Assignment-2

CHANAKYA UNIVERSITY

Real-time analysis using kafka

Submitted by
Dhanya Shanbhag [CU22MSD001A]

Submission Date
05-May-2024

Instructor
Prof. Sarath Sribhasyam

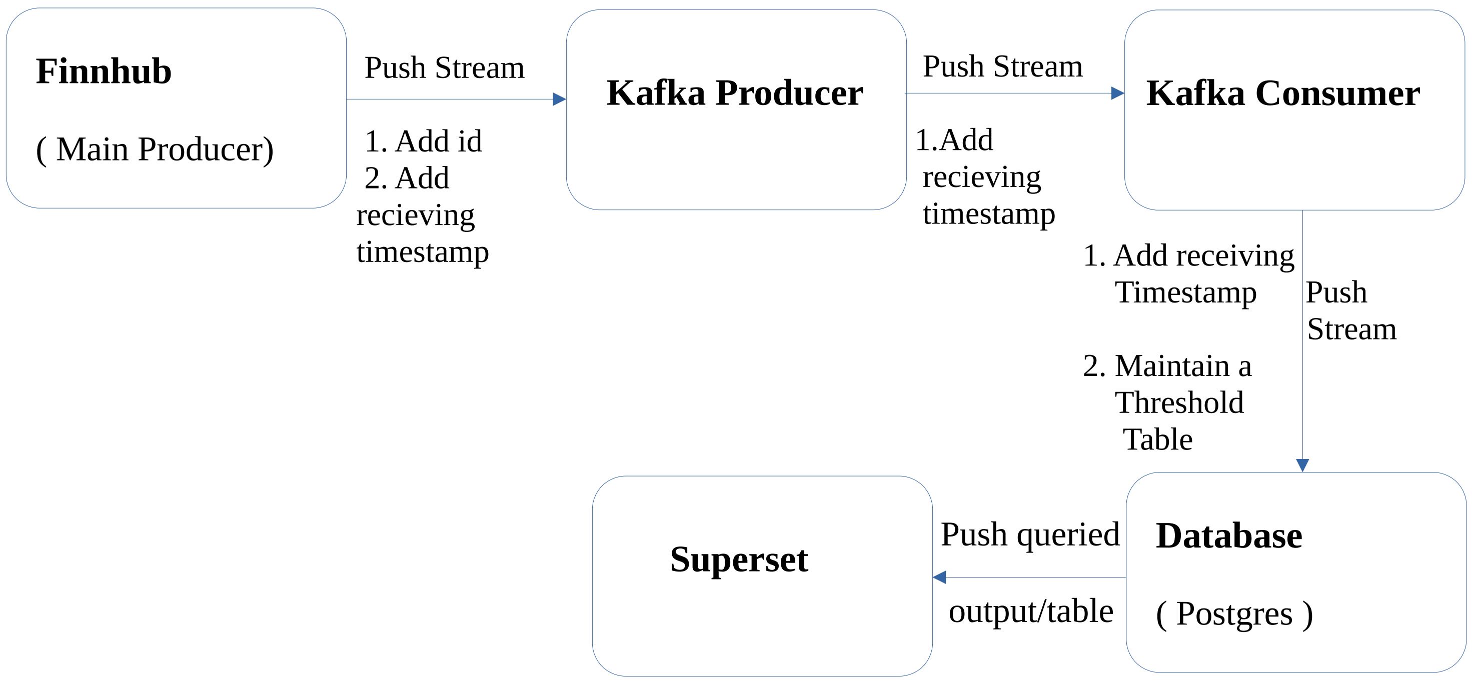
In partial fulfilment of the course

Real-time Analytics
3rd Semester 2024

Understand the scope :

The process of analyzing the data as it is recieved or generated can be reffered to as real-time analytics. Traditional data analysis often deals with processing of data that is collected over time, whereas real-time analysis deals with the data streams that need to be analysed very instantly, within a short time frame. The scope of real time analytics is very vast encompassing Business intelligence, Finance and Trading, Internet of things, Social media monitoring, Cybersecurity etc. The architechture of the analysis followed in this particular project can be seen as below.

Architechture of the process :


Finnhub
(Main Producer)
Push Stream
1. Add id
2. Add recieving timestamp
Kafka Producer
Push Stream
1.Add recieving timestamp
Kafka Consumer
1. Add receiving Timestamp
2. Maintain a Threshold Table
Push Stream
Database
(Postgres)
Push queried output/table
Superset

The real time data is streamed from the website named Finnhub (https://finnhub.io/), hence it is considered as the main/primary producer. Trade data from finnhub is used in this project. The data is streamed to kafka where a topic is created and the kafka producer reads the data that is arriving from the main producer. It also updates the time at which the data is being received. Then the data is streamed to kafka consumer where the receiving time is updated again. Then the data is written to the Database (Postgres in this case). The time to write to database is also updated here. In the database, once the table is created then the querying of the data begins. The data is queried based upon the KPI that needs to be generated. Also, a threshold table is maintained here. Finally the database is connected to the visualization tool (Superset in this case). Various metrics are been calculated and a dashboard is generated based upon the requirement of KPI's.

Note:
The whole project is carried out in Ubuntu OS – 22.04.4 LTS. The Kafka and zookeper are installed and the experiment is also done completely using the terminal interphase.

Stage 1 :

The code used to get the data from the finhub and make it available for the kafka-producer is as below. Python language is used in this project.

Code:

```
from kafka import KafkaProducer
import json
import time
from datetime import datetime, timezone, timedelta
import websocket

# Initialize Kafka producer
producer = KafkaProducer(bootstrap_servers='localhost:9092')

# Function to get current time in IST format
def current_time_ist():
    # Get current UTC time
    utc_now = datetime.utcnow()
    # Convert to IST
    ist_now = utc_now + timedelta(hours=5, minutes=30)
    # Format as string
    return ist_now.strftime('%Y-%m-%d %H:%M:%S')

# Function to handle websocket message
def on_message(ws, message):
    try:
        data = json.loads(message)
    except json.JSONDecodeError as e:
        print("Error decoding JSON:", e)
        return

    # Check if 'data' key exists
    if 'data' not in data:
        print("Skipping message as 'data' key is missing")
        return

    for record in data['data']:
        record['to_kafka'] = current_time_ist()
        producer.send('new_trade', json.dumps(record).encode('utf-8'))
        time.sleep(0.1)  # Add a small delay between sending messages (optional)
        print("Sent:", json.dumps(record))  # Print the sent message (optional)

# Function to handle websocket error
def on_error(ws, error):
    print("WebSocket error:", error)
```

```
# Function to handle websocket close
def on_close(ws):
    print("### closed ###")

# Function to handle websocket open
def on_open(ws):
    ws.send('{"type":"subscribe","symbol":"AAPL"}')
    ws.send('{"type":"subscribe","symbol":"AMZN"}')
    ws.send('{"type":"subscribe","symbol":"BINANCE:BTCUSDT"}')
    ws.send('{"type":"subscribe","symbol":"IC MARKETS:1"}')

# Run the Kafka producer
if __name__ == "__main__":
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("wss://ws.finnhub.io?
token=cnvamh9r01qub9j0mnq0cnvamh9r01qub9j0mnqg",
                              on_message=on_message,
                              on_error=on_error,
                              on_close=on_close)
    ws.on_open = on_open

    # Run the websocket for 5 minutes
    ws.run_forever()
    time.sleep(300)  # 5 minutes
    ws.close()

# Close the Kafka producer
producer.close()
```

Stage 2: Kafka Producer

Here, the data streamed by the finhub is received by the kafka producer. Also, the receiving timestamp is added. The complete code is as below:

Code

```
from kafka import KafkaProducer
import json
import time
from datetime import datetime, timezone, timedelta
import websocket

# Initialize Kafka producer
producer = KafkaProducer(bootstrap_servers='localhost:9092')

# Function to get current time in IST format
def current_time_ist():
    # Get current UTC time
    utc_now = datetime.utcnow()
    # Convert to IST
    ist_now = utc_now + timedelta(hours=5, minutes=30)
    # Format as string
    return ist_now.strftime('%Y-%m-%d %H:%M:%S')

# Function to handle websocket message
def on_message(ws, message):
    try:
        data = json.loads(message)
    except json.JSONDecodeError as e:
        print("Error decoding JSON:", e)
        return

    # Check if 'data' key exists
    if 'data' not in data:
        print("Skipping message as 'data' key is missing")
        return

    for record in data['data']:
        record['to_kafka'] = current_time_ist()
        producer.send('realtrade', json.dumps(record).encode('utf-8'))
        time.sleep(0.1)  # Add a small delay between sending messages (optional)
        print("Sent:", json.dumps(record))  # Print the sent message (optional)

# Function to handle websocket error
def on_error(ws, error):
    print("WebSocket error:", error)

# Function to handle websocket close
def on_close(ws):
    print("### closed ###")

# Function to handle websocket open
def on_open(ws):
    ws.send('{"type":"subscribe","symbol":"AAPL"}')
    ws.send('{"type":"subscribe","symbol":"AMZN"}')
    ws.send('{"type":"subscribe","symbol":"BINANCE:BTCUSDT"}')
    ws.send('{"type":"subscribe","symbol":"IC MARKETS:1"}')
# Run the Kafka producer
if __name__ == "__main__":
    websocket.enableTrace(True)
    ws = websocket.WebSocketApp("wss://ws.finnhub.io?
token=cnvamh9r01qub9j0mnq0cnvamh9r01qub9j0mnqg",
                                on_message=on_message,
                                on_error=on_error,
                                on_close=on_close)
    ws.on_open = on_open

    # Run the websocket for 5 minutes
    ws.run_forever()
    time.sleep(300)  # 5 minutes
    ws.close()

# Close the Kafka producer
producer.close()
```

Stage 3: Kafka Consumer and writing to database

Here, the data streamed from the kafka producer is caught hold by the kafka consumer and is written to the database by making a connection to the postgresql. The complete code used to perform this task is as below:

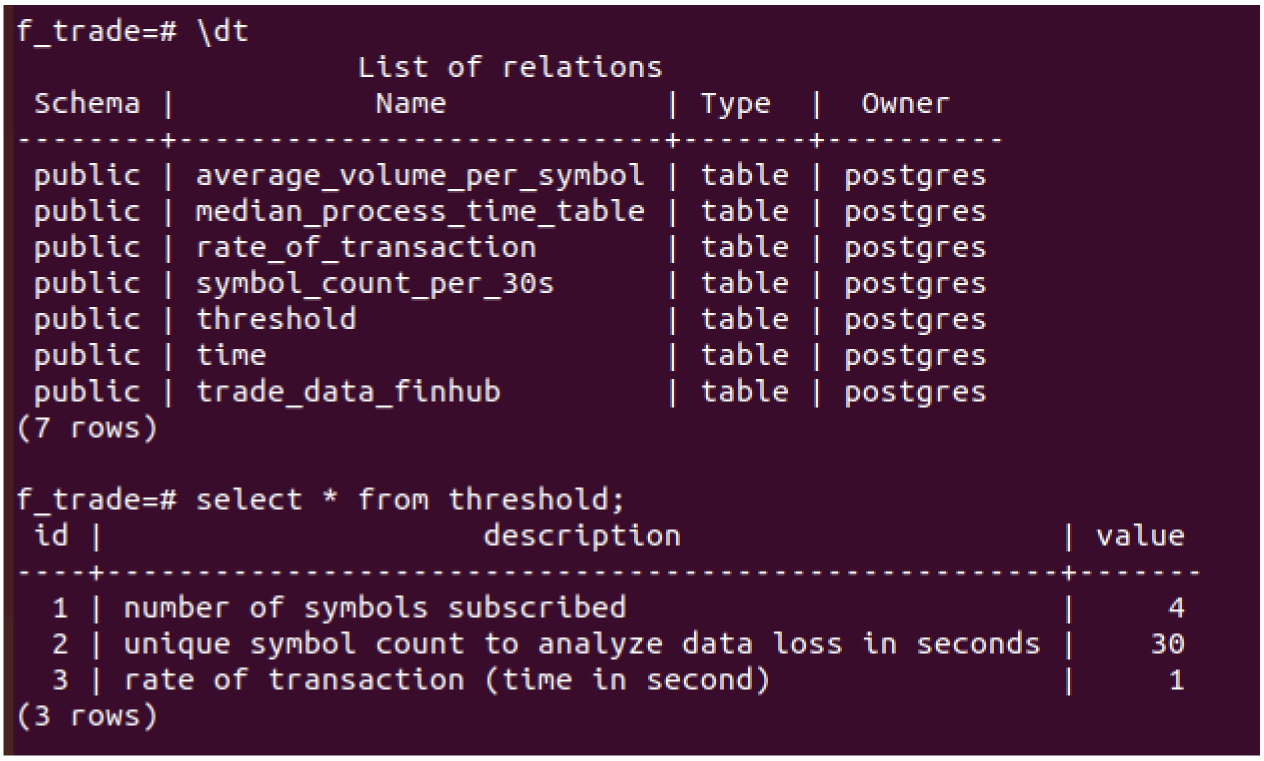
Code

```
import psycopg2
from kafka import KafkaConsumer
import json
from datetime import datetime, timedelta

# Function to convert epoch time to IST format
def epoch_to_ist(epoch_time):
    # Convert milliseconds to seconds
    epoch_seconds = epoch_time / 1000
    # Convert epoch time to UTC datetime object
    utc_datetime = datetime.utcfromtimestamp(epoch_seconds)
    # Add 5 hours and 30 minutes to convert to IST
    ist_datetime = utc_datetime + timedelta(hours=5, minutes=30)
    # Format the IST datetime as a string
    ist_time = ist_datetime.strftime('%Y-%m-%d %H:%M:%S')
    return ist_time

# Function to get current time in IST format
def current_time_ist():
    # Get current UTC time
    utc_now = datetime.utcnow()
    # Convert to IST
    ist_now = utc_now + timedelta(hours=5, minutes=30)
    # Format as string
    return ist_now.strftime('%Y-%m-%d %H:%M:%S')

# Connect to PostgreSQL database
conn = psycopg2.connect(
    dbname='f_trade',
    user='postgres',
    password='1234',
    host='localhost'
)

# Create a cursor object
cur = conn.cursor()
```

```
# Create table if not exists
cur.execute("""
    CREATE TABLE IF NOT EXISTS Trade_data_finhub (
        id SERIAL PRIMARY KEY,
        symbol VARCHAR(255),
        price NUMERIC,
        timestamp TIMESTAMP,
        volume NUMERIC,
        to_kafka TIMESTAMP,
        to_consumer TIMESTAMP,
        to_db TIMESTAMP
    )
""")

# Initialize Kafka consumer
consumer = KafkaConsumer('realtrade', bootstrap_servers='localhost:9092')

# Read data from Kafka topic
for message in consumer:
    try:
        # Parse JSON from Kafka message value
        data = json.loads(message.value.decode('utf-8'))

        # Check if 'data' key exists
        if 'data' not in data:
            print("Skipping message as 'data' key is missing")
            continue

        # Iterate over each record in the 'data' list
        for record in data['data']:
            # Add 'to_consumer' field with current IST time
            record['to_consumer'] = current_time_ist()

            # Add 'to_db' field with current IST time for database insertion
            record['to_db'] = current_time_ist()

            # Convert record to JSON string
            record_json = json.dumps(record)

            # Insert record into PostgreSQL database
            cur.execute("""
                INSERT INTO tradedata (symbol, price, timestamp, volume, to_kafka, to_consumer,
to_db)
                VALUES (%s, %s, %s, %s, %s, %s, %s)
            """, (record['s'], record['p'], epoch_to_ist(int(record['t'])), record['v'], record['to_kafka'],
record['to_consumer'], record['to_db']))
            conn.commit()

            # Print the record with 'to_consumer' and 'to_db' fields added
            print("Inserted into database:", record_json)
    except json.JSONDecodeError as e:
        print(f"Skipping message due to JSON decode error in Consumer: {e}")  # Print JSON
decoding error
    except Exception as ex:
        print(f"An error occurred: {ex}")

# Close the consumer and database connection
consumer.close()
conn.close()
```

Database Stage:

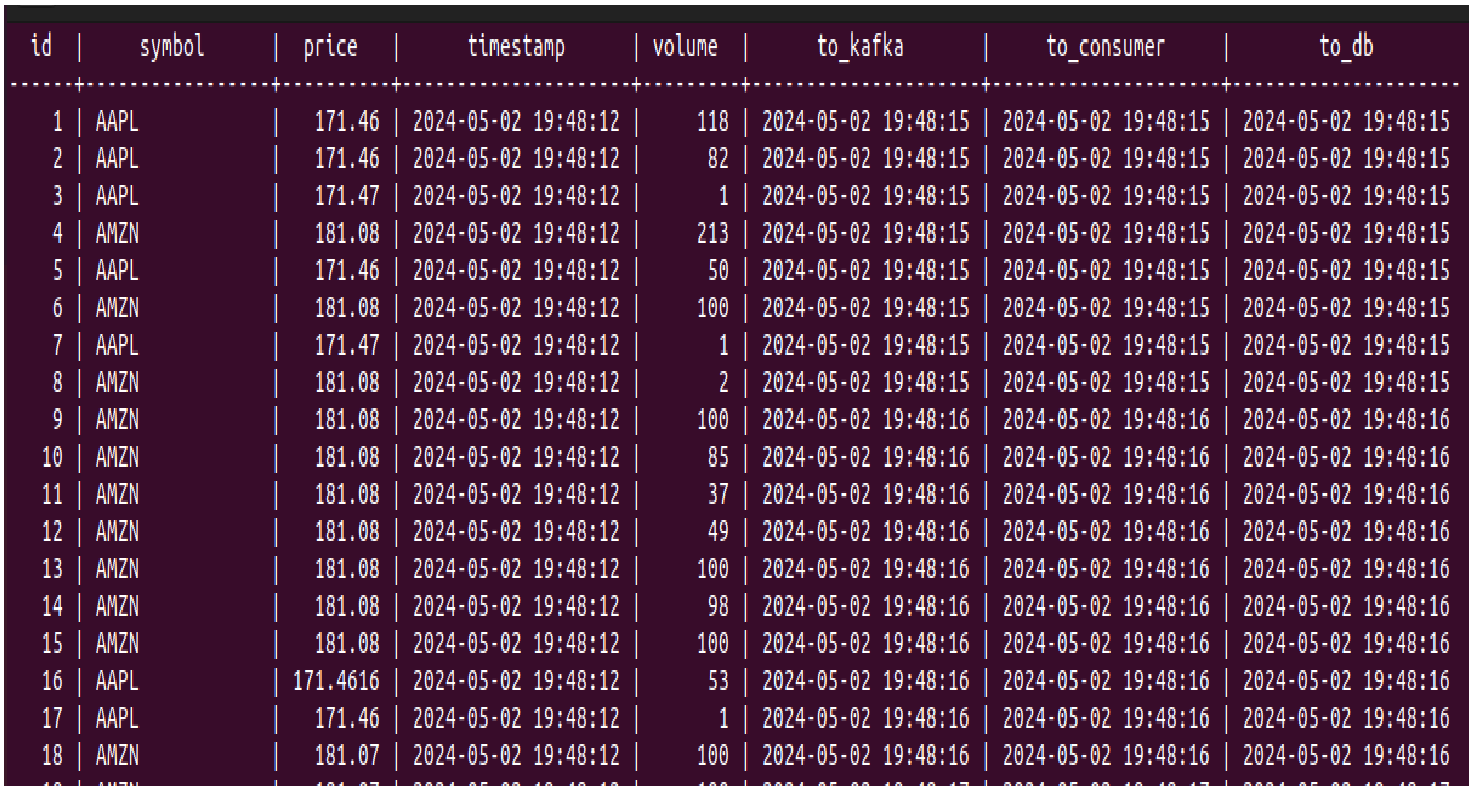
Below is the image from the terminal interphase of the postgresql. The database name is f_trade. It has 7 tables. The main table that has the data that was written by the kafka consumer is named as ‘trade_data_finhub’. The rest of the tables are derived based upon querying upon the main table ‘trade_data_finhub’. This is done to generate the different metric representaions necessary to display the KPI in the dashboard. Also, the threshold table is created that has the description of what values have been stored. The contents of the threshold table is displayed in the image1 below.



```
f_trade=# \dt
                  List of relations
 Schema |           Name            | Type  |  Owner
--------+---------------------------+-------+----------
 public | average_volume_per_symbol | table | postgres
 public | median_process_time_table | table | postgres
 public | rate_of_transaction       | table | postgres
 public | symbol_count_per_30s      | table | postgres
 public | threshold                 | table | postgres
 public | time                      | table | postgres
 public | trade_data_finhub         | table | postgres
(7 rows)

f_trade=# select * from threshold;
 id |                      description                      | value
----+-------------------------------------------------------+-------
  1 | number of symbols subscribed                          |     4
  2 | unique symbol count to analyze data loss in seconds   |    30
  3 | rate of transaction (time in second)                  |     1
(3 rows)
```

image1

The sample of the trade_data_finhub' table is ahown in the image2 below:



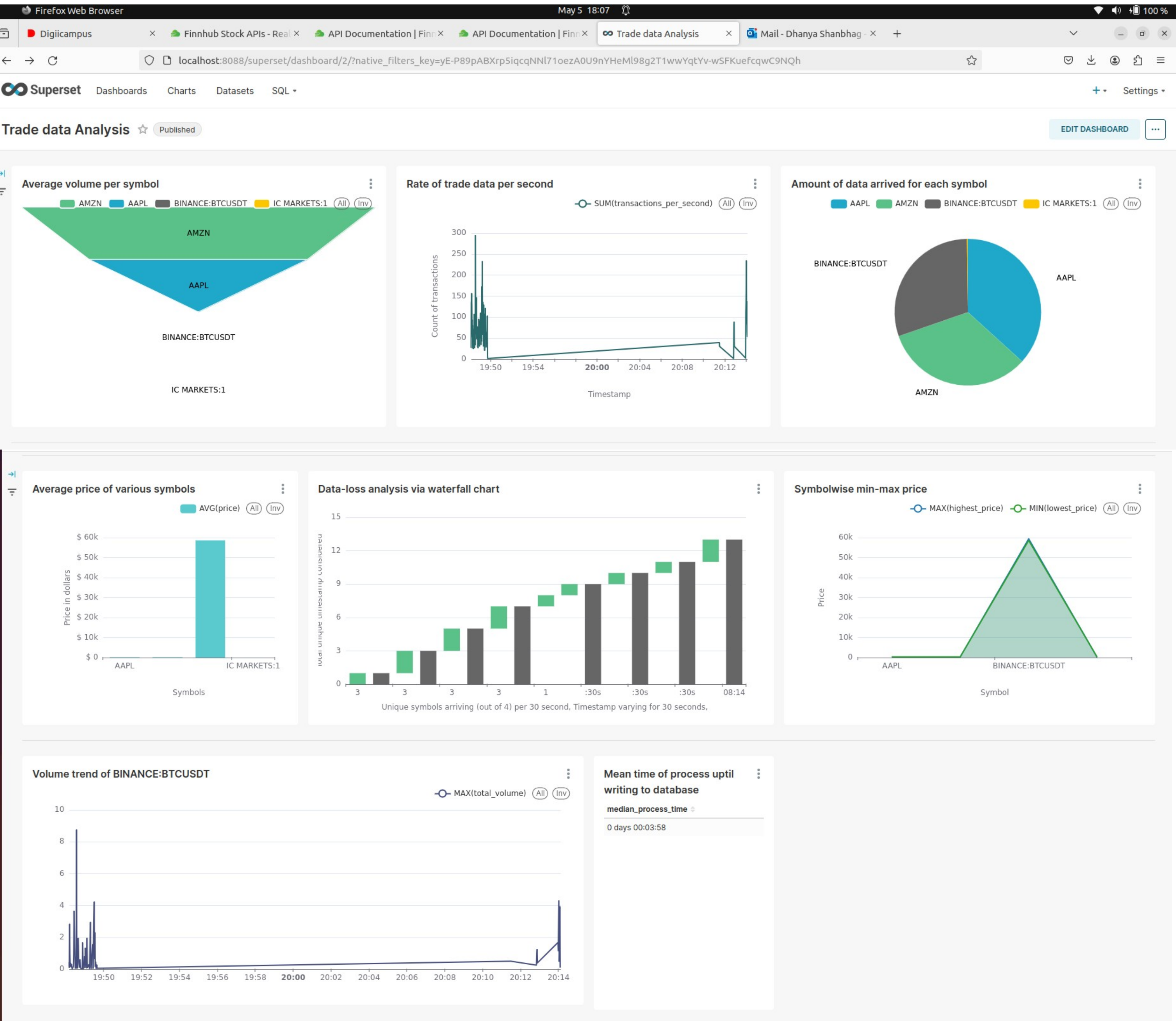
```
 id |     symbol      |  price   |      timestamp      | volume  |      to_kafka       |     to_consumer     |        to_db
----+-----------------+----------+---------------------+---------+---------------------+---------------------+---------------------
  1 | AAPL            |   171.46 | 2024-05-02 19:48:12 |     118 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15
  2 | AAPL            |   171.46 | 2024-05-02 19:48:12 |      82 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15
  3 | AAPL            |   171.47 | 2024-05-02 19:48:12 |       1 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15
  4 | AMZN            |   181.08 | 2024-05-02 19:48:12 |     213 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15
  5 | AAPL            |   171.46 | 2024-05-02 19:48:12 |      50 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15
  6 | AMZN            |   181.08 | 2024-05-02 19:48:12 |     100 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15
  7 | AAPL            |   171.47 | 2024-05-02 19:48:12 |       1 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15
  8 | AMZN            |   181.08 | 2024-05-02 19:48:12 |       2 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15 | 2024-05-02 19:48:15
  9 | AMZN            |   181.08 | 2024-05-02 19:48:12 |     100 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 10 | AMZN            |   181.08 | 2024-05-02 19:48:12 |      85 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 11 | AMZN            |   181.08 | 2024-05-02 19:48:12 |      37 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 12 | AMZN            |   181.08 | 2024-05-02 19:48:12 |      49 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 13 | AMZN            |   181.08 | 2024-05-02 19:48:12 |     100 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 14 | AMZN            |   181.08 | 2024-05-02 19:48:12 |      98 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 15 | AMZN            |   181.08 | 2024-05-02 19:48:12 |     100 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 16 | AAPL            | 171.4616 | 2024-05-02 19:48:12 |      53 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 17 | AAPL            |   171.46 | 2024-05-02 19:48:12 |       1 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
 18 | AMZN            |   181.07 | 2024-05-02 19:48:12 |     100 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16 | 2024-05-02 19:48:16
```

image2

The trade_data_finhub consists of id, symbol, price, timestamp, volume, to_kafka, to_consumer and to_db columns. The columns symbol, price, timestamp and volume were originally present inside the data. The symbol column has four unique symbols that have been subscribed to.Whereas the rest of the columns are added as a part of the process. to_kafka indicates the time when each record is received by the kafka producer. Similarly, to_consumer is the time when the data is read by the kafka consumer, and to_db is the time when each data record is written to the database.

Final stage:

The f_trade database is connected to the visualization tool named Superset. The overview of the dashboard 'Trade data Analysis' can be seen as below in image3. A total of 8 charts can be seen on the dashboard, each of which have been explained below individually.

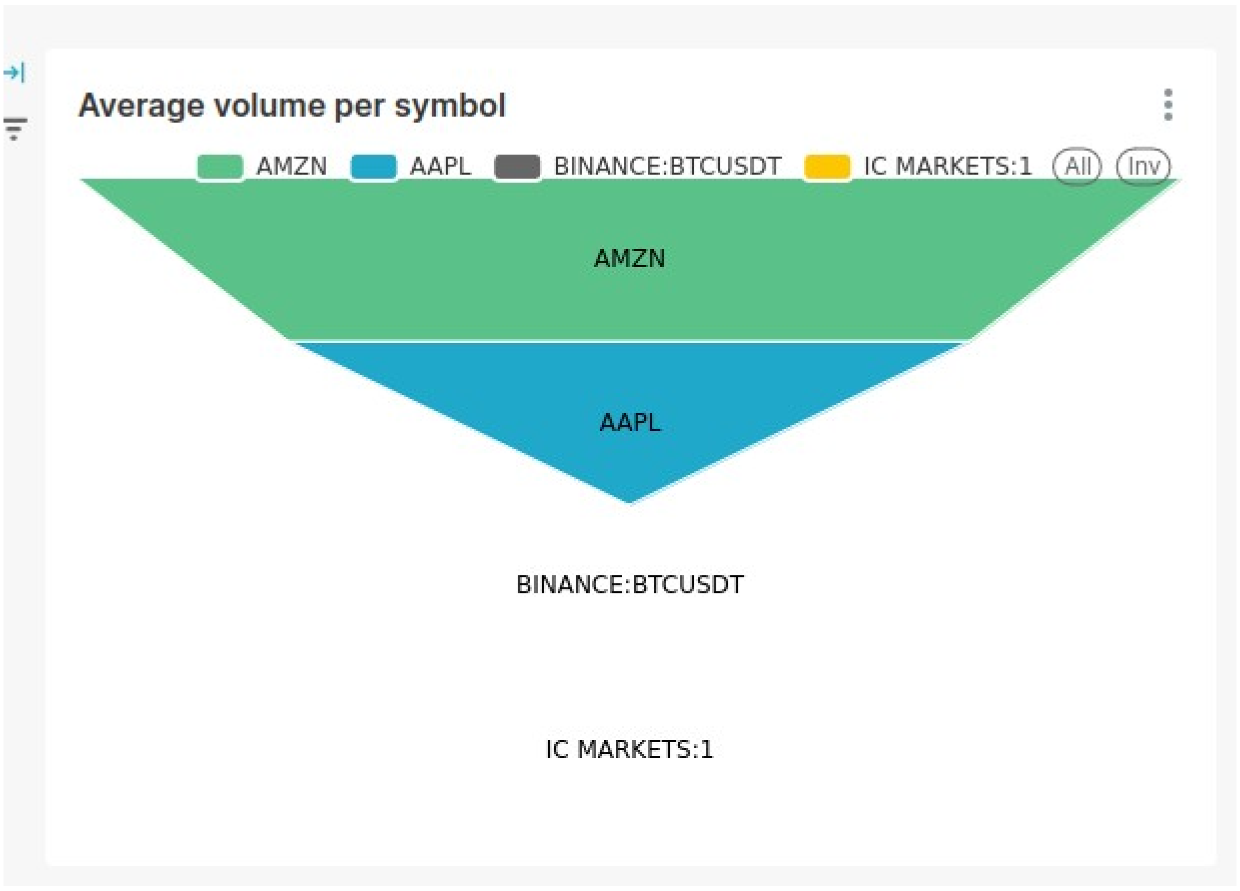

Firefox Web Browser
May 5 18:07
100 %
Digiicampus
Finnhub Stock APIs - Real
API Documentation | Finn
API Documentation | Finn
Trade data Analysis
Mail - Dhanya Shanbhag
localhost:8088/superset/dashboard/2/?native_filters_key=yE-P89pABXrp5iqcqNNl71oezA0U9nYHeMl98g2T1wwYqtYv-wSFKuefcqwC9NQh
Superset
Dashboards
Charts
Datasets
SQL
Settings
Trade data Analysis
Published
EDIT DASHBOARD
Average volume per symbol
AMZN
AAPL
BINANCE:BTCUSDT
IC MARKETS:1
Rate of trade data per second
SUM(transactions_per_second)
Count of transactions
Timestamp
Amount of data arrived for each symbol
Average price of various symbols
AVG(price)
Price in dollars
Symbols
Data-loss analysis via waterfall chart
Unique symbols arriving (out of 4) per 30 second, Timestamp varying for 30 seconds.
Symbolwise min-max price
MAX(highest_price)
MIN(lowest_price)
Price
Symbol
Volume trend of BINANCE:BTCUSDT
MAX(total_volume)
Mean time of process uptil writing to database

median_process_time
0 days 00:03:58

Image3

The KPI’s that are visible on the dashboard (image3) are calculated as below:

1. Average volume per symbol [Funnel chart]:



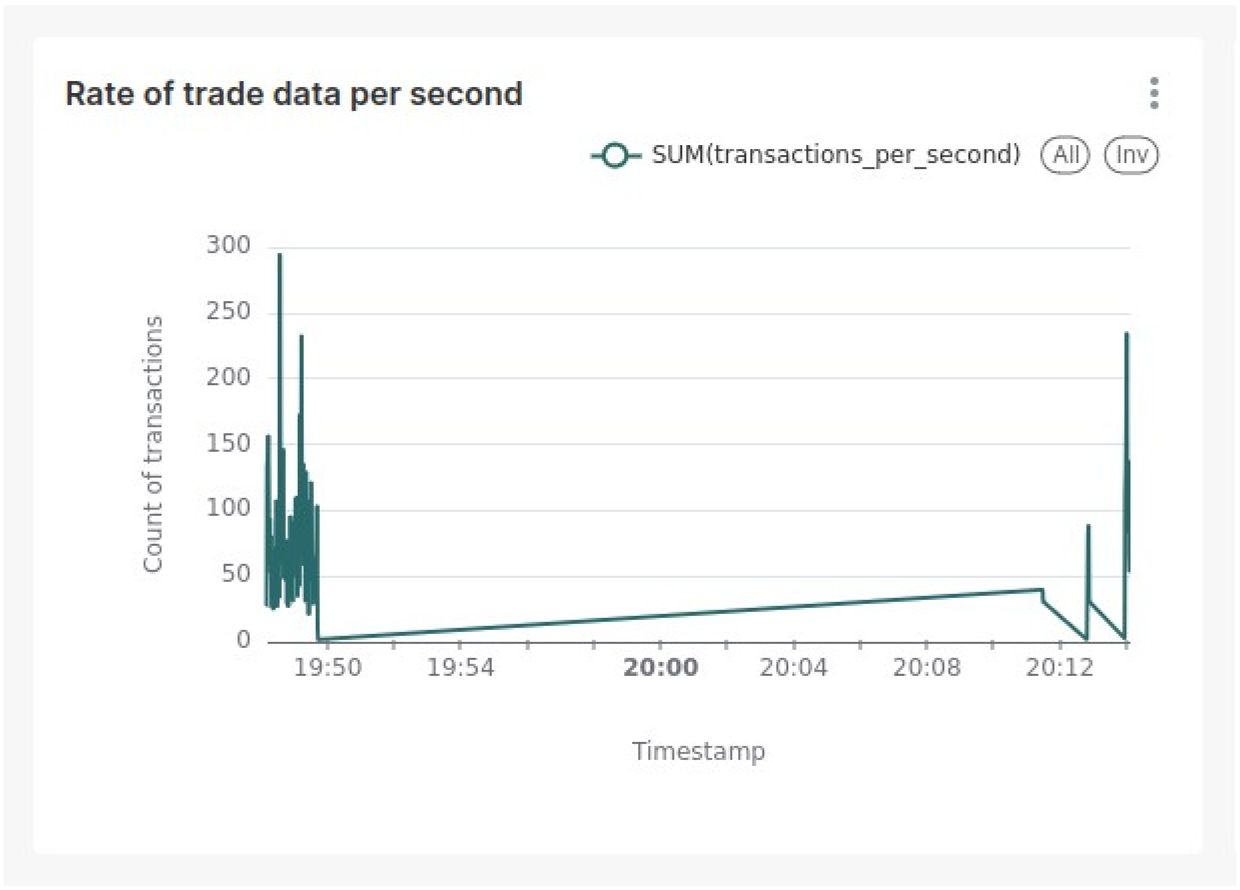
KPI = Average volume per symbol = AVG(volume) and group by symbol.

This funnel chart shows the highest average volume is for AMZN followed by AAPL. But BINANCE:BTCUSTD and IC MARKETS:1 have the average volume nearing to zero. To detect the reason behind, a volume trend analysis is done for the BINANCE:BTCUSTD in the chart number 7.

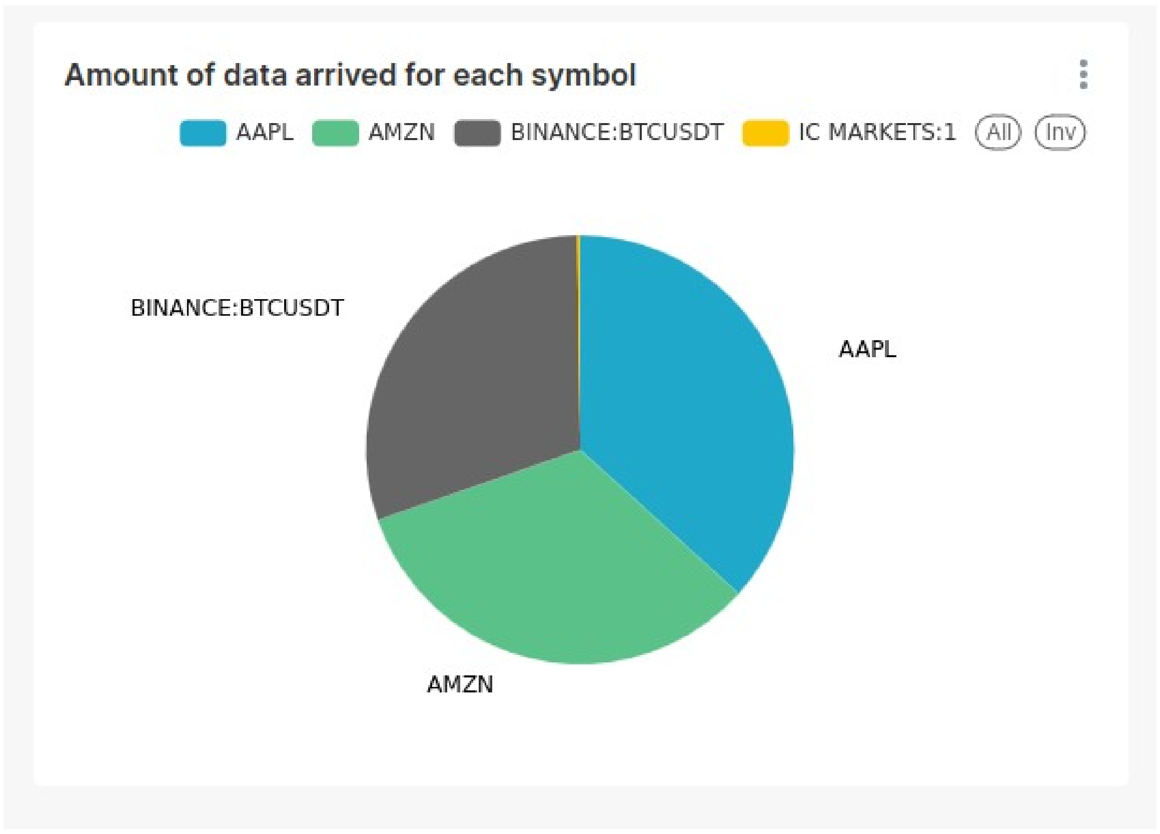
2. Rate of trade data per second [Line chart]:

KPI = Rate of trade data per second = SUM (Transactions_per_second)

Timestamp column is taken in the x-axis and the count of transactions is taken on the y-axis. A line chart is plotted to view the amount of data that is arriving per second. The detailed view of this can be viewed in the image shown below.



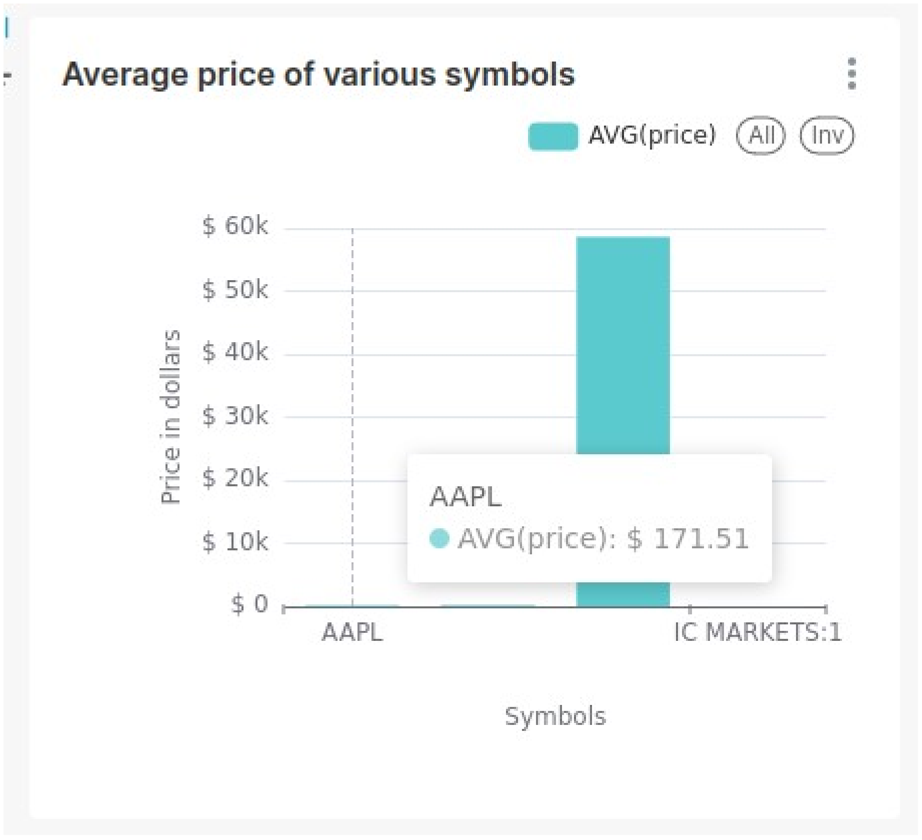
3. Amount of data arrived for each symbol [Pie chart]:



KPI = Amount of data arrived for each symbol = SUM(id), group by symbols

The pie chart clearly tells that even though 4 symbols were subscribed, the amount of data related to IC MARKETS:1 is drastically less. This accounts for potential data loss.

4. Average price of various symbols [Bar graph]:



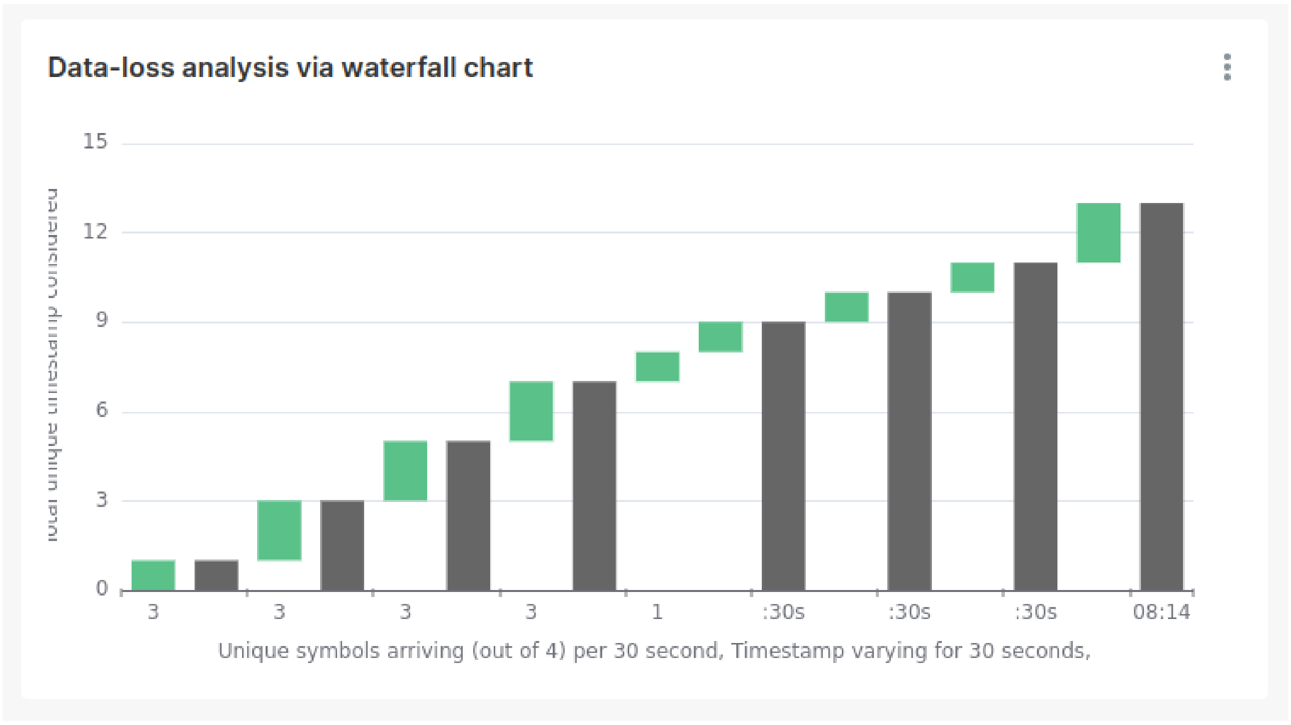
KPI = Average price of various symbols = AVG(Price), groupby symbol

Symbols is taken in the x-axis and Price in dollars is taken in the y-axis. A bar graph is plotted to calculate the average price of each of the symbols. The BINANCE:BTCUSTD has very high average price wheras other symbols like AAPL have a very low average price comparitively.

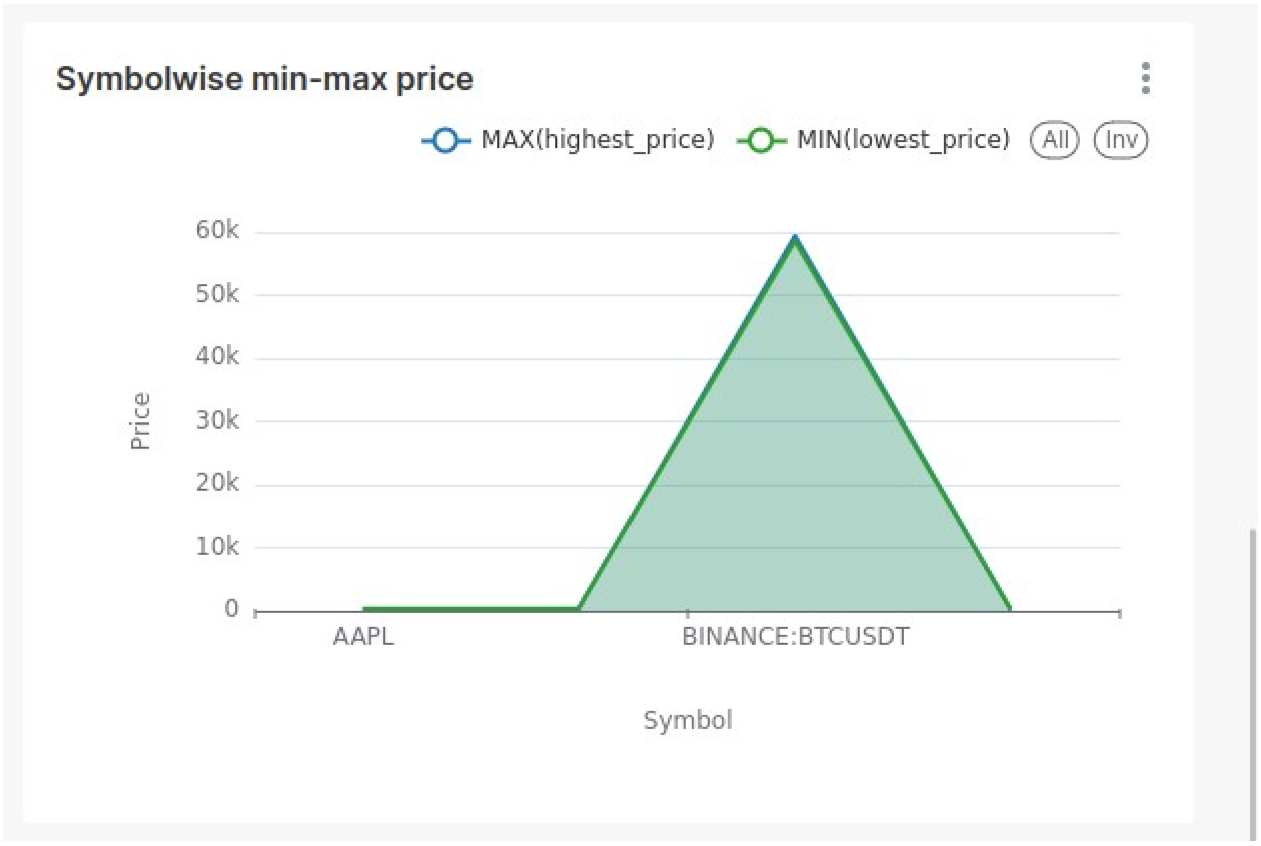
5. Data-loss analysis via waterfall chart:

KPI = Data-loss analysis via waterfall chart = groupby (unique_symbol_count) per 30 seconds.

A threshold is set to 30 seconds, and how many unique symbol related data are arriving is counted. The green bar count on the x-axis indicates the number of unique symbols out of 4. The x-axis in general is the timestamp per 30 seconds. It can be seen that 2024-05-02 20:11:30 and 2024-05-02 20:11:30 have only one and two symbols data respectively.



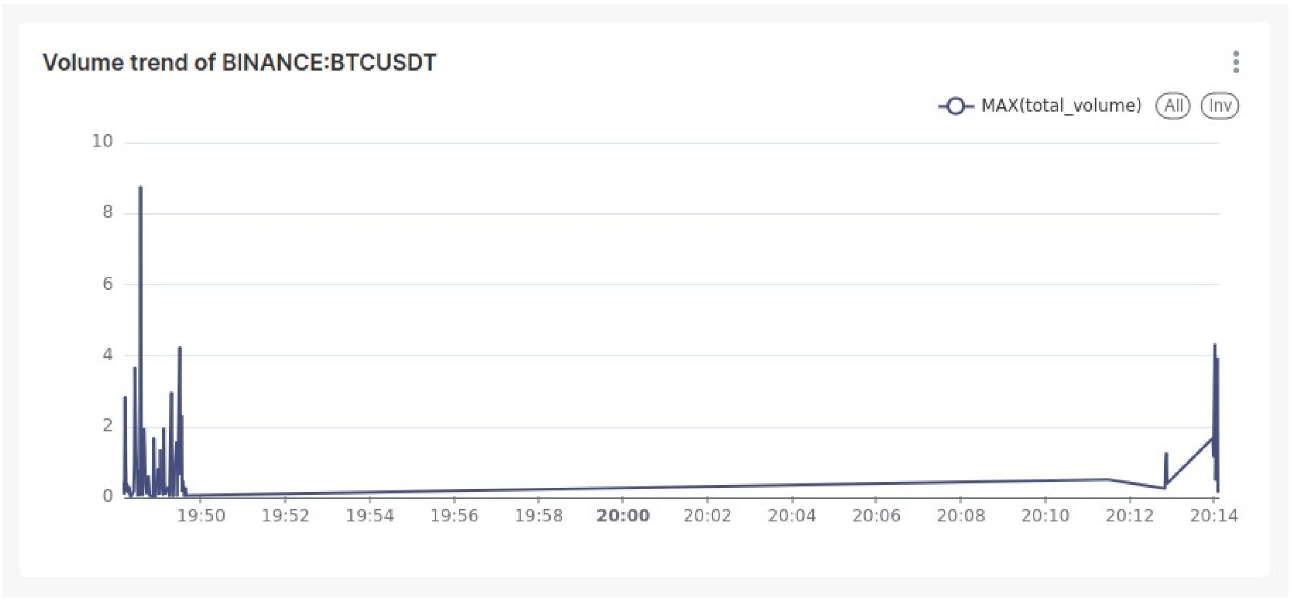
6. Symbolwise min-max price [Area chart]:



KPI = Symbolwise min-max price = MIN(Price), MAX(Price), groupby symbol

The area chart depicts two trends, i.e., minimum price line (green color), and maximum price line (Blue color). We can clearly see that both the lines are almost coinciding each other. This means that there is very less difference between the minimum and maximum price of every company. Also, we can see that the BINANCE:BTCUSTD has highest minimum and maximum price comparitively.

7. Volume trend of BINANCE:BTCUSTD [Line chart]:



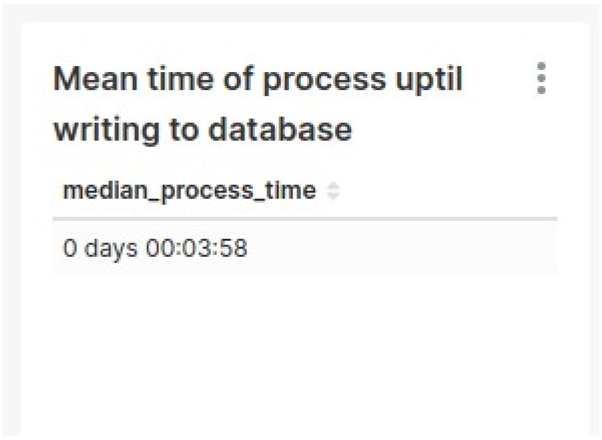
KPI = Volume trend of BINANCE:BTCUSTD = Volume across time per second

This particular analysis is done to find out why the average volume displayed using funnel chart for this particular symbol was having zero as value. This line chart of volume tren further makes it clear that majority of the volume data arriving is having zero as value which is influencing the total average.

8. Mean time of process uptil writing to database [Table chart]:

KPI = Mean time of process uptil writing to database = AVG(timestamp – to_db)

This particular metrics helps us know what is the mean time required to complete the whole process from the original timestamp of data till the data is written into the database. In this case it is 3 minutes and 58 seconds.



Mean time of process uptil writing to database

median_process_time
0 days 00:03:58