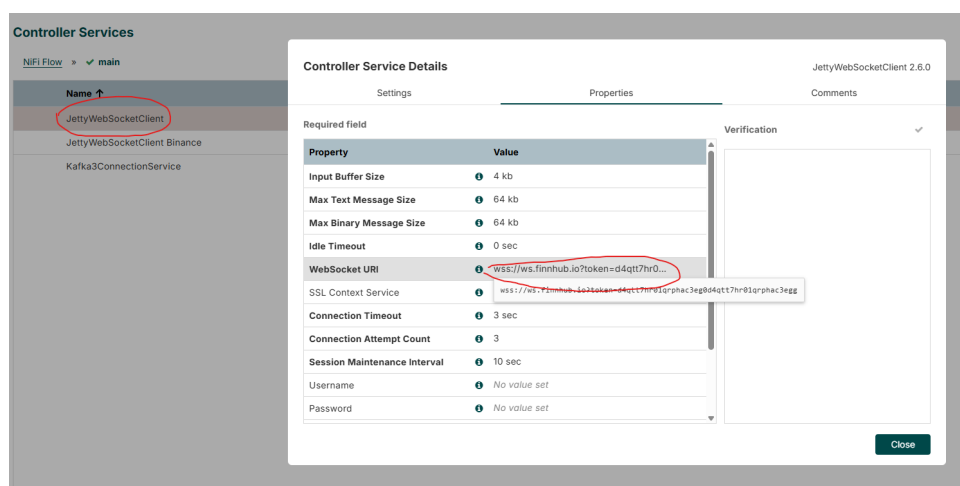


Figure 24: WebSocket Configuration

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

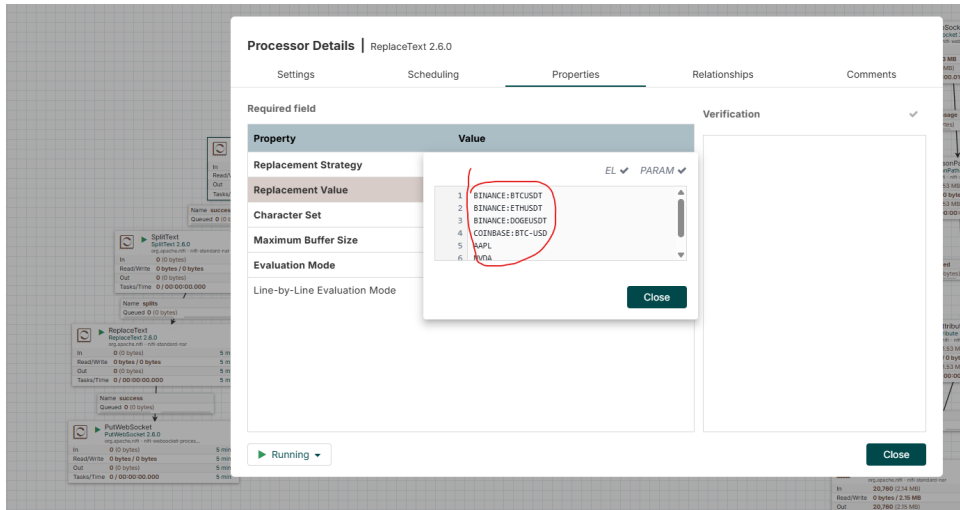


Figure 25: Subscription Message Configuration

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.

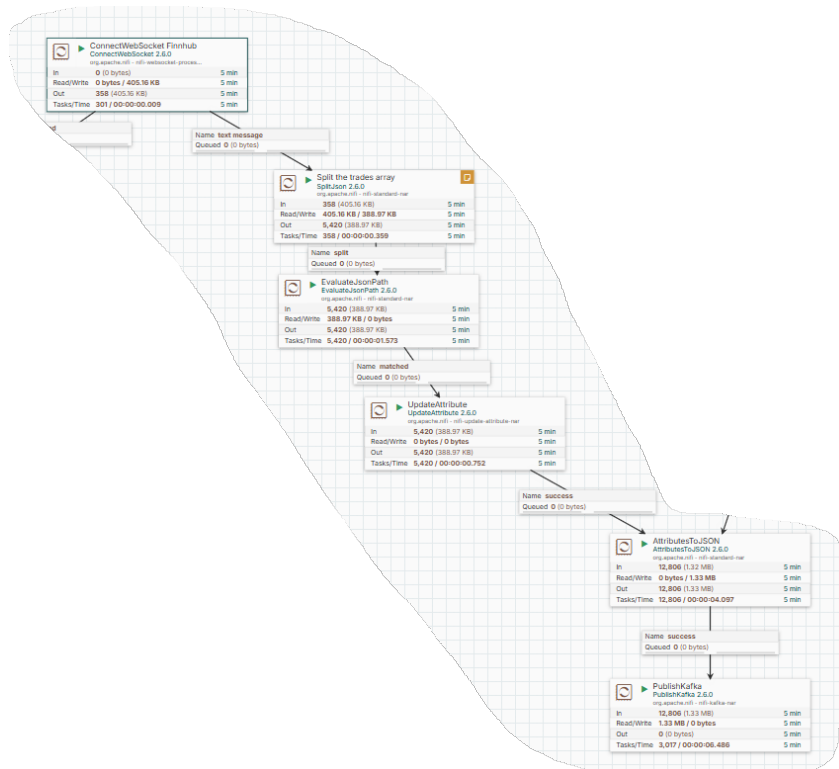


Figure 26: JSON Splitting Process

Processor Details | EvaluateJsonPath 2.6.0

Settings

Scheduling

Properties

Required field










Property	Value
Destination	 flowfile-attribute
Return Type	 auto-detect
Path Not Found Behavior	 ignore
Null Value Representation	 empty string
Max String Length	 20 MB
price	 \$.p
symbol	 \$.s
timestamp	 \$.t
volume	 \$.v

Figure 27: Attribute Renaming

Controller Service Details








Kafka3ConnectionService 2.6.0

Settings

Properties

Comments

Required field

Property	Value
Bootstrap Servers	 kafka:9092
Security Protocol	 PLAINTEXT
Transaction Isolation Level	 Read Committed
Max Poll Records	 10000
Client Timeout	 60 sec
Max Metadata Wait Time	 5 sec
Acknowledgment Wait Time	 5 sec

Verification

✓

Close

Figure 28: Kafka3ConnectionService Configuration

Processor Details | PublishKafka 2.6.0

Settings Scheduling **Properties** Relationships Comments

Required field

Property	Value
Max Request Size	1 MB
Transactions Enabled	false
Partitioner Class	DefaultPartitioner
Partition	No value set
Message Demarcator	No value set
Record Reader	No value set
Record Writer	No value set
FlowFile Attribute Header Pattern	No value set
Kafka Key	\$(symbol)
Kafka Key Attribute Encoding	UTF-8 Encoded
Record Key Writer	No value set

Verification ✓

▶ Running ▼ **Close**

Figure 29: PublishKafka Processor: Using Symbol as Kafka Key

Topic Name **financial_trades**

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.

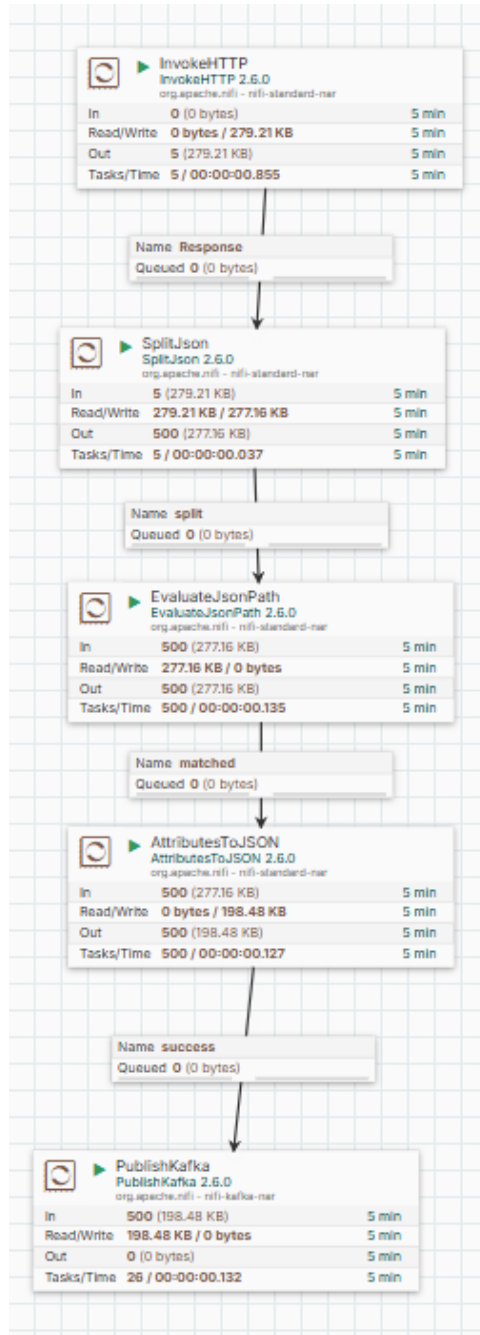


Figure 31: News Data Ingestion Flow

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

Processor Details | InvokeHTTP 2.6.0

Settings Scheduling **Properties** Relationships Comments

Required field

Property	Value
HTTP Method	GET
HTTP URL	https://finnhub.io/api/v1/news?categ...
HTTP/2 Disabled	https://finnhub.io/api/v1/news?category=general&token=d4qtt7hr0lqrp...
SSL Context Service	No value set
Connection Timeout	5 secs
Socket Read Timeout	15 secs
Socket Write Timeout	15 secs
Socket Idle Timeout	5 mins
Socket Idle Connections	5
Proxy Configuration Service	No value set
Request OAuth2 Access Token Prov...	No value set

Verification ✓

Running ▼ Close

Figure 32: InvokeHTTP Configuration

The scheduler is set to run every 1 minute.

Processor Details | InvokeHTTP 2.6.0

Settings **Scheduling** Properties Relationships

Scheduling Strategy ⓘ*
Timer driven ▼

Run Duration ⓘ
0ms

Concurrent Tasks ⓘ*
1

Run Schedule ⓘ*
60 sec

Execution ⓘ*
All nodes ▼

Figure 33: Scheduling Configuration

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

Processor Details | SplitJson 2.6.0

Settings Scheduling **Properties** Relationships Comments

Required field

Property	Value
JsonPath Expression	\$
Null Value Representation	empty string
Max String Length	20 MB

Verification ✓

Figure 34: SplitJson Configuration

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

Processor Details | EvaluateJsonPath 2.6.0

Settings Scheduling **Properties** Relationships Comments

Required field		Verification
Property	Value	
Destination	flowfile-attribute	
Return Type	auto-detect	
Path Not Found Behavior	ignore	
Null Value Representation	empty string	
Max String Length	20 MB	
article_url	\$.url	
headline	\$.headline	
summary	\$.summary	
timestamp	\$.datetime	

Figure 35: Attribute Renaming for News

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

Processor Details | AttributesToJson 2.6.0

Settings Scheduling **Properties** Relationships Comments

Required field		Verification
Property	Value	
Attributes List	headline,summary,article_url,timestamp	
Attributes Regular Expression	No value set	
Destination	flowfile-content	
Include Core Attributes	true	
Null Value	false	
JSON Handling Strategy	Escaped	
Pretty Print	false	

Figure 36: Converting Attributes to JSON

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

Processor Details | PublishKafka 2.6.0

Settings | Scheduling | **Properties** | Relationships | Comments

Required field

Property	Value
Kafka Connection Service	Kafka3ConnectionService
Topic Name	financial_news
Failure Strategy	Route to Failure
Delivery Guarantee	Guarantee Replicated Delivery
Compression Type	none
Max Request Size	1 MB
Transactions Enabled	false
Partitioner Class	DefaultPartitioner
Partition	No unlive cat

Verification

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:

```
1 df_parsed = df_kafka.select(
2     from_json(col("value").cast("string"), schema).alias("data")
3 ).select(
4     col("data.symbol"),
5     col("data.source"),
6     col("data.price").cast(DoubleType()).alias("price"),
7     col("data.volume").cast(DoubleType()).alias("volume"),
8     (col("data.timestamp") / 1000).cast("timestamp").alias("
9         event_time")
10 )
```

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

```

1 df_analytics = df_parsed \
2   .withWatermark("event_time", "1 minute") \
3   .groupBy(
4     window(col("event_time"), "1 minute", "10 seconds"),
5     col("symbol"),
6     col("source")
7   ) \
8   .agg(
9     avg("price").alias("avg_price"),
10    sum("volume").alias("total_volume"),
11    count("*").alias("trade_count")
12  )

```

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)


```

1 def analyze_sentiment_textblob(text):
2     from textblob import TextBlob
3     blob = TextBlob(text)
4     sentiment_score = blob.sentiment.polarity
5     return float(sentiment_score)
6
7 sentiment_score_udf = udf(analyze_sentiment_textblob, FloatType())

```

4. **Categorize Sentiment:** Convert the numerical score into categories:

- **Positive:** $\text{score} \geq 0.15$
- **Negative:** $\text{score} \leq -0.15$
- **Neutral:** $-0.15 < \text{score} < 0.15$

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.