CryptoStreamX:

Real-time Crypto Trade Analytics Pipeline

Name: Nuwan Keshara Galappaththi
LinkedIn: https://www.linkedin.com/in/nuwan-keshara/

Contents

1) Project

- **CryptoStreamX: Real-time Crypto Trade Analytics Pipeline**

CryptoStreamX is a fully streaming data pipeline that captures and processes live cryptocurrency trade data from Binance in real time. Built with Apache Kafka, the system ingests, buffers and distributes high-throughput event streams. Kafka Streams and ksqlDB power in-flight transformations, aggregations and real-time analytics without external compute layers. Kafka Connect integrates with ClickHouse, a high-performance columnar database, for ultra-fast storage and query execution on time-series trade data. This architecture delivers a scalable, fault-tolerant data flow from ingestion to analytics, enabling instant insights into trading activity, price movements and market patterns.

2) Objectives / Problem Statement

- Live cryptocurrency trade data is highly volatile and comes as a continuous stream. Analyzing this data using traditional batch processes introduces latency, limiting real-time insights into trading activity and price fluctuations..

  - Data Stream (Websocket) - wss://stream.binance.com:9443/stream?streams=btcusdt@trade

- *Goal*:

Build a fully streaming data pipeline that:

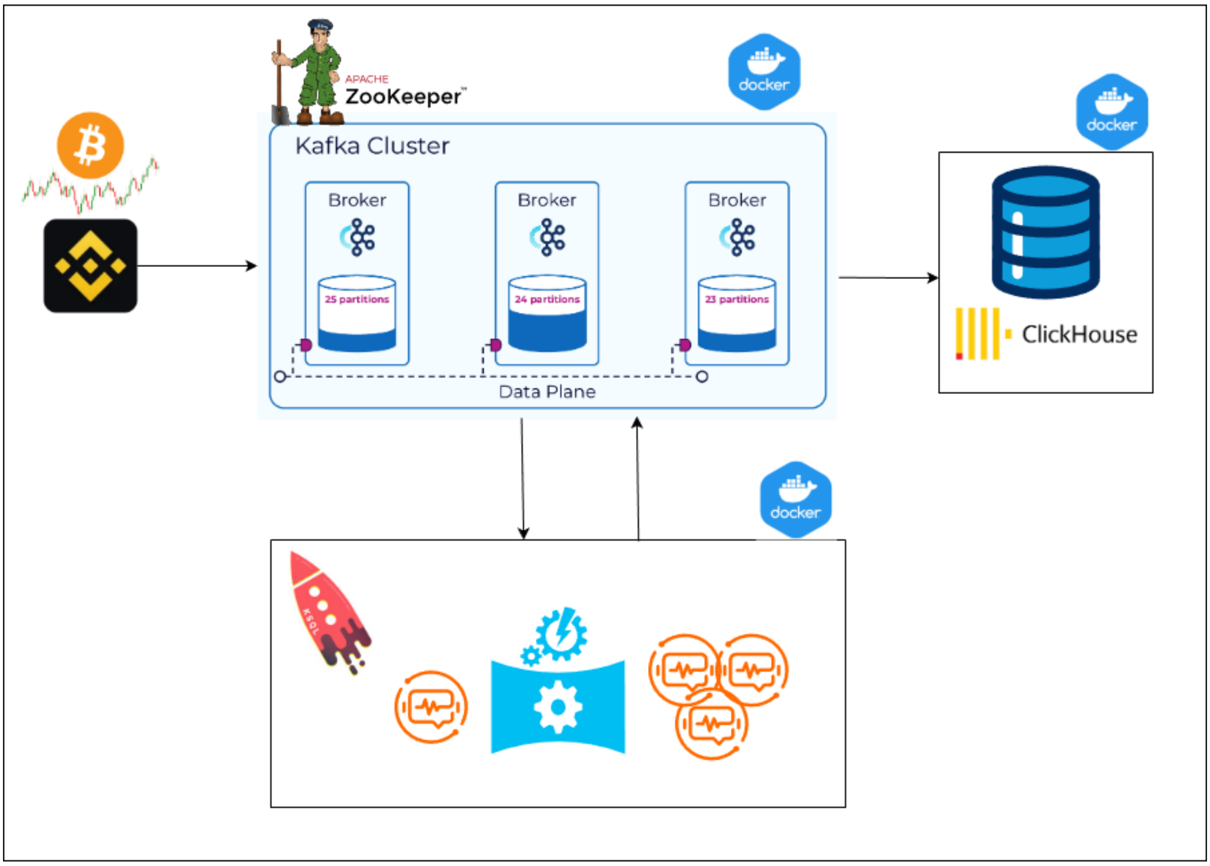
- Ingests live trade data from Binance in real time.
- Ensures data quality and schema consistency via Avro schemas and Kafka.
- Applies in-flight transformations and aggregations with ksqlDB.
- Stores processed trade data in ClickHouse for high-performance analytics.
- Enables near-real-time reporting.

3) Tech Stack

Layer	Tool	Purpose
Data Source	Binance WebSocket API	Live cryptocurrency trades (BTC, ETH, etc.)
Messaging & Streaming	Apache Kafka	High-throughput message broker for real-time ingestion and buffering
Cluster Management	Apache Zookeeper	Manage Cluster brokers
Schema Management	Confluent Schema Registry	Enforces Avro schemas for data consistency
Streaming Processing	ksqlDB / Kafka Streams	In-flight data transformations, aggregations and windowed analytics
Storage / Analytics	ClickHouse	Columnar DB optimized for time-series and high-performance queries
Programming	Python (WebSockets, Confluent Kafka)	Data ingestion, transformations
Version Control	Git / GitHub	Code management
Environment Management	Python dotenv	Secure configuration management

4) Architecture / System Design

- Pipeline Flow



Workflow Steps:

I. Live ingestion from Binance WebSocket into Kafka topics.

II. Schema validation using Confluent Schema Registry to ensure Avro schema compliance.

III. In-flight transformations and aggregations in ksqlDB:

IV. Windowed aggregations (1-minute and 5- minute tumbling windows, 10-second grace periods)

v. Real-time calculation of metrics like trade count, total quantity, average price, min/max prices.

vi. High-performance storage in ClickHouse for real-time querying.

5) Implementation Steps

1. Data Ingestion (Python)

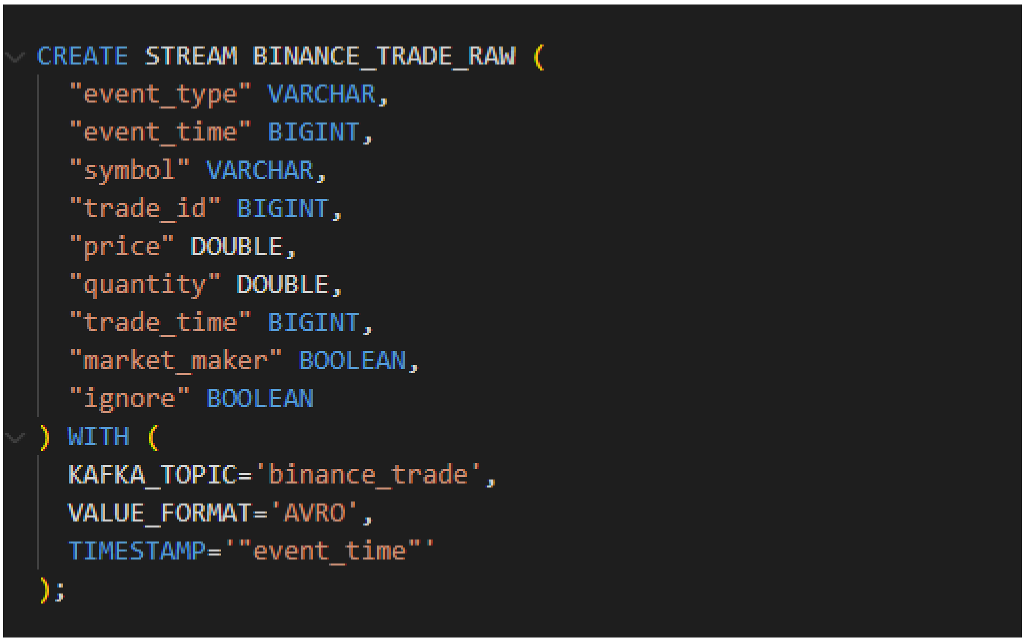
- WebSocket Client:
  - Connects to Binance WebSocket API.
  - Receives live trade messages and parses JSON payloads.
  - Transforms each trade into Avro-compliant dictionary records.
- Kafka Producer:
  - Serializes messages with AvroSerializer.
  - Publishes trade data to Kafka topic ‘binance_trade’.
  - Handles errors, retries and ensures minimal message loss.

2. Data Quality Checks (Python)

- Validates messages for:
  - Missing fields (e.g., event_time, symbol, trade_id)
  - Numeric validation (price ≥ 0, quantity ≥ 0)
  - Boolean field normalization (market_maker, ignore)
  - Logs issues to logs/producer.log.

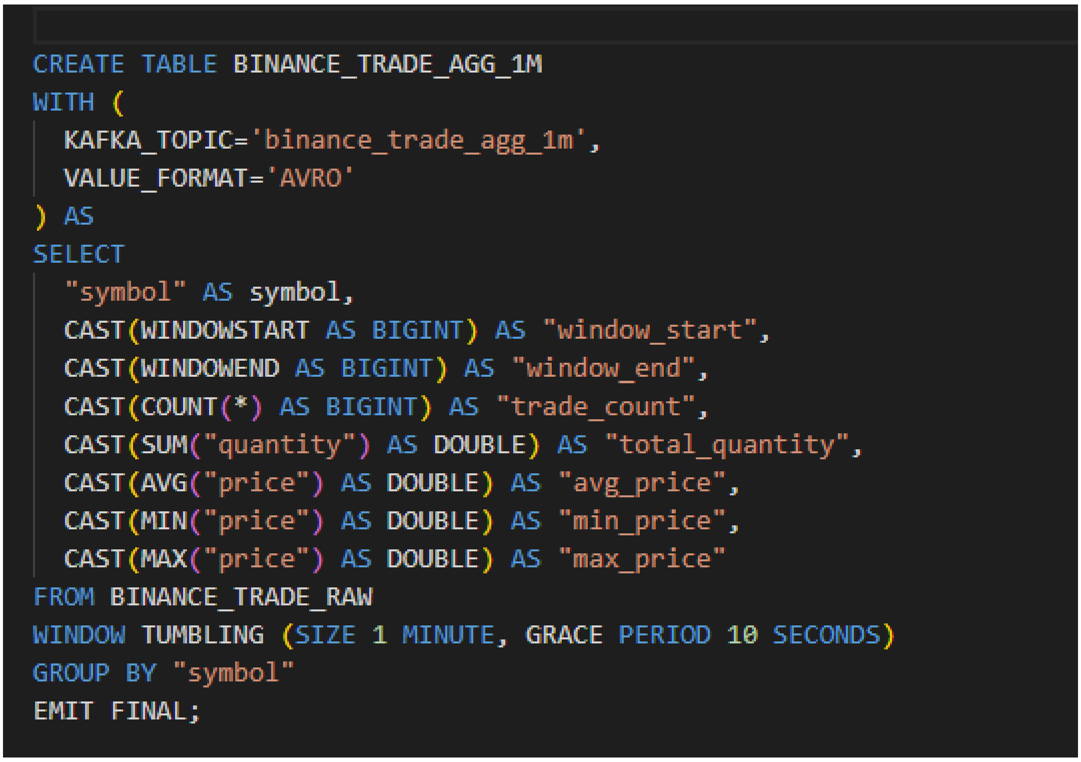
3. Transformation and Modeling (ksqlDB)

- Stream Creation:



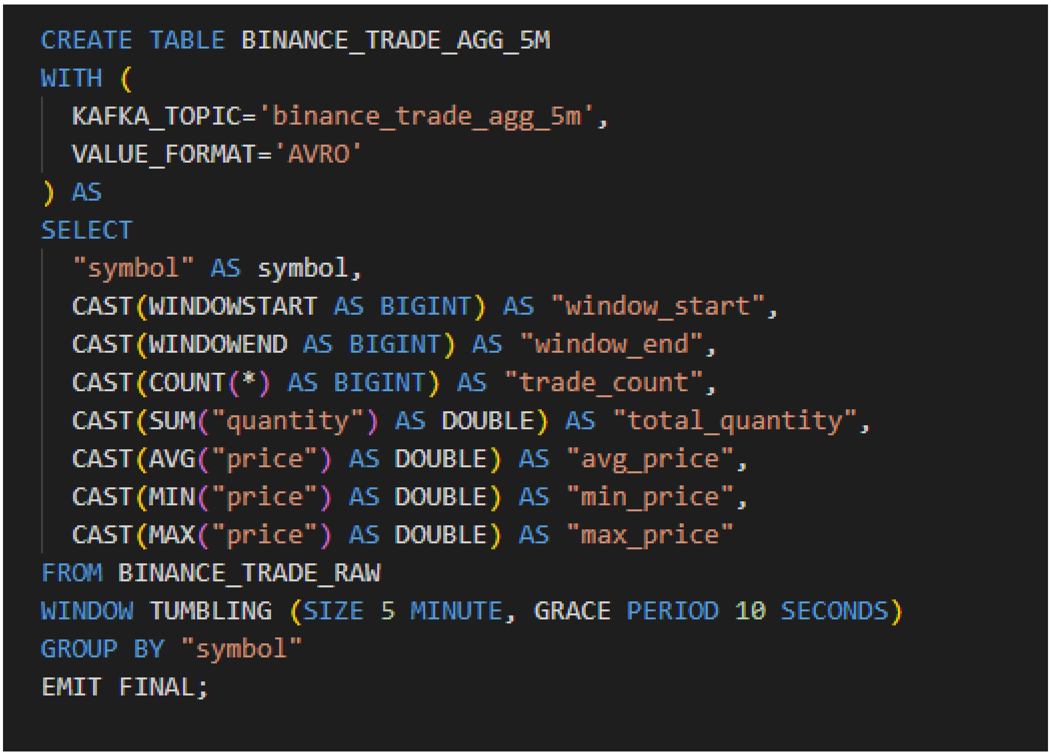
```sql
CREATE STREAM BINANCE_TRADE_RAW (
  "event_type" VARCHAR,
  "event_time" BIGINT,
  "symbol" VARCHAR,
  "trade_id" BIGINT,
  "price" DOUBLE,
  "quantity" DOUBLE,
  "trade_time" BIGINT,
  "market_maker" BOOLEAN,
  "ignore" BOOLEAN
) WITH (
  KAFKA_TOPIC='binance_trade',
  VALUE_FORMAT='AVRO',
  TIMESTAMP='"event_time"'
);
```

- Aggregations (1-minute tumbling windows):



```
CREATE TABLE BINANCE_TRADE_AGG_1M
WITH (
  KAFKA_TOPIC='binance_trade_agg_1m',
  VALUE_FORMAT='AVRO'
) AS
SELECT
  "symbol" AS symbol,
  CAST(WINDOWSTART AS BIGINT) AS "window_start",
  CAST(WINDOWEND AS BIGINT) AS "window_end",
  CAST(COUNT(*) AS BIGINT) AS "trade_count",
  CAST(SUM("quantity") AS DOUBLE) AS "total_quantity",
  CAST(AVG("price") AS DOUBLE) AS "avg_price",
  CAST(MIN("price") AS DOUBLE) AS "min_price",
  CAST(MAX("price") AS DOUBLE) AS "max_price"
FROM BINANCE_TRADE_RAW
WINDOW TUMBLING (SIZE 1 MINUTE, GRACE PERIOD 10 SECONDS)
GROUP BY "symbol"
EMIT FINAL;
```

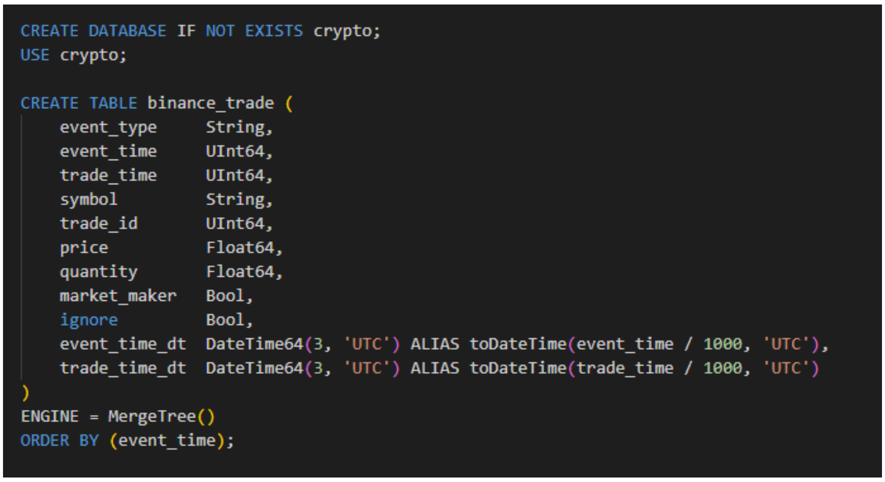
- Aggregations (5-minute tumbling windows):



```
CREATE TABLE BINANCE_TRADE_AGG_5M
WITH (
  KAFKA_TOPIC='binance_trade_agg_5m',
  VALUE_FORMAT='AVRO'
) AS
SELECT
  "symbol" AS symbol,
  CAST(WINDOWSTART AS BIGINT) AS "window_start",
  CAST(WINDOWEND AS BIGINT) AS "window_end",
  CAST(COUNT(*) AS BIGINT) AS "trade_count",
  CAST(SUM("quantity") AS DOUBLE) AS "total_quantity",
  CAST(AVG("price") AS DOUBLE) AS "avg_price",
  CAST(MIN("price") AS DOUBLE) AS "min_price",
  CAST(MAX("price") AS DOUBLE) AS "max_price"
FROM BINANCE_TRADE_RAW
WINDOW TUMBLING (SIZE 5 MINUTE, GRACE PERIOD 10 SECONDS)
GROUP BY "symbol"
EMIT FINAL;
```

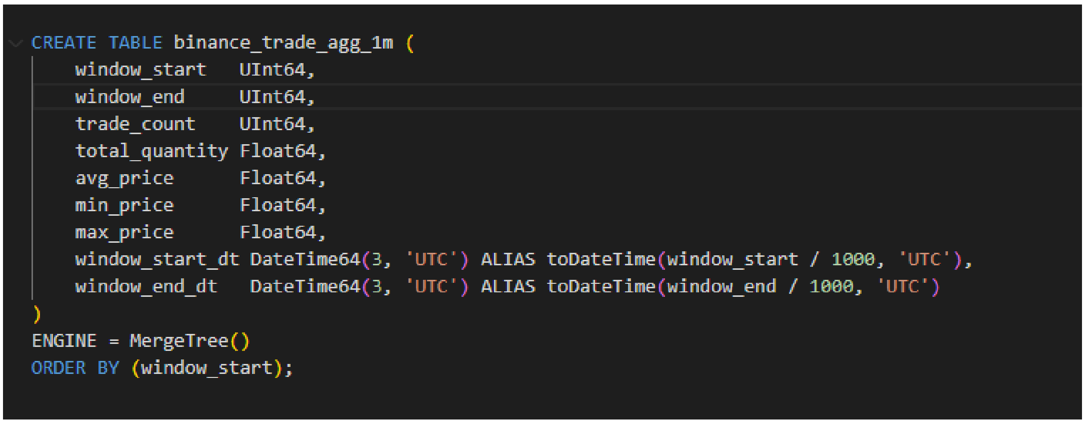
4. Storage (ClickHouse)

- Kafka Connect Sink streams aggregated tables into ClickHouse.
    - ClickHouse schema:



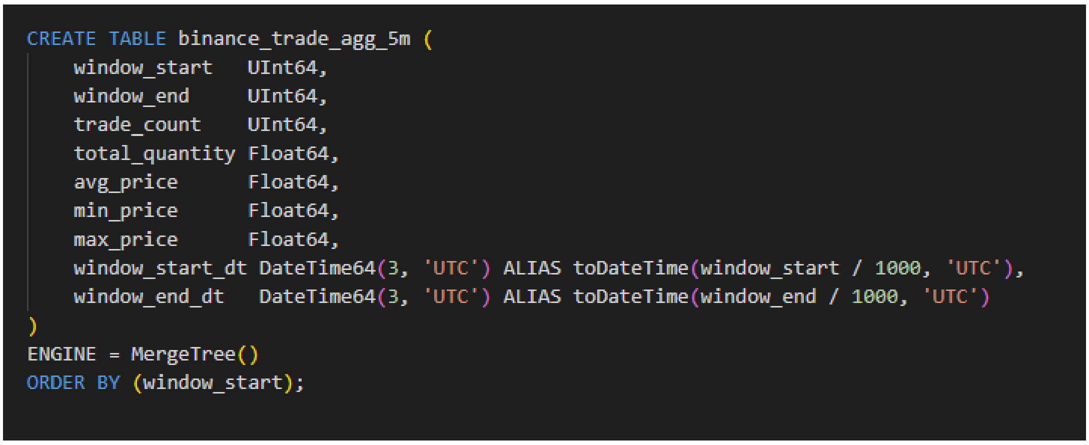
```sql
CREATE DATABASE IF NOT EXISTS crypto;
USE crypto;

CREATE TABLE binance_trade (
    event_type     String,
    event_time     UInt64,
    trade_time     UInt64,
    symbol         String,
    trade_id       UInt64,
    price          Float64,
    quantity       Float64,
    market_maker   Bool,
    ignore         Bool,
    event_time_dt  DateTime64(3, 'UTC') ALIAS toDateTime(event_time / 1000, 'UTC'),
    trade_time_dt  DateTime64(3, 'UTC') ALIAS toDateTime(trade_time / 1000, 'UTC')
)
ENGINE = MergeTree()
ORDER BY (event_time);
```

    - Aggregations (1-minute tumbling windows):



```sql
CREATE TABLE binance_trade_agg_1m (
    window_start   UInt64,
    window_end     UInt64,
    trade_count    UInt64,
    total_quantity Float64,
    avg_price      Float64,
    min_price      Float64,
    max_price      Float64,
    window_start_dt DateTime64(3, 'UTC') ALIAS toDateTime(window_start / 1000, 'UTC'),
    window_end_dt   DateTime64(3, 'UTC') ALIAS toDateTime(window_end / 1000, 'UTC')
)
ENGINE = MergeTree()
ORDER BY (window_start);
```

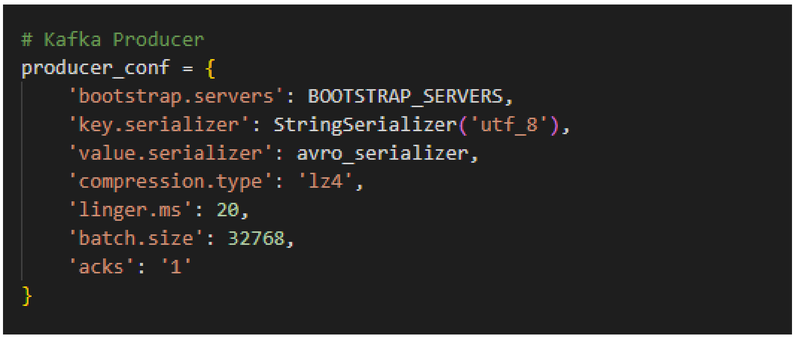
    - Aggregations (5-minute tumbling windows):



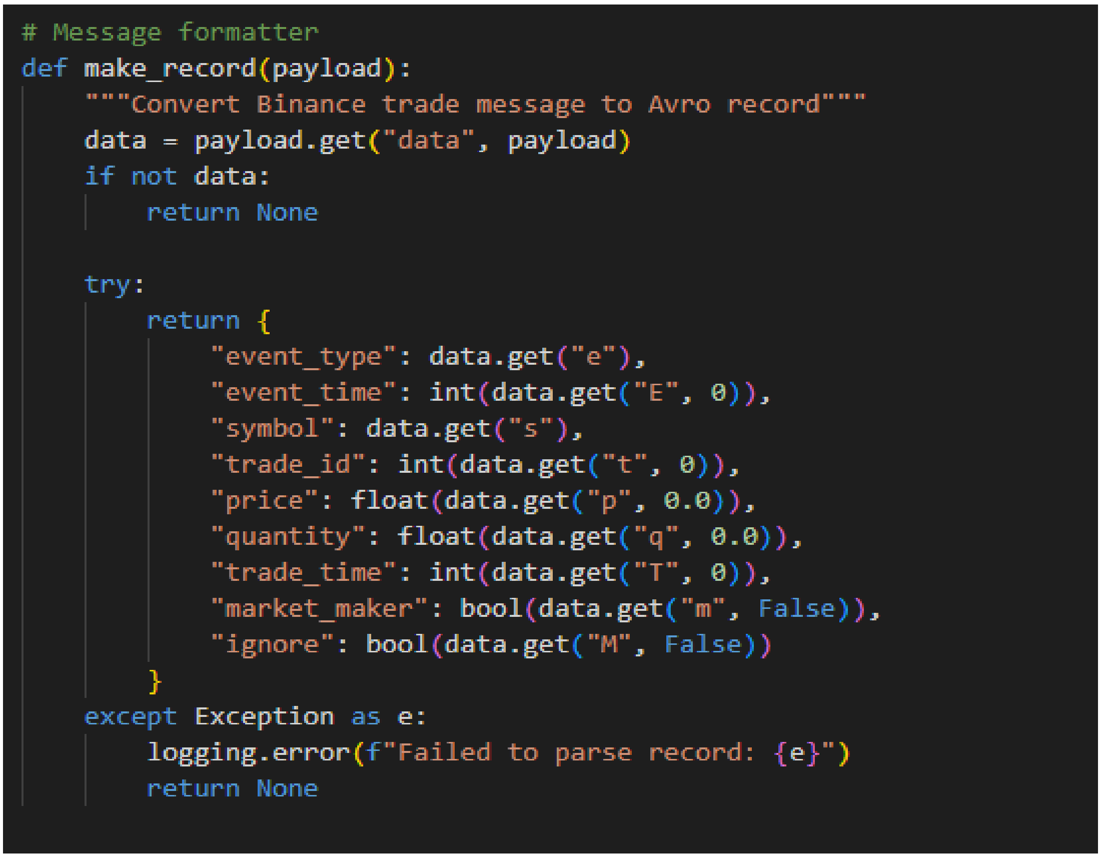
```sql
CREATE TABLE binance_trade_agg_5m (
    window_start   UInt64,
    window_end     UInt64,
    trade_count    UInt64,
    total_quantity Float64,
    avg_price      Float64,
    min_price      Float64,
    max_price      Float64,
    window_start_dt DateTime64(3, 'UTC') ALIAS toDateTime(window_start / 1000, 'UTC'),
    window_end_dt   DateTime64(3, 'UTC') ALIAS toDateTime(window_end / 1000, 'UTC')
)
ENGINE = MergeTree()
ORDER BY (window_start);
```

5. Code and Results

- Producer Example:

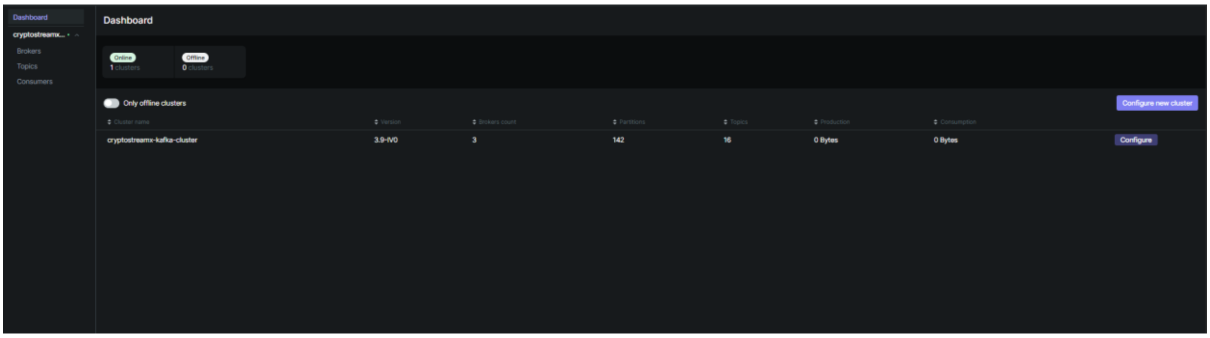


```python
# Kafka Producer
producer_conf = {
    'bootstrap.servers': BOOTSTRAP_SERVERS,
    'key.serializer': StringSerializer('utf_8'),
    'value.serializer': avro_serializer,
    'compression.type': 'lz4',
    'linger.ms': 20,
    'batch.size': 32768,
    'acks': '1'
}
```



```python
# Message formatter
def make_record(payload):
    """Convert Binance trade message to Avro record"""
    data = payload.get("data", payload)
    if not data:
        return None

    try:
        return {
            "event_type": data.get("e"),
            "event_time": int(data.get("E", 0)),
            "symbol": data.get("s"),
            "trade_id": int(data.get("t", 0)),
            "price": float(data.get("p", 0.0)),
            "quantity": float(data.get("q", 0.0)),
            "trade_time": int(data.get("T", 0)),
            "market_maker": bool(data.get("m", False)),
            "ignore": bool(data.get("M", False))
        }
    except Exception as e:
        logging.error(f"Failed to parse record: {e}")
        return None
```

- Kafka UI:



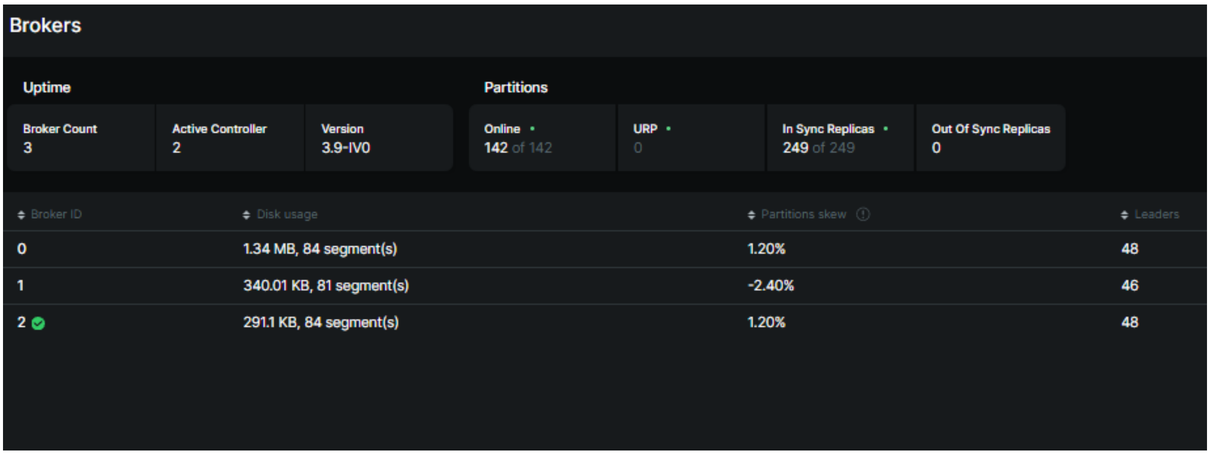
Dashboard

Online
1 clusters

Offline
0 clusters

Only offline clusters

Cluster name	Version	Brokers count	Partitions	Topics	Production	Consumption	
cryptostreamx-kafka-cluster	3.9-IV0	3	142	16	0 Bytes	0 Bytes	Configure



Brokers

Uptime

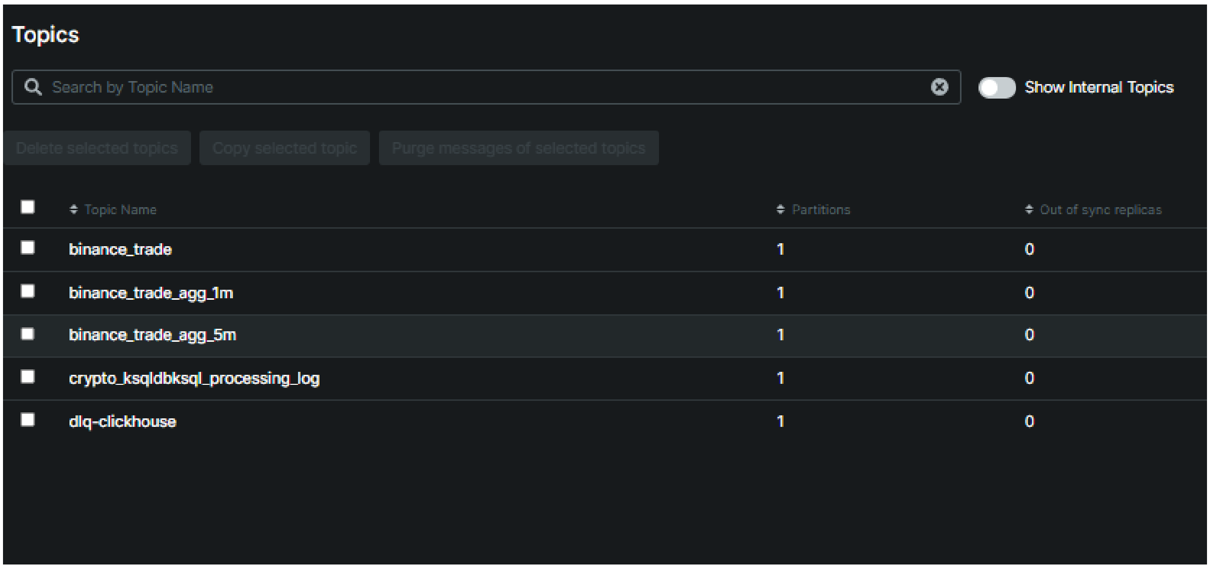
Broker Count	Active Controller	Version
3	2	3.9-IV0

Partitions

Online	URP	In Sync Replicas	Out Of Sync Replicas
142 of 142	0	249 of 249	0

Broker ID	Disk usage	Partitions skew	Leaders
0	1.34 MB, 84 segment(s)	1.20%	48
1	340.01 KB, 81 segment(s)	-2.40%	46
2 ✓	291.1 KB, 84 segment(s)	1.20%	48

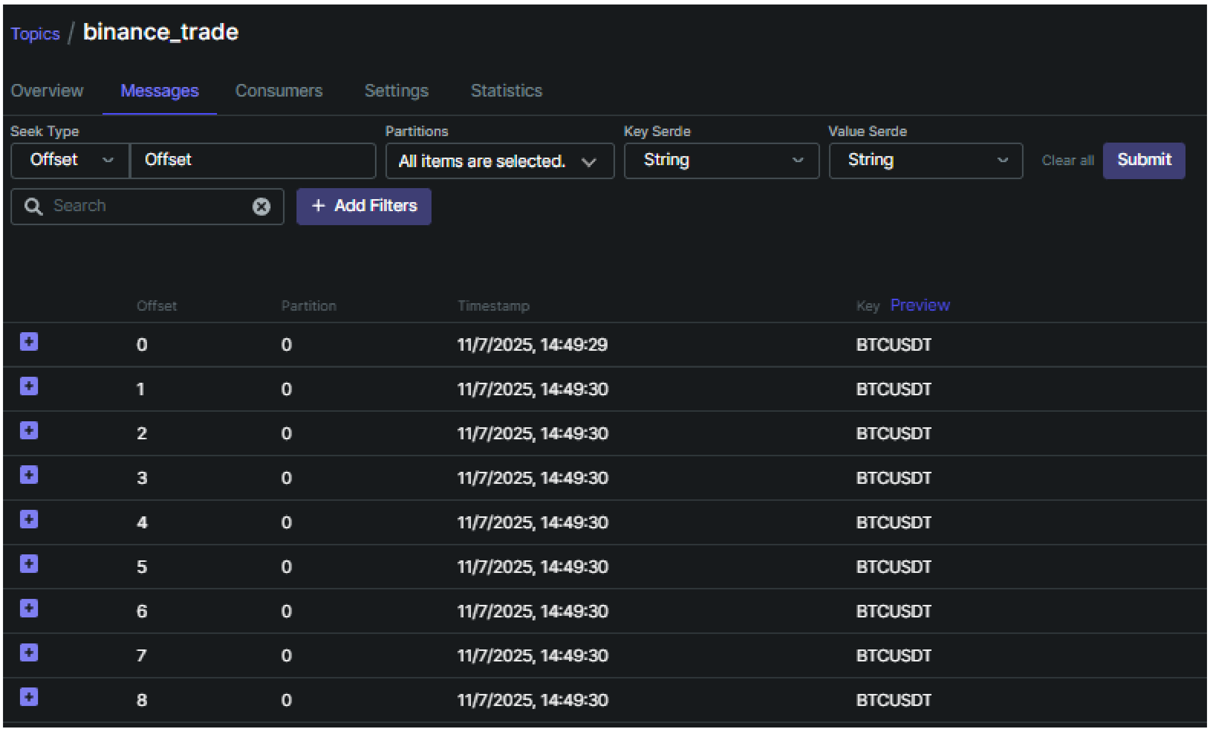
- Kafka Topics:



Topics

□	Topic Name	Partitions	Out of sync replicas
□	binance_trade	1	0
□	binance_trade_agg_1m	1	0
□	binance_trade_agg_5m	1	0
□	crypto_ksqldbksql_processing_log	1	0
□	dlq-clickhouse	1	0

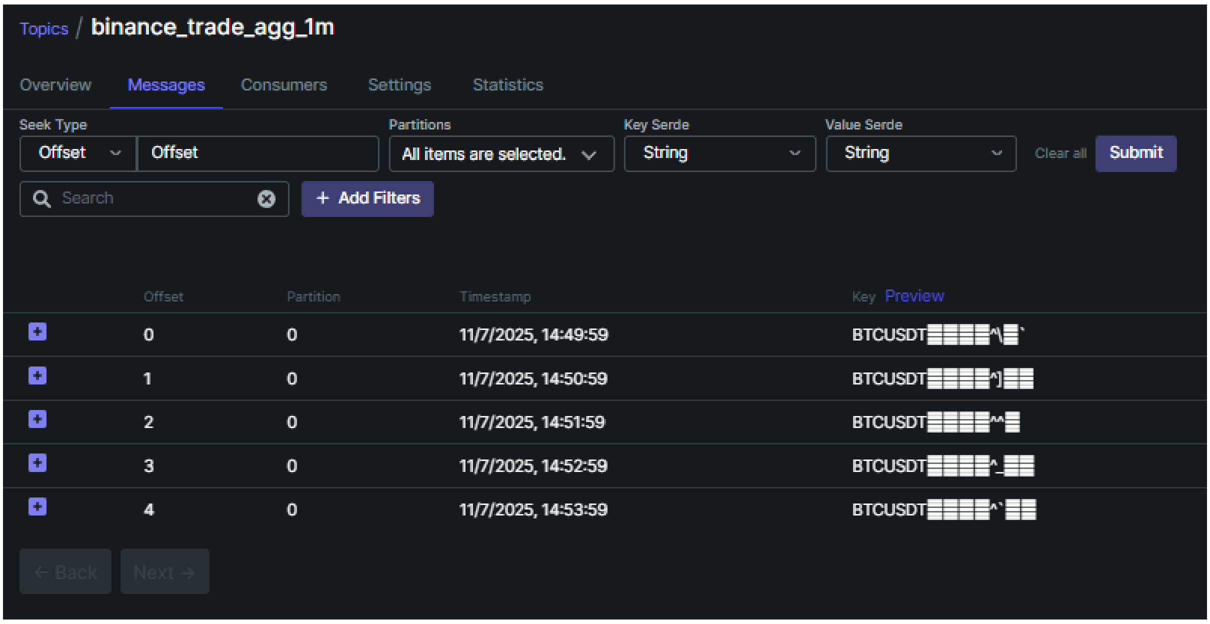
- Raw data stream- binance_trade topic



Seek Type: Offset | Offset
Partitions: All items are selected.
Key Serde: String
Value Serde: String

	Offset	Partition	Timestamp	Key Preview
⊞	0	0	11/7/2025, 14:49:29	BTCUSDT
⊞	1	0	11/7/2025, 14:49:30	BTCUSDT
⊞	2	0	11/7/2025, 14:49:30	BTCUSDT
⊞	3	0	11/7/2025, 14:49:30	BTCUSDT
⊞	4	0	11/7/2025, 14:49:30	BTCUSDT
⊞	5	0	11/7/2025, 14:49:30	BTCUSDT
⊞	6	0	11/7/2025, 14:49:30	BTCUSDT
⊞	7	0	11/7/2025, 14:49:30	BTCUSDT
⊞	8	0	11/7/2025, 14:49:30	BTCUSDT

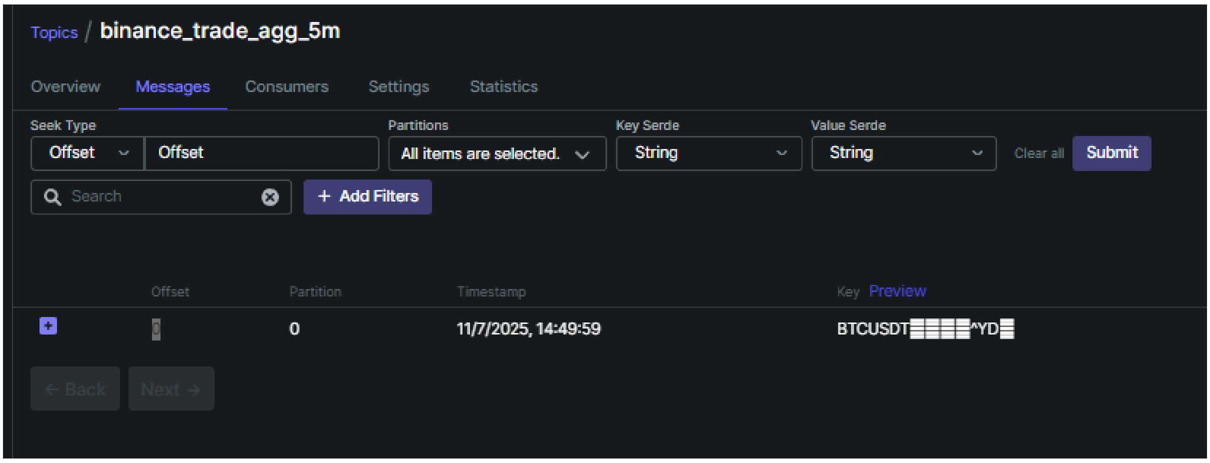
- Aggregated Data Stream 1-minute period - binance_trade_agg_1m (Avro)



Seek Type: Offset | Offset
Partitions: All items are selected.
Key Serde: String
Value Serde: String

	Offset	Partition	Timestamp	Key Preview
⊞	0	0	11/7/2025, 14:49:59	BTCUSDT▤▤▤▤^\▤`
⊞	1	0	11/7/2025, 14:50:59	BTCUSDT▤▤▤▤^]▤▤
⊞	2	0	11/7/2025, 14:51:59	BTCUSDT▤▤▤▤^^▤
⊞	3	0	11/7/2025, 14:52:59	BTCUSDT▤▤▤▤^_▤▤
⊞	4	0	11/7/2025, 14:53:59	BTCUSDT▤▤▤▤^`▤▤

- Aggregated Data Stream 5-minute period - binance_trade_agg_5m (Avro)



Seek Type: Offset | Offset

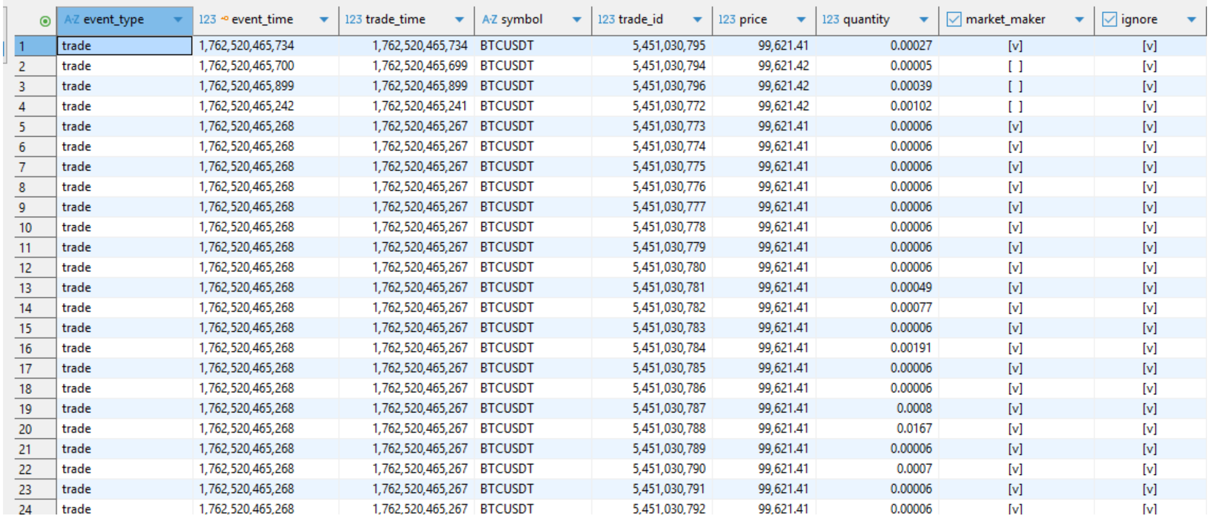
Partitions: All items are selected.

Key Serde: String

Value Serde: String

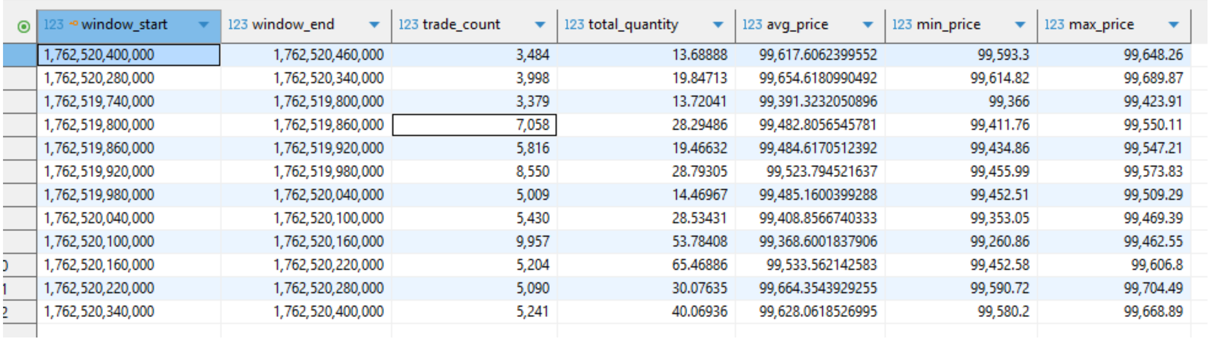
Offset	Partition	Timestamp	Key Preview
▯	0	11/7/2025, 14:49:59	BTCUSDT▆▆▆▆^YD▆

- Clickhouse Database raw data stream



	event_type	event_time	trade_time	symbol	trade_id	price	quantity	market_maker	ignore
1	trade	1,762,520,465,734	1,762,520,465,734	BTCUSDT	5,451,030,795	99,621.41	0.00027	[v]	[v]
2	trade	1,762,520,465,700	1,762,520,465,699	BTCUSDT	5,451,030,794	99,621.42	0.00005	[ ]	[v]
3	trade	1,762,520,465,899	1,762,520,465,899	BTCUSDT	5,451,030,796	99,621.42	0.00039	[ ]	[v]
4	trade	1,762,520,465,242	1,762,520,465,241	BTCUSDT	5,451,030,772	99,621.42	0.00102	[ ]	[v]
5	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,773	99,621.41	0.00006	[v]	[v]
6	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,774	99,621.41	0.00006	[v]	[v]
7	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,775	99,621.41	0.00006	[v]	[v]
8	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,776	99,621.41	0.00006	[v]	[v]
9	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,777	99,621.41	0.00006	[v]	[v]
10	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,778	99,621.41	0.00006	[v]	[v]
11	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,779	99,621.41	0.00006	[v]	[v]
12	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,780	99,621.41	0.00006	[v]	[v]
13	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,781	99,621.41	0.00049	[v]	[v]
14	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,782	99,621.41	0.00077	[v]	[v]
15	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,783	99,621.41	0.00006	[v]	[v]
16	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,784	99,621.41	0.00191	[v]	[v]
17	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,785	99,621.41	0.00006	[v]	[v]
18	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,786	99,621.41	0.00006	[v]	[v]
19	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,787	99,621.41	0.0008	[v]	[v]
20	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,788	99,621.41	0.0167	[v]	[v]
21	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,789	99,621.41	0.00006	[v]	[v]
22	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,790	99,621.41	0.0007	[v]	[v]
23	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,791	99,621.41	0.00006	[v]	[v]
24	trade	1,762,520,465,268	1,762,520,465,267	BTCUSDT	5,451,030,792	99,621.41	0.00006	[v]	[v]

- Clickhouse aggregated data (1m and 5m)



window_start	window_end	trade_count	total_quantity	avg_price	min_price	max_price
1,762,520,400,000	1,762,520,460,000	3,484	13.68888	99,617.6062399552	99,593.3	99,648.26
1,762,520,280,000	1,762,520,340,000	3,998	19.84713	99,654.6180990492	99,614.82	99,689.87
1,762,519,740,000	1,762,519,800,000	3,379	13.72041	99,391.3232050896	99,366	99,423.91
1,762,519,800,000	1,762,519,860,000	7,058	28.29486	99,482.8056545781	99,411.76	99,550.11
1,762,519,860,000	1,762,519,920,000	5,816	19.46632	99,484.6170512392	99,434.86	99,547.21
1,762,519,920,000	1,762,519,980,000	8,550	28.79305	99,523.794521637	99,455.99	99,573.83
1,762,519,980,000	1,762,520,040,000	5,009	14.46967	99,485.1600399288	99,452.51	99,509.29
1,762,520,040,000	1,762,520,100,000	5,430	28.53431	99,408.8566740333	99,353.05	99,469.39
1,762,520,100,000	1,762,520,160,000	9,957	53.78408	99,368.6001837906	99,260.86	99,462.55
1,762,520,160,000	1,762,520,220,000	5,204	65.46886	99,533.562142583	99,452.58	99,606.8
1,762,520,220,000	1,762,520,280,000	5,090	30.07635	99,664.3543929255	99,590.72	99,704.49
1,762,520,340,000	1,762,520,400,000	5,241	40.06936	99,628.0618526995	99,580.2	99,668.89

6) Summary

CryptoStreamX demonstrates a modern, end-to-end streaming data architecture that is scalable, fault-tolerant and built for real-time analytics. This system captures live cryptocurrency trade data, validates and transforms it in-flight and delivers analytics-ready metrics with minimal latency.

Core Components:

- Apache Kafka:
    - Handles high-throughput, real-time data ingestion and buffering between producers and consumers.
- Apache Zookeeper:
    - Manages Kafka cluster coordination, leader election and configuration synchronization across brokers, ensuring high availability and fault tolerance.
- Confluent Schema Registry:
    - Maintains Avro schemas to enforce consistent message structure and backward compatibility between producer and consumer applications.
- ksqlDB:
    - Performs continuous, in-flight transformations and real-time aggregations directly on Kafka topics.
- ClickHouse:
    - A high-performance columnar database for storing and querying time-series trade data, enabling fast analytical queries and historical trend analysis.

Key Achievements:

- Real-time trade analytics with low latency from data ingestion to visualization.
- Fault-tolerant Kafka cluster managed by Zookeeper for reliable message delivery and partition replication.
- Schema validation via Avro, preventing ingestion of inconsistent messages.

- Fully modular architecture, easily extendable to new data sources or additional metrics.
- Instant analytical queries, supporting aggregations on trade data.