special-pancake

Group 1:
Brian, Felicia Faustina Victoria Santoso,
Muhammad Syahmi Bin Abbas

Table of contents

Architecture

REST API and Microservice Architecture

Languages and Libraries

Redis for low-latency access, rapid execution!

Extra: Resilience with redis sentinel setup!

Python for simplicity, universal language!

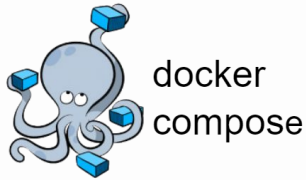
Gunicorn production server

Kubernetes

Extra: To allow for quick scaling and load balancing for high amount of users

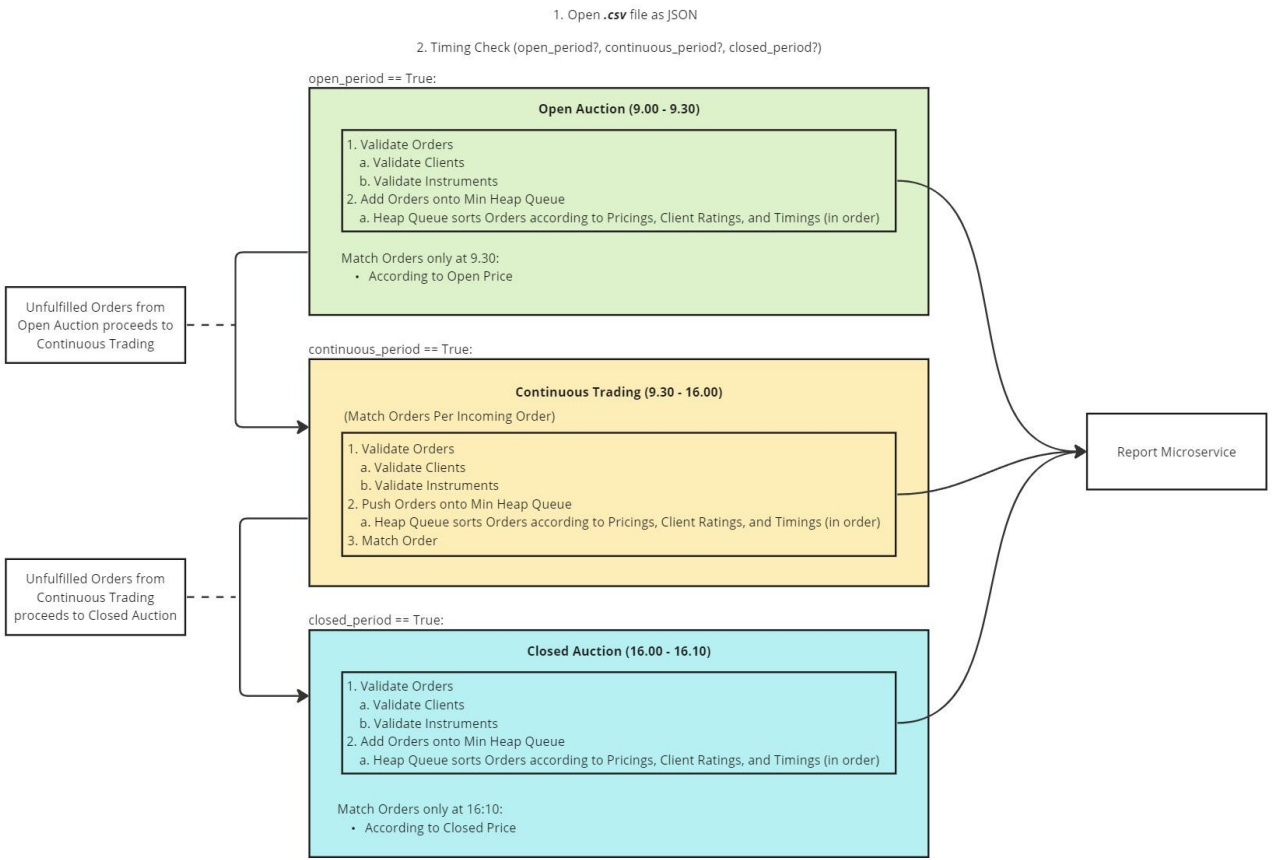
Docker

Containerize our Redis and application



One command to run our server and containers

Design


1. Open .csv file as JSON
2. Timing Check (open_period?, continuous_period?, closed_period?)
open_period == True:
Open Auction (9.00 - 9.30)
1. Validate Orders
a. Validate Clients
b. Validate Instruments
2. Add Orders onto Min Heap Queue
a. Heap Queue sorts Orders according to Pricings, Client Ratings, and Timings (in order)
Match Orders only at 9.30:
• According to Open Price
Unfulfilled Orders from Open Auction proceeds to Continuous Trading
continuous_period == True:
Continuous Trading (9.30 - 16.00)
(Match Orders Per Incoming Order)
1. Validate Orders
a. Validate Clients
b. Validate Instruments
2. Push Orders onto Min Heap Queue
a. Heap Queue sorts Orders according to Pricings, Client Ratings, and Timings (in order)
3. Match Order
Unfulfilled Orders from Continuous Trading proceeds to Closed Auction
closed_period == True:
Closed Auction (16.00 - 16.10)
1. Validate Orders
a. Validate Clients
b. Validate Instruments
2. Add Orders onto Min Heap Queue
a. Heap Queue sorts Orders according to Pricings, Client Ratings, and Timings (in order)
Match Orders only at 16:10:
• According to Closed Price
Report Microservice

Design

1. Open ***.csv*** file as JSON

2. Timing Check (open_period?, continuous_period?, closed_period?)

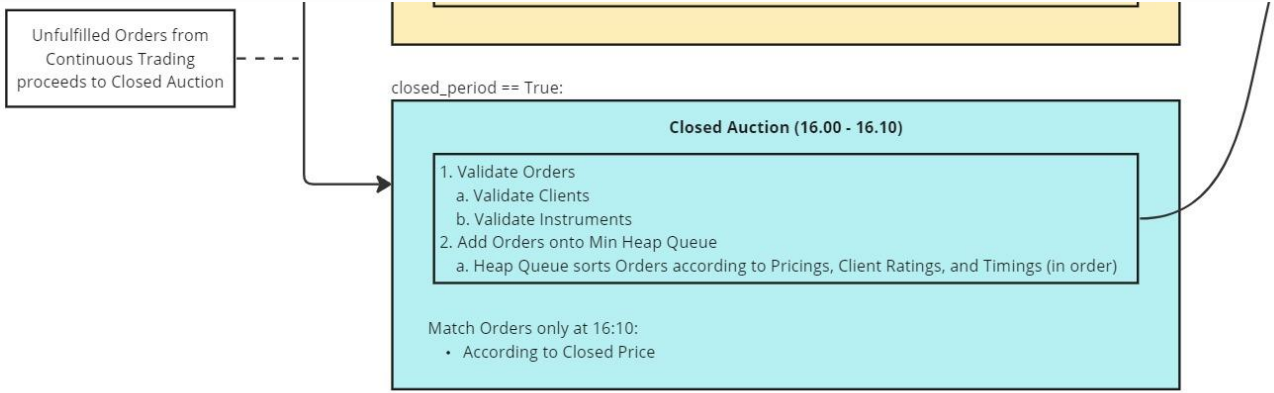
Design


open_period == True:
Open Auction (9.00 - 9.30)
1. Validate Orders
a. Validate Clients
b. Validate Instruments
2. Add Orders onto Min Heap Queue
a. Heap Queue sorts Orders according to Pricings, Client Ratings, and Timings (in order)
Match Orders only at 9.30:
• According to Open Price
Unfulfilled Orders from Open Auction proceeds to Continuous Trading
continuous_period == True:
Continuous Trading (9.30 - 16.00)
(Match Orders Per Incoming Order)
1. Validate Orders
Report Microservice

Design


Unfulfilled Orders from Open Auction proceeds to Continuous Trading
continuous_period == True:
Continuous Trading (9.30 - 16.00)
(Match Orders Per Incoming Order)
1. Validate Orders
a. Validate Clients
b. Validate Instruments
2. Push Orders onto Min Heap Queue
a. Heap Queue sorts Orders according to Pricings, Client Ratings, and Timings (in order)
3. Match Order
Report Microservice
Unfulfilled Orders from Continuous Trading proceeds to Closed Auction
closed_period == True:
Closed Auction (16.00 - 16.10)

Design


Unfulfilled Orders from Continuous Trading proceeds to Closed Auction
closed_period == True:
Closed Auction (16.00 - 16.10)
1. Validate Orders
a. Validate Clients
b. Validate Instruments
2. Add Orders onto Min Heap Queue
a. Heap Queue sorts Orders according to Pricings, Client Ratings, and Timings (in order)
Match Orders only at 16:10:
• According to Closed Price

Consolidated Reports:

We strive to provide clear reports; our reports are hosted on secure, REST endpoints →

1. **POST** /update_exchange_report
2. **POST** /update_client_report
3. **POST** /update_instrument_report
4. **GET** /report

With a reports microservice, we are able to compartmentalise, and move the compute to a separate service.

We can also scale up the report microservice, without causing any downtime. Thanks to the independent operation of each service.

A simple frontend is also created for users to view the reports.

Tests:

We also made use of unit tests:

```
(matching) brianchew@Brians-MacBook-Pro matching % python3 report_test.py
.Report Type:  exchange_report
Details:  [{'order_id': 1, 'reason': 'insufficient_balance'}]
Exported exchange_report to CSV.
.[{'client_report': [{'client_id': 1, 'order_id': 1, 'reason': 'invalid policy'}]}]
.[{'client_report': [{'client_id': 2, 'order_id': 20, 'reason': 'invalid policy'}]}]
.[{'exchange_report': [{'order_id': 1, 'reason': 'insufficient_balance'}]}]
.[{'instrument_report': [{'instrument_id': 'SIA', 'open_price': 30, 'closed_price': 43,
 'total_traded_vol': 400, 'day_high': 43, 'day_low': 29, 'vwap': 142, 'timestamp': 1715
582657}]}]
.[{'instrument_report': [{'instrument_id': 'INST_010', 'open_price': 50, 'closed_price'
: 60, 'total_traded_vol': 500, 'day_high': 60, 'day_low': 49, 'vwap': 155, 'timestamp':
 1715582658}]}]
.
----------------------------------------------------------------------
Ran 7 tests in 0.007s

OK
(matching) brianchew@Brians-MacBook-Pro matching %
```

Main Considerations

Client Validation

- Rating, sorted by Heap Queue
- PositionCheck, amount of asset owned

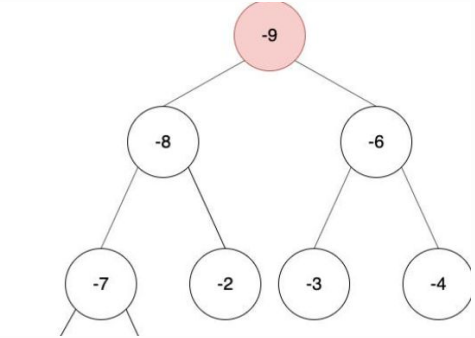
Instrument Validation

- Check if Order's InstrumentID exists
- Check if Client Currency corresponds to valid Instrument Currency

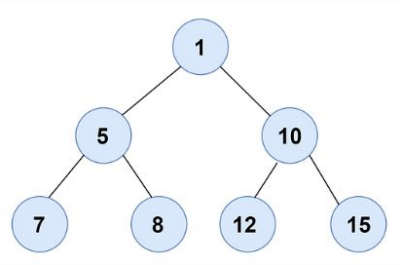
Differentiate Auction vs Continuous

- Makes use of reusable Heap Queues
- Differentiated by evaluation of match_orders for every addition of orders OR by Open/Close Price

Highest Price Buyer

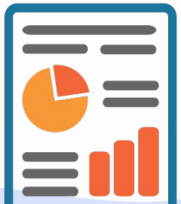


Lowest Price Seller



REST API Endpoints for Reports

- Calls report s by REST APIS for dynamic generation of reports



Thank you