Real-Time Market Volatility and Trade Activity Tracking using Kafka, PySpark, and Amazon Redshift

Prepared by: Michael Hein

Student ID: 300375535

CSIS 4495 – 003, Applied Research Project

Instructor: Padmapriya Arasanipalai Kandhadai

Contents

Real-Time Market Volatility and Trade Activity Tracking using Kafka, PySpark, and Amazon Redshift 1

1.Introduction 2

2. Research Hypotheses and Assumptions 2

3.Problem Statement 3

4. Proposed Research Project 3

4.1 Research Design and Methodology Justification 3

4.2 Data Collection and Analysis Techniques 5

4.3 Technology Stack 5

4.4 Expected Results and Practical Applications 6

5. Project Planning and Timelines 6

Gantt Chart 7

Risk Mitigation Strategy 7

Project Contract 8

6. AI Use Section 9

AI Usage Summary 9

7. Work Log Table 10

8. References 10

1.Introduction

The cryptocurrency market operates continuously and exhibits extreme volatility, generating large volumes of trade-related events every second. Unlike traditional financial markets with defined trading hours, cryptocurrency exchanges operate 24/7, requiring real-time monitoring systems capable of ingesting and analyzing streaming data with minimal latency. This research project is situated within the domain of Event-Driven Architectures (EDA), where data is processed as continuous streams of events rather than static batches.

Background and Context: Conventional data warehousing and Extract-Transform-Load (ETL) pipelines rely heavily on batch processing, introducing delays that can range from minutes to hours. In highly volatile markets such as cryptocurrencies, these delays severely limit the effectiveness of risk monitoring, operational analytics, and anomaly detection. When markets experience rapid price movements—commonly known as "flash crashes"—batch-oriented systems cannot provide timely alerts or insights.

This project addresses this limitation by designing and implementing a real-time data pipeline that ingests live cryptocurrency trade-related events from the **Coinbase WebSocket API**, processes them using **Apache Kafka** and **PySpark Structured Streaming**, and stores analytical results in **Amazon Redshift** for near real-time querying and visualization.

Literature Review and Existing Gaps

Industry Context: Real-time financial data pipelines are well-established in institutional trading environments (Reis & Housley, 2022). However, most published implementations focus on proprietary systems with limited technical disclosure. Open-source implementations typically demonstrate basic streaming concepts without domain-specific optimizations for financial time-series data.

Existing Solutions: Traditional ETL tools (Informatica, Talend) operate on batch schedules. Generic Kafka-Spark tutorials lack financial domain expertise. Cryptocurrency exchange APIs provide data but not analytical frameworks.

Research Gap: This project addresses the gap between tutorial-level streaming demonstrations and production-grade cryptocurrency analytics by implementing:

- Domain Specific windowed aggregations (VWAP, volatility measures)
- Handling of out-of-order WebSocket events with watermarking
- Optimized columnar storage patterns for time-series queries

2. Research Hypotheses and Assumptions

Hypothesis 1 – Kafka Buffering:

A properly configured Kafka cluster can absorb sudden spikes in cryptocurrency trading volume (up to 10x normal load) without message loss, maintaining sub-second ingestion latency.

Hypothesis 2 – Stream Processing Accuracy:

PySpark Structured Streaming with a 10-minute watermark threshold can achieve >99% accuracy in windowed aggregations despite out-of-order event delivery common in WebSocket feeds.

Hypothesis 3 – Storage Efficiency:

Amazon Redshift's columnar storage, when properly partitioned by date and symbol, can execute analytical queries on millions of trade records with sub-second response times suitable for operational dashboards.

Assumption 1 – Data Availability:

The Coinbase WebSocket API will provide sufficient event frequency (100-1,000 events per minute for BTC-USD and ETH-USD) to demonstrate real-time streaming capabilities on a single development machine.

Assumption 2 – Technology Accessibility:

Open-source technologies (Kafka, Spark, Docker) can deliver production-quality streaming analytics without requiring expensive enterprise software licenses or cloud infrastructure beyond basic S3 storage.

Assumption 3 – Latency Tolerance:

A 10-minute watermark for late-arriving events provides an acceptable balance between processing completeness (capturing >99% of events) and end-to-end latency (<30 seconds for 95th percentile).

Assumption 4 – Development Environment:

A single macOS development machine with Docker can adequately simulate a distributed streaming architecture for demonstration and educational purposes.

3.Problem Statement

The Core Challenge

The primary challenge addressed by this project is insight latency in traditional batch-based analytics systems. Specifically, how can a scalable system ingest, process, and analyze high-frequency cryptocurrency trade events in near real time without data loss or significant delay?

Research Questions

- How can Apache Kafka act as a fault-tolerant buffer to absorb sudden spikes in market activity during periods of high volatility?
- How effectively can PySpark Structured Streaming perform real-time, windowed aggregations on out-of-order streaming data originating from WebSocket feeds?
- How can Amazon Redshift be used to store and query high-velocity time-series data efficiently for downstream monitoring and alerting?

Addressing these questions is essential for building modern real-time monitoring systems capable of detecting abnormal market behavior, such as sudden price swings or volume spikes**.**

4. Proposed Research Project

4.1 Research Design and Methodology Justification

This project adopts an **Event-Driven Architecture (EDA)** optimized for high-velocity financial data streams. The system follows a Kafka-based streaming architecture, where all incoming data is treated as a continuous flow of events rather than discrete batches.

Apache Kafka is used as a distributed, fault-tolerant messaging system that decouples data ingestion from downstream processing and prevents data loss during traffic spikes. **PySpark Structured Streaming** is selected for its ability to perform scalable, stateful stream processing, including windowed aggregations and watermarking for late-arriving events. **Amazon Redshift** serves as the analytical data warehouse, leveraging its massively parallel processing (MPP) capabilities for efficient querying of aggregated time-series data. This architecture reflects industry-standard practices used in real-time financial analytics systems.

Justification:

Kafka over RabbitMQ:

- Higher throughput (thousands of messages/second per partition vs. hundreds)

- Log-based storage enables replay and reprocessing of historical events

- Better horizontal scalability for high-volume financial data streams

- Native integration with Spark Structured Streaming

Kafka over Apache Pulsar:

- More mature ecosystem with extensive documentation

- Wider industry adoption (better for resume/portfolio)

- Simpler operational model for single-node development

Pyspark Structured Streaming vs. Alternatives

Pyspark over Apache Flick:

- Lower learning curve with DataFrame API (familiar to data analysts)

- Better documentation and community resources for beginners

- Sufficient for project requirements (sub-second latency acceptable)

- Flink's ultra-low-latency advantages (milliseconds) not required for this use case

Pyspark over Apache Storm:

- Higher-level abstraction reduces implementation complexity

- Built-in support for event-time semantics and watermarking

- SQL-like interface for analytical transformations

- Better fault tolerance with exactly once semantics

Pyspark over AWS Kinesis Analytics:

- More control over processing logic

- Better educational value (understanding stream processing fundamentals)

- Portable skills applicable to on-premise or multi-cloud environments

Amazon Redshift vs. Alternatives

Redshift over snowflake:

- Native integration with AWS S3 via COPY command

- Lower cost for small academic workloads with Serverless pricing

- Documented best practices for time-series data in financial services

- Sufficient query performance for operational analytics

Redshift over Google BigQuery:

- Simpler cost model for students (pay-per-query vs. on-demand scanning)

- Better alignment with AWS ecosystem (S3 staging)

- More control over cluster configuration and query optimization

Redshift over PostgreSQL (local):

- Demonstrates cloud-native architectural patterns

- MPP architecture provides better scalability story

- Industry-relevant experience (Redshift widely used in fintech)

- Fallback plan: PostgreSQL can substitute if AWS costs exceed budget

4.2 Data Collection and Analysis Techniques

Data Sources

- **Primary Stream:** Real-time trade-related market events are ingested from the **Coinbase WebSocket API** using the ticker channel. This channel provides event-driven price and volume updates triggered by executed trades.
- **Secondary Reference Stream:** A static or slowly changing reference dataset containing cryptocurrency symbol metadata (e.g., base asset, quote asset, category) is published to Kafka at startup.

Stream Processing

PySpark Structured Streaming performs a **stream-to-static join** between the live trade stream and the reference dataset to enrich events before aggregation.

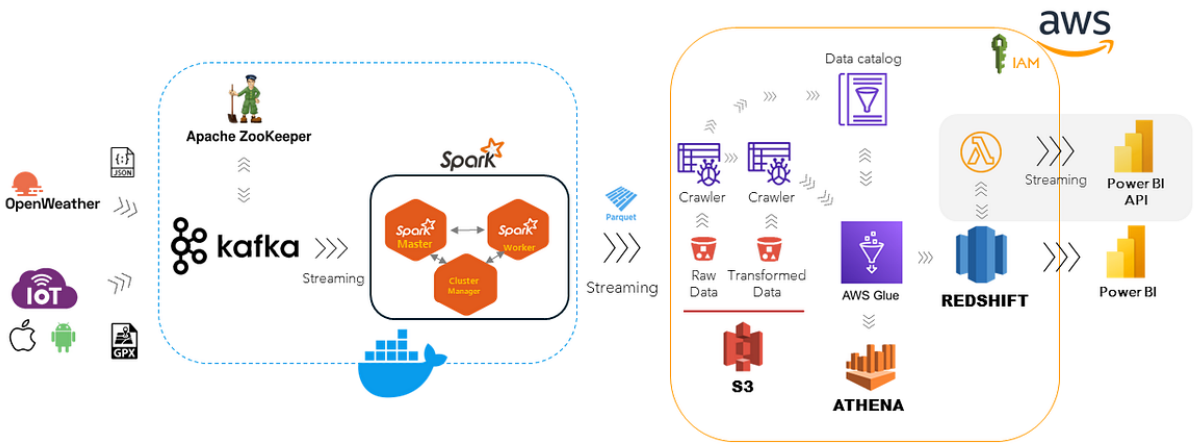
Analysis Techniques

- Windowed aggregations using 1-minute and 5-minute sliding windows
- Volume Weighted Average Price (VWAP)
- Moving averages
- Real-time volatility estimation using rolling standard deviation
- Watermarking with a 10-minute threshold to handle late-arriving data

4.3 Technology Stack

- **Operating System/Platform: macOS** (Development) and **AWS (Amazon Web Services)** for production deployment (EC2, MSK, Redshift).
- **Programming Languages/Frameworks: Python 3.11, PySpark 3.5,** and **Docker Compose** for environment orchestration.
- **Database: Amazon Redshift** (Data Warehouse) and S3 (Staging).
- **Backend: Apache Kafka** (Message Broker) and **Spark Streaming** (Processing Engine).
- **Frontend: Grafana** or **Amazon Quick Sight** for real-time dashboarding of trade metrics.

AWS Data Processing End-to-end pipeline | Street Drive lessons realtime monitoring



4.4Expected Results and Practical Applications

- **Near Real-Time Analytics:** The pipeline is expected to achieve seconds-level latency from trade occurrence to analytical visibility, suitable for operational monitoring.
- **Volatility Monitoring:** Aggregated metrics enable detection of abnormal price movements and volume spikes in real time.
- **Practical Contribution:** The project provides a reference architecture for transitioning from batch-based reporting to real-time market monitoring systems, supporting faster decision-making in high-frequency financial environments.

5. Project Planning and Timelines

Week	Phase	Milestone & Deliverables	Estimated Hours
02	**Research & Design**	Finalize Proposal; Research Coinbase WebSocket & Redshift schemas.	12 hrs
3	**Env. Setup**	Configure Docker, Kafka, and Spark Master/Worker containers.	10 hrs
4	**Data Ingestion**	Build producer.py to stream live BTC/ETH trades into Kafka.	15 hrs
5	**Processing I**	Develop PySpark script for basic stream consumption and cleaning.	15 hrs
6	**Midterm**	**Deliverable: Midterm Report & 10-min Video Demo**.	12 hrs
7	**Processing II**	Implement windowed aggregations (VWAP, Moving Averages).	15 hrs
8	**Storage**	Configure Redshift connector and implement data sinking logic.	12 hrs
9	**Refinement**	Implement real-time alerting and perform performance testing.	15 hrs
10	**Finalization**	**Deliverable: Final Report & Oral Defense Presentation**.	14 hrs

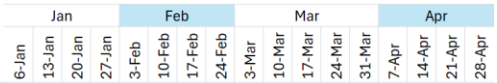
Gantt Chart

Project Start Date: 06-Jan-26

Project Name: Real-Time Market Sentiment and Volatility Tracking

Week starting:

#	Activity	Assigned To	Start	End	Days	Status	%Done
1	Research & Design	Michael	6-Jan-26	13-Jan-26	6	Complete	100%
2	Evn. Setup	Michael	13-Jan-26	20-Jan-26	3	Complete	100%
3	Data Ingestion	Michael	20-Jan-26	27-Jan-26	4	In progress	70%
4	Processing I	Michael	3-Feb-26	10-Feb-26	4	Not started	0%
5	Midterm Report	Michael	17-Feb-26	24-Feb-26	4	Not started	0%
6	Processing II	Michael	3-Mar-26	10-Mar-26	4	Not started	0%
7	Storage	Michael	17-Mar-26	24-Mar-26	4	Not started	0%
8	Refinement	Michael	24-Mar-26	31-Mar-26	4	Not started	0%
9	Final Report	Michael	31-Mar-26	7-Apr-26	4	Not started	0%
10							



Risk Mitigation Strategy

****Risk 1: Coinbase WebSocket Disconnections or API Changes****

- ****Likelihood:**** Medium (WebSocket connections can be unstable)

- ****Impact:**** Medium (could interrupt data collection)

- ****Mitigation:****

    - Implement automatic reconnection logic with exponential backoff

    - Store raw JSON messages to S3 as "Bronze layer" for replay capability

    - Monitor Coinbase developer changelog for API deprecations - Test reconnection logic under simulated network failures

****Risk 2: AWS Cost Overruns****

- ****Likelihood:**** Low (with proper management)

- ****Impact:**** High (budget constraint for student)

- ****Mitigation:****

    - Use Redshift Serverless (pay-per-query model)

    - Shut down clusters immediately after testing/demos

    - Set up AWS billing alerts at $5 and $10 thresholds

    - Fallback architecture: Replace Redshift with PostgreSQL (zero cost)

    - Store most data in S3 (minimal cost: <$1 for entire project)

****Risk 3: Scope Creep Leading to Timeline Delays****

- ****Likelihood:**** Medium (common in solo projects)

- ****Impact:**** High (could jeopardize final deliverable)

- ****Mitigation:****

    - Clearly defined "must-have" vs. "nice-to-have" features

- Phased delivery: Core pipeline operational by Week 6 (midterm)
- Weekly self-review against Gantt chart timeline
- Ruthlessly cut non-essential features if falling behind

****Risk 4: PySpark Learning Curve Delays****

- ****Likelihood:**** Low-Medium (new technology)
- ****Impact:**** Medium (could slow Week 5-7 progress)
- ****Mitigation:****
  - Week 5 dedicated specifically to learning Structured Streaming basics
  - Leverage extensive Spark documentation and tutorials
  - Start with simple transformations before adding complexity
  - Seek instructor/TA feedback early if struggling

****Risk 5: Data Quality Issues from WebSocket Feed****

- ****Likelihood:**** Medium (malformed JSON, network issues)
- ****Impact:**** Low (can be handled with robust error handling)
- ****Mitigation:****
  - Implement strict schema validation in PySpark
  - Route invalid messages to Dead Letter Queue (DLQ) for analysis
  - Comprehensive error logging and monitoring
  - Accept 1-2% message loss as acceptable for demonstration purposes

Project Contract

I, Michael Hein (Student ID: 300375535), hereby agree to the scope of work, deliverables, and timelines outlined in this project proposal for CSIS 4495.

Project Scope Commitment

I commit to delivering a functional real-time cryptocurrency data pipeline that demonstrates:
- Real-time ingestion from Coinbase WebSocket API
- Distributed message streaming via Apache Kafka
- Stream processing with PySpark Structured Streaming including windowed aggregations, watermarking, and late-event handling
- Data storage in cloud warehouse (Amazon Redshift Serverless or PostgreSQL alternative)
- Basic monitoring, alerting, and dashboard visualization

Timeline commitment

I commit to:
- Properly citing all external resources, tutorials, and AI assistance used in this project
- Documenting all AI tool usage comprehensively in the AI Use Section
- Ensuring all code submitted represents my own understanding and implementation work
- Maintaining detailed, honest work logs reflecting actual effort and progress
- Following all course policies regarding collaboration and originality

6. AI Use Section

AI Tool Name	Version, Account Type	Specific feature for which the AI tool was used	Value Addition
Gemini	Advanced (Pro)	Architectural Design: Brainstorming the integration of Kafka, PySpark, and Redshift for crypto data.	I defined the specific business use case (crypto volatility) and selected the Coinbase WebSocket as the primary data source.
Gemini	Advanced (Pro)	Technical Setup: Generating Docker file, docker-compose.yml, and requirements.txt for Mac.	I performed the manual installation and troubleshooting on my local Mac terminal to verify the environment was functional.
Google Antigravity	Advanced (Pro)	Python IDE: AI-assisted code generation for Kafka producers, PySpark streaming scripts, and Redshift connectors	I wrote and debugged all final implementation code. AI provided boilerplate structure and syntax suggestions. All business logic (windowing strategies, aggregation functions, watermarking configuration, error handling, DLQ routing) was designed, implemented, and tested by me.

AI Usage Summary

Percentage Breakdown (Estimated):

- AI-Generated Content (initial templates): ~20%

- My Modifications, Extensions, Debugging: ~50%

- Entirely Original Work (design decisions, business logic, testing): ~30%

Total value-added: ~80%

Academic Integrity Statement

I confirm that:
1. All code submitted is my own implementation, even where AI provided initial templates 2. I understand how every line of code works and can explain it in technical detail
3. All architectural and design decisions were made by me based on project requirements 4. AI tools were used as assistants to accelerate development, not as replacement for my own learning and understanding 5. This disclosure represents a complete and honest accounting of all AI usage throughout the project

7. Work Log Table

Date	Number of Hours	Description of work done
2026-01-16	3	Configured project environment by creating Docker file, docker-compose.yml, devcontainer. Json vis Mac terminal to ensure full-service isolation for Kafka and Spark
2026-01-16	4	Researched and initialized requirement.txt with essential dependencies with pyspark, Kafka and amazon redshift integration
2026-01-16	4	Researched and wrote the introduction, problem statement, proposed research project sections of the Project Proposal
2026-01-20	4	Designed project planning and timelines. Watched YouTube video on how to create Gantt chart on excel. Created the Gantt chart.
2026-01-20	1.5	Watched YouTube videos about Kafka
2026-01-21	1	Touched on the project proposal
2026-01-22	2	Refined and rewrote some parts of the project proposal

8. References

1. Fundamentals of Data Engineering Masterclass
   https://www.youtube.com/watch?v=hf2go3E2m8g

2. Learn Kafka in 10 minutes
   https://www.youtube.com/watch?v=Ydts27Qa8H0

3. Stock Market Real-time Data Analysis Using Kafka
   https://www.youtube.com/watch?v=KerNf0NANMo

4. Building a Real-Time Data Pipeline with PySpark, Kafka, and Redshift
   https://www.youtube.com/watch?v=iluXuIM-als

5. Real-Time Data Pipeline with Apache Kafka and AWS
   https://www.youtube.com/watch?v=G87fm-tjhmY
6. Fundamentals of Data Engineering: Plan and Build Robust Data Systems book by Joe Reis

https://www.amazon.ca/Fundamentals-Data-Engineering-Robust-Systems/dp/1098108302

7. Reis, J., & Housley, M. (2022). Fundamentals of Data Engineering: Plan and Build Robust Data Systems. O'Reilly Media.

8. Kleppmann, M. (2017). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. O'Reilly Media.

9. Coinbase. (2024). Advanced Trade WebSocket API Documentation. Retrieved from https://docs.cloud.coinbase.com/advanced-trade-api/

10. Apache Software Foundation. (2024). Apache Kafka Documentation. Retrieved from https://kafka.apache.org/documentation/

11. Databricks. (2024). Structured Streaming Programming Guide. Retrieved from https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

12. Amazon Web Services. (2024). Amazon Redshift Developer Guide. Retrieved from https://docs.aws.amazon.com/redshift/