Checkpoint 1 – Big Data Project

"Data & Service Brief" Report

Anas Khalil Mourad Mahmoudi

November 19, 2025

Project Context

As part of the Big Data module, we must design and implement an end-to-end data pipeline based on the following technologies: Apache Kafka for continuous data ingestion, HDFS for raw data storage, Apache Spark (Batch and Structured Streaming) for processing, a time-series database (InfluxDB or TimescaleDB) for storing aggregated metrics, and Grafana for real-time visualization.

We are working in pairs, using a cluster composed of four nodes (two virtual machines per student).

1 Data Sources and Application Domain

1.1 Target Domain

We selected a simple, accessible, yet highly relevant topic: **real-time monitoring of the cryptocurrency market (Bitcoin, Ethereum, etc.)**.

Crypto-asset prices are volatile, updated in real time, and provide an ideal use case for a Big Data streaming pipeline.

The objective is to build a platform that allows:

- real-time tracking of Bitcoin, Ethereum, and other major crypto prices;
- instant visualization of short-term variations (1 min, 5 min, 1h);
- monitoring of global trading volume;
- detection of sudden movements ("price jumps", "crashes").

1.2 Selected Data Streams

The data comes from two simple, public APIs:

- **CoinGecko API** (free, no API key required). Example endpoint: `https://api.coingecko.com/api/v3/simple/price?ids=bitcoin,ethereum&vs_currencies=usd&include_24hr_vol=true`
  - Provides price in USD/EUR;
  - 24h trading volume;
  - Variations over 1h/24h.

- **Binance Public API** (real time, no key). Example: `/api/v3/ticker/24hr?symbol=BTCUSDT`
  - Current price;
  - High/low values over the last 24h;
  - 24h volume.

A Kafka producer queries the APIs every 5 seconds and publishes the results into separate topics (one per source).

These APIs are extremely easy to use, fast, free, and without strict rate limits.

1.3 Logical Schema of Messages

We standardize all Kafka messages in JSON format, split into two topics:

- **Topic** `crypto_coingecko` :
  - `symbol`: BTC, ETH, ...
  - `price`: price in USD;
  - `volume_24h`;
  - `change_1h`;
  - `event_time`.
- **Topic** `crypto_binance` :
  - `symbol`;
  - `price`;
  - `high_24h`, `low_24h`;
  - `volume_24h`;
  - `event_time`.

The `event_time` field is used by Spark Structured Streaming as the event timestamp for windowing operations.

2 Problem Statement

The business problem addressed is:

> *How can we efficiently monitor, in near-real-time, the price and volume evolution of major cryptocurrencies in order to detect sudden movements and analyze market trends?*

Such a service is commonly used in:

- cryptocurrency analytics platforms;
- trading algorithms;
- price alert applications.

3 Metrics and KPIs

3.1 Price-based Indicators

- **Real-time price** ;
- **Price change over 1 min, 5 min, 1h, 24h** ;
- **Volatility**, computed using sliding Spark windows ;
- **Price jump detection** based on threshold variations.

3.2 Volume-based Indicators

- **Total 24h volume** ;
- **Volume variation over time** ;
- **Volume anomalies** detected via standard deviation.

3.3 Technical KPIs

- messages per minute on Kafka topics;
- Spark → InfluxDB end-to-end latency;
- daily HDFS storage volume;
- average micro-batch processing time.

4 Implementation Plan

1. **Kafka Producer**: Python script (requests + json) querying CoinGecko and Binance every 5 seconds.
2. **HDFS Storage**: Spark Streaming reads the topics and writes raw data to HDFS partitioned by date.
3. **Spark Processing**:
   - computing price variations;
   - computing volatility;
   - detecting price jumps;
   - writing metrics into InfluxDB.
4. **Grafana**: dashboards showing:
   - real-time prices;
   - consolidated indicators;
   - event detection (spikes, crashes).

Conclusion

For this first checkpoint, we have:

- selected a simple yet relevant topic: crypto market monitoring;
- identified the CoinGecko and Binance data sources;
- defined the JSON message schemas and Kafka topics;
- specified both business KPIs (price, volatility) and technical KPIs;
- outlined the overall architecture.

Next steps include: implementing the Kafka producers, ingesting raw data into HDFS, developing the first Spark jobs, and building the Grafana dashboards.