upstox

Trade Test

Problem

Create an Analytical Server "OHLC" (Open/High/Low/Close) time series based on the 'Trades' input dataset.

The ‘Trades’ dataset is available at http://kaboom.rksv.net/trades-test/trades-data.zip

Input

a. Attached please find trades.json of historical trades for a set of symbols.

   Below are samples from the JSON file for ease of reference

```
{"sym":"XETHZUSD", "T":"Trade", "P":226.85, "Q":0.02,
"TS":1538409733.3449, "side": "b", "TS2":1538409738828589281}
{"sym":"XETHZUSD", "T":"Trade", "P":226.85, "Q":4.98,
"TS":1538409733.3502, "side": "b", "TS2":1538409738828643608}
{"sym":"XXBTZUSD", "T":"Trade", "P":6538.8, "Q":1,
"TS":1538409739.0962, "side": "s", "TS2":1538409748332169137}
{"sym":"XXBTZUSD", "T":"Trade", "P":6538.2, "Q":0.498558,
"TS":1538409739.1111, "side": "s", "TS2":1538409748332223252}
```

b. Below are the keys and their associated meanings:

```
struct barOHLC {
      sym : Stock name string
      T: Ignore this field string
      P: Price of Trade double
      Q: Quantity Traded double
      TS: Ignore this field uint64
      Side: Ignore this field string
      TS2: Timestamp in UTC uint64
};
```

Detailed Problem Charter

a. Use one of the following languages: Java, NodeJS, C++.

b. Create a multi-worker/multi-threaded (3) console program that has achieved the desired result.
   i. **Worker_1 (Thread 1)**: Reads the Trades data input (line by line from JSON), and sends the packet to the FSM (Finite-State-Machine) thread.
   ii. **Worker_2 (Thread 2)**: (FSM) computes OHLC packets based on 15 seconds (interval) and constructs 'BAR' chart data, based on timestamp TS2. (ignore TS)

iii. **Worker_3 (Thread 3)**: (WebsocketThread) Client subscriptions come here. Maintains client list, and publishes (transmits) the BAR OHLC data as computed in real time.

c. Additional Design Criteria

i. The 15-second bar starts on the first trade, and maintains the bar_num series.
ii. Every bar is identified by its bar_num attribute, starting at 1, and incrementing.
iii. The 15-second bar closes upon the expiration of the bar-interval.
iv. The next 15-second bar starts with bar_num++ as its identifier
v. Don't wait for the next trade to start the next bar.
vi. You can have empty bars during a 15 second interval! (Acceptable)

Output

1. Either have OHLC data sent on EVERY trade.

OR

2. Send the OHLC "ONLY" when the bar closes.

Example

a. User Subscription

```
{"event": "subscribe", "symbol": "XXBTZUSD", "interval": 15}
```

b. Response:

i. c (close) will remain EMPTY until the 15 second bar closes
ii. bar_num (bar number) is incremental
iii. volume denotes the total quantity aggregated within that 15 sec interval when the bar closed

```
{"o":6538.8,"h":6538.8,"l":6538.8,"c":0,"volume":1,"event":"ohlc_notify","symbol":"XXBTZUSD","bar_num":1}
{"o":6538.8,"h":6538.8,"l":6538.2,"c":0,"volume":1.498558,"event":"ohlc_notify","symbol":"XXBTZUSD","bar_num":1}
{"o":6538.8,"h":6538.8,"l":6537.9,"c":0,"volume":4.556558,"event":"ohlc_notify","symbol":"XXBTZUSD","bar_num":1}
{"o":6538.8,"h":6538.8,"l":6537.7,"c":0,"volume":4.999999,"event":"ohlc_notify","symbol":"XXBTZUSD","bar_num":1}
{"o":6538.8,"h":6538.8,"l":6537.7,"c":0,"volume":4.99999942,"event":"ohlc_notify","symbol":"XXBTZUSD","bar_num":1}
{"o":6538.8,"h":6538.8,"l":6537.7,"c":6537.7,"volume":4.99999942,"event":"ohlc_notify","symbol":"XXBTZUSD","bar_num":1}
```

Note: At the 15 sec boundary, the bar closes and a new bar starts which stays empty as no trade is placed in that interval.

```
{"event":"ohlc_notify","symbol":"XXBTZUSD","bar_num":2}
```

Then a new bar starts at the next 15 second interval

```
{"event":"ohlc_notify","symbol":"XXBTZUSD","bar_num":3}
{"o":6537.7,"h":6537.7,"l":6537.7,"c":0,"volume":0.556558,"event":"ohlc_notify","
symbol":"XXBTZUSD","bar_num": 3}
....
```

Optional

a. If the client-side WebSocket is not done, it’s OK. Make sure the program LOGS (prints) OHLC output clearly for us to verify.

b. Use `wscat -c ws://localhost:port` followed by the subscribe event (initiating client-websocket).

c. Ability to collect Performance-Statistics within the Program is a Bonus.

Submission

1. Good `README.txt`: Simple Design Criteria, methods, workers, data structures etc.
2. Instructions to setup and run the solution on a LINUX environment (preferred).
3. Unit Tests.
4. Decent package structure.
5. Good code documentation.
6. Maven/gradle or any other tool.
7. Exception handling.
8. Clean code (no commented code / no warnings)