# **Real Time Analysis Of Bitcoin Network Congestion**

**Group 8:**

Tejaskumar Pareshbhai Patel - 885174433

Dhruv Ashokkumar Dhorajiya - 885177451

## Functionalities

**- Real-Time Data ingestion:**

- The application has the ability to process Bitcoin transaction data in real-time as it is being processed on the network. This involves trustworthy data source, integrated web-socket from:

- <https://www.blockchain.com/explorer/api/api_websocket>

- **Real-time data enrichment:**

- The application is enriched Bitcoin transaction data with additional contextual information, such as the location or type of the sending and receiving addresses. This can help identify patterns and trends in transaction fees based on different user behaviors or usage patterns.

- **Data processing:**

- Once the Bitcoin transaction data has been ingested, the application must be able to process the data in real-time using Spark Structured Streaming. This can involve a range of data processing tasks, such as filtering, aggregation, and transformation.

**- Windowing:**

- To calculate average Bitcoin transaction fees over time, the application has the ability to use windowing techniques to aggregate transaction data over specific time intervals. This can involve defining fixed length window over time stamps.

**- Statistical analysis:**

- Once the Bitcoin transaction data has been processed and aggregated, the application is able to perform statistical analysis to calculate the average transaction fee for each time interval. This can involve using Spark's built-in statistical functions, such as mean.

**- Visualization and reporting:**

- To communicate the results of the analysis, the application is able to visualize and report the average Bitcoin transaction fees over time. This involves using a frontend framework ReactJS and chart-js-2 library to generate live streaming dashboard.

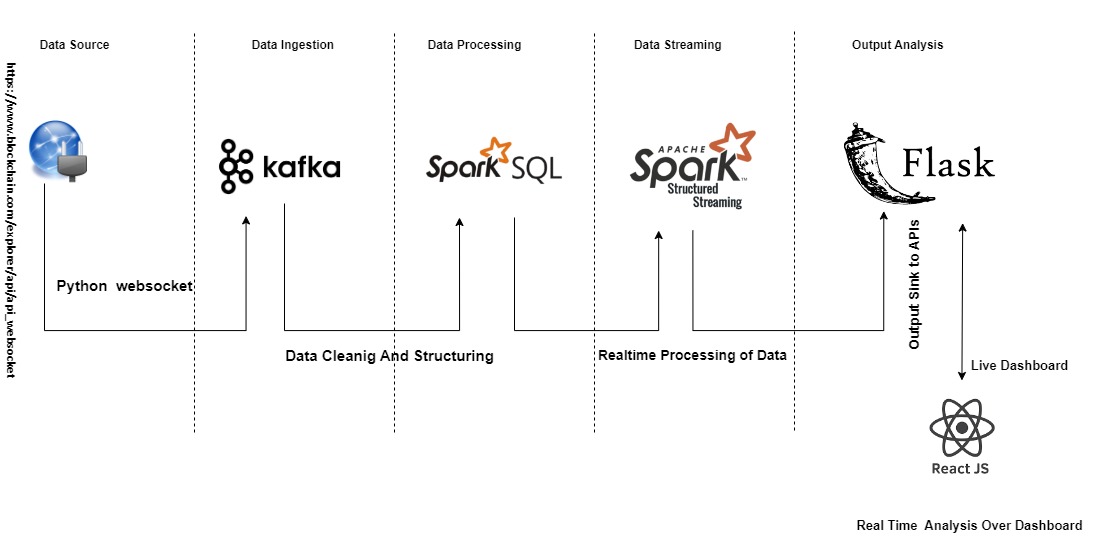
- **Integration with other data sources:**

- The application can be integrated with other data sources, such as social media or news feeds, to provide additional context for Bitcoin transaction data. This can help identify correlations between transaction fees and external events or trends.

- **Distributed processing: (i.e can be)**

- The application can be designed to process Bitcoin transaction data using a distributed computing architecture i.e cluster, such as Google Cloud Platform DataProc. This can help improve scalability and performance for large-scale data processing and analysis.

## Architecture & Design



**- Data Ingestion :**

- Get the Real Time bitcoin transactions, from endpoint :

- wss://ws.blockchain.info/inv

- **Project File Name :**

Web-socket.py

- **Tools and Tech :**

Kafka

Python Web Socket

- Usage of Python Websocket, to retrieve data from socket, OnMessage, push raw data to Kafka Producer with new topic : “bitcoin-1”.

- **Data Processing:**

- Consume the Kafka Topic, named: “bitcoin-1”.

- **Project File Name :**

Spark\_stream\_btc.py

- **Tools and Tech :**

SparkSQL

- Consumes raw bytes, convert to string , to JSON structured.

- Structure the raw data to JSON, convert to dataframe, with the pre-defned schema.

- Streaming applications needs to be windowed, to be run seamlessly and aggregation to be applied.

- **Data Streaming:**

- Real Time Streaming of Transaction fees , with spark writestream.

- **Project File Name :**

Spark\_stream\_btc.py

- **Tools and Tech :**

Spark Streaming

- Process data with streaming micro batch processing.

- Outmode to be complete, which aggregates over all the incoming data, since the spark job submit.

i.e streaming query execution.

- Finally, spark application will POST the processed data to flask, in real time via REST APIs.

- **Output Sink:**

- Spark streaming application sinks the processed data to Flask, via REST APIs.

- **Project File Name :**

Spark\_stream\_btc.py

App.py

- **Tools and Tech :**

Spark

Flask

- Frontend, Reactjs Charts calls the API to refresh the dashboard screen, repeatedly.

- **Data Analysis:**

- Frontend, calls the REST APIs of Flask, to update the chart, repeatedly, to get the new processed micro batch data output.

- **Project File Name :**

Src/BigData/Screens/BTCBarChart.jsx

- **Tools and Tech :**

ReactJS

Chart-js-2

- Processed data is loaded to chart, to render/re-render new metrics i.e real time streaming.

## GitHub Location of Code

- <https://github.com/tejs13/Spark-Streaming-BTC>

- Branch : main

## Deployment Instructions

- **Deployment Instructions for Local Mode:**

1. Install Spark:
   1. Install the Spark distribution on the local machine. You can download Spark from the Apache Spark website and follow the installation instructions.
   2. Version 3.2.3
2. Configure environment variables:
   1. Set the environment variables for Spark, such as SPARK\_HOME and PATH, PYSAPRK\_PYTHON, to point to the Spark installation directory and the Spark binary files.
3. Install Kafka
   1. Install Distributed message service, kafka.
   2. Version 3.2.3
4. Install system dependencies :
   1. Pyhton 3.10.9
   2. Java 9
   3. Node JS
5. Open port 5001, on local machine for communication between Spark and Flask and ReactJS Dashboard.
6. Make virtual environment and install the dependencies, i.e requirements.txt.

- pip install -r requirement.txt

1. Submit the Spark Structured Streaming application to local mode:
   1. Spark-submit spark\_stream\_btc.py

## Steps to Run the Application

1. Make virtual environment and install the dependencies, i.e requirements.txt.

- pip install -r requirement.txt

1. Start services:
   1. Zookeeper
      1. .\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties
   2. Kafka
      1. .\bin\windows\kafka-server-start.bat .\config\server.properties
2. Start capturing the real time bitcoin transactions with web socket and publishing to kafka topic “bitcoin-1”
   1. Python web-socket-kafka.py
3. Submit the spark streaming application to run in local mode:
   1. Spark-submit spark-stream-btc.py
4. Start the Flask interface application for spark job to sink the output via REST APIs.
   1. Python app.py
5. Start the react application, for real time dashboard:
   1. cd amplify-react
   2. npm start

## Test Results

1. Unit Testing:
2. Integration Testing:
3. Performance Testing: