Get Real-time data streams for your TRON projects using Bitquery - A Beginner Friendly guide

Bitquery supports TRON ecosystem in both historical data as well as with real time streams (V2 Streaming APIs).With only a sub second delay one has access to realtime TRON mempool and confirmed data.

Let's see how to implement real time data for your TRON project. Developer plan gives a limited access to work on your Individual hobby projects and for a large scale application commercial plan suits better.

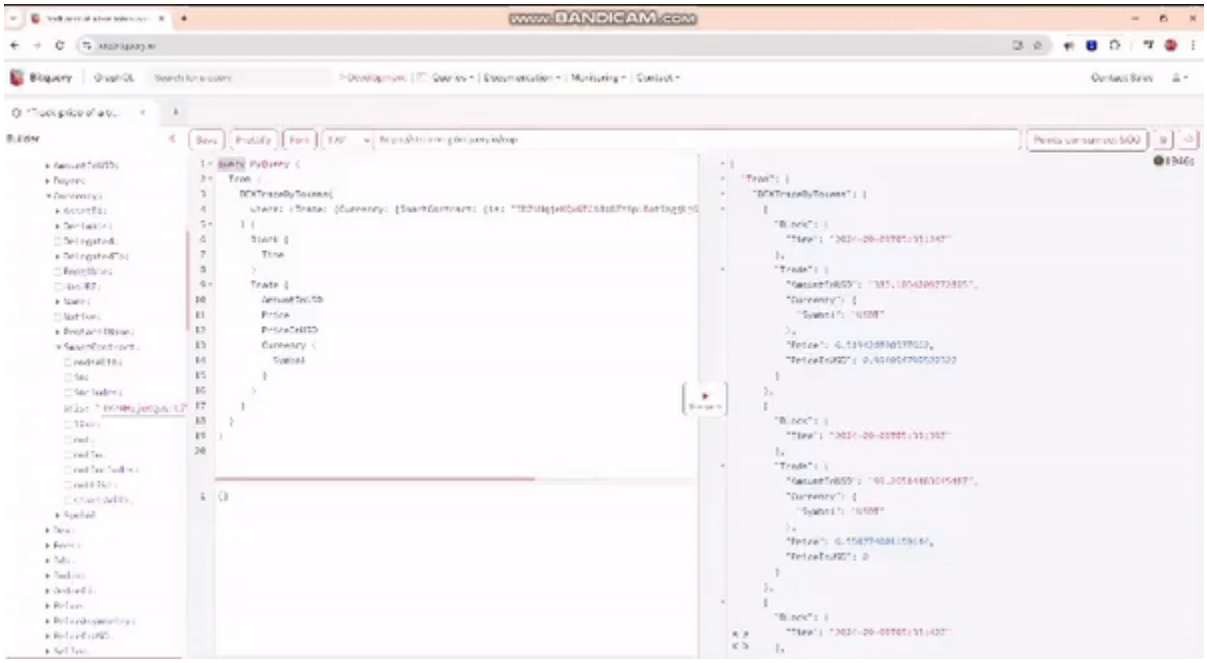
Pre-requisites:

- GraphQL subscription. Frame a subscription query in Bitquery IDE.
    - Streaming API to use in your application
        - **WebSocket**: `wss://streaming.bitquery.io/eap`
        - **HTTPS**: `https://streaming.bitquery.io/eap`

- Create an access token for your application.Use the key for authentication. Post key generation, check in Postman whether you are able to retrieve data for your key here.

Example Query: Track Real-time TRON Token Price

Check if you're able to retrieve the data on the IDE.



Let's execute the same query in both React and Python projects.

React Implementation

Dependencies: axios library

Step 1: Create a Trondata component and write a fetchData function

```
function Trondata() {
    const [trondata, setTRONData] = useState([]);

    const fetchData = () => {
        const  query = `***TRON Subscription Query Here***`;

        const data = JSON.stringify({ query });

        const config = {
            method: 'post',
            url: 'https://streaming.bitquery.io/eap',
            headers: {
            'Content-Type': 'application/json',
            'Authorization': "Bearer
ory_at_6jdb37d-BpnmhaJIPxW0f0iR6xRNzxZhdNWIcajq7Q8.N1RSoyojS3rcCzNgrrfvlig4yJdDdq
0FO2tnU5cV4yA",
            },
            data: data
        };

        axios(config).then((res) => {
            setTRONData(res.data.data.Tron.DEXTradeByTokens); // Assuming setData
is defined elsewhere
        }).catch((error) => {
            console.error('Error:', error); // Use console.error for error
messages
        });

    }
```

Step 3: Call the function in useEffect

```
useEffect(() => {
        fetchData();
    },[]);
```

Step 4: Map your Tron data onto your UI

```
return (
        <div>
          <h1>Real Time TRON Token Price</h1>
          <table>
            <thead>
                <tr><td>Time</td><td>Currency</td><td>Amount In USD</td><td>Price
In USD</td></tr>
            </thead>
            <tbody>
                  {trondata.map((item, index) => (
                    <tr key={index}>
                      <td>{item.Block.Time}</td> <td>{item.Trade.Currency.Symbol}
</td><td>{item.Trade.AmountInUSD}</td> <td>{item.Trade.PriceInUSD}</td>
                    </tr>
                  ))}
            </tbody>
        </table>
        </div>
      );
```

You have your TRON realtime data on your react application

Real Time TRON Token Price

Time	Currency	Amount In USD	Price In USD
2024-09-11T12:24:15Z	USDT	2294.9634395887756	1.0036579676346677
2024-09-11T12:24:51Z	USDT	555.0169372558594	1.006049677553198
2024-09-11T12:25:03Z	USDT	2896.4900251631466	0.9990189652403403
2024-09-11T12:25:15Z	USDT	1798.0037170444336	1.0046042037877403
2024-09-11T12:25:15Z	USDT	450.01373291015625	1.0024546417193658
2024-09-11T12:26:24Z	USDT	34965.71208628693	0
2024-09-11T12:28:21Z	USDT	32.50115841627121	0
2024-09-11T12:28:21Z	USDT	33.64646123750615	1.223711200801909
2024-09-11T12:29:21Z	USDT	36.00128316879272 5	0.99646169485033 8

Python Implementation

Step 1: Importing the needed libraries gql Client, pandas and Websockets. Optional modules to display data are table and colorama

Step 2: Setup websocket connection with connect using GraphQL

```
    async def run_subscription():
      transport = WebsocketsTransport(
        url="wss://streaming.bitquery.io/eap?token= Your OAuthToken Here",
        headers={"Sec-WebSocket-Protocol": "graphql-ws"})

    # Establish the connection
    await transport.connect()
    print("Connected to WebSocket")

      try:
            while True:
            async for result in transport.subscribe(
                    gql("""Your Query Here""")):
                if result.data:
                   new_data =
pd.json_normalize(result.data['Tron']['DEXTradeByTokens'])
                   new_data = new_data.reindex(columns=expected_columns)

                   if tron_price.empty:
                        tron_price = new_data
                    else:
                        tron_price = pd.concat([tron_price, new_data],
ignore_index=True)
                   table = tabulate(formatted_rows, headers=colored_headers,
tablefmt='pretty', showindex=False)

      finally:
        await transport.close()

      def main():
            asyncio.run(run_subscription())
```

Step 3: Print the result.data on a formatted table to see the real-time data printed onto the console.

```
(venv) C:\Users\srees\OneDrive\Desktop\Bitquery\python-project>python main.py
Connected to WebSocket
RealTime TRON Token Price
+----------------------+--------------------+-----------------------+--------------+--------------------+
|      Block.Time      | Trade.AmountInUSD  | Trade.Currency.Symbol | Trade.Price  |  Trade.PriceInUSD  |
+----------------------+--------------------+-----------------------+--------------+--------------------+
| 2024-09-10T05:08:36Z | 2000.3139972686768 |          USDT         | 6.4773007335 | 0.9930963675357687 |
+----------------------+--------------------+-----------------------+--------------+--------------------+
```

Now you've implemented real-time data on your TRON project.

Cloud Products

TRON blockchain data is published on Cloud products like AWS S3 and CLI, Snowflake, Azure and Google Cloud and can be directly accessed by the apps deployed onto your cloud solutions.

Self Help

You can also check out our pre-built queries on our TRON explorer or on our documentation. Replace the "*query*" keyword to "*subscription* to fetch real time data to integrate to your current project.

References

Streaming API Demo Link

Bitquery documentation: How to build dApps?

Discord Community: TRON Developers and SRs

Add-Ons

We also provide raw blockchain data on Discord and Telegram Bots where you can do a seamless integration to your applications.

Stuck in a step?

If you get stuck on a query or an implementation, DM on our Telegram channel where we assist you on queries and your projects.

Real time support for your projects

We also do custom integrations with your existing Cloud infrastructure (AWS, Google Cloud), Data Warehouse products.

You can also contact us on our form for your exact specific needs or send us a mail to sales@bitquery.io.