VRIJE
UNIVERSITEIT
BRUSSEL

CLOUD COMPUTING REPORT

Gérard Lichtert
0557513
gerard.lichtert@vub.be

December 30, 2024

Sciences and Bio-Engineering Sciences

Contents

1 Architecture and choices **2**
- 1.1 Client Gateway . . . 2
- 1.2 Order Manager . . . 2
- 1.3 Matching Engine . . . 2
- 1.4 Market Data Publisher . . . 3
- 1.5 Exchange Dashboard . . . 3

2 Deployment instructions **3**

1 Architecture and choices

Prior to beginning with the actual components of the project there is some information that should be mentioned. All but the Matching engine use TypeScript, however it is compiled to JavaScript when containerized and all components but the Exchange Dashboard use Fastify for http communications. I chose Fastify because it is one of the fastest http libraries available in Node. I make use of a MySQL database to move the state from stateful applications to the database as well as keep track of the state of the orders.

1.1 Client Gateway

For the Client Gateway, as mentioned earlier, I use Fastify to create a http server. This libary also allows for requests to be pre-validated as they come in using fluent-json-schema, denoting the required types of the incoming request and returning a Bad Request response otherwise. The requests that are validated are forwarded to the Order Manager. Note that when the cluster is deployed, this application is scaled horizontally and is behind a load balancer to, well, balance the load between the pods.

1.2 Order Manager

To implement the Order Manager I made use of Fastify as my http server and I also make use of mysql2/promise, which is a library allowing me to connect to a database and execute queries asynchronously. When the Order Manager receives a request, it strips and inserts the order in the database. With the returned secnum of the order, the order is then forwarded to the Matching Engine as well as the Market Data Publisher. When the order is saved in the database, it returns a 201 response to the gateway, which returns the same status to the client. This application can also be horizontally scaled as its state is now delegated to the database.

1.3 Matching Engine

Just like the previous applications, the Matching Engine also makes use of the Fastify library to run a http server. This application is implemented in JavaScript because otherwise I had to do type gymnastics due to the Engine being implemented in JavaScript. The matching engine receives orders from the Order Manager sends it to a Subject, which in turn executes the order in the Engine, matching it with existing orders. Whenever orders are matched, it collects the executions, grouping them by secnum, updates the order in the database and sends these 'executions' to the Order Manager, which in turn forwards them to the Market Data Publisher. Since there is only one instance of the Matching Engine running, it has a recovery system, should it crash. It retrieves the unfullfilled orders and places them back in the subject, whilst keeping track of the orders currently in the Engine. Should an order be processed twice, it will not impact the state of the orders, since there's a constraint on the database keeping the amount left on the order in check.

1.4 Market Data Publisher

1.5 Exchange Dashboard

2 Deployment instructions