Cloud Computing Project Assignment 2024-2025

Jens Nicolay (jens.nicolay@vub.be), Angel Luis Scull Pupo (ascullpu@vub.be)

Scenario

You are now part of our engineering team, which is busy building the next-generation stock exchange infrastructure to handle trades on a growing stock market. The fundamental function of a stock exchange is to match trade orders of buyers (*bids*) and sellers (*asks*) accurately and as fast as possible. Another important aspect of the exchange is to give users visualization tools such as the status of the order book and price history of the traded stock symbols, as shown in Figure 1.

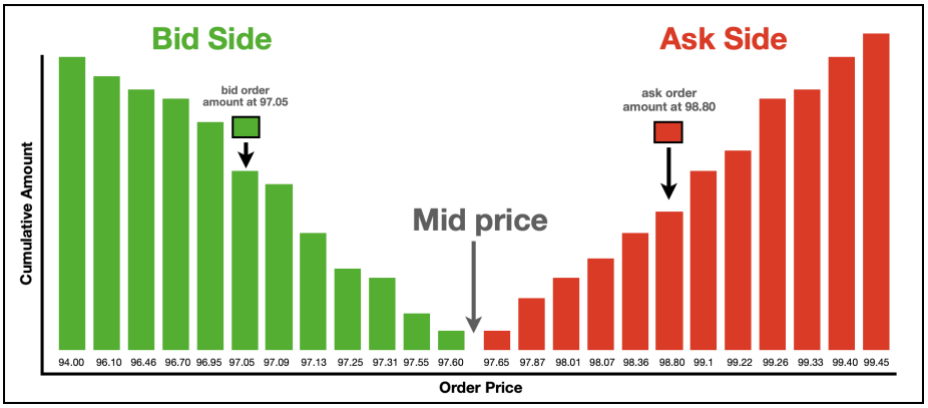

Figure 1. Order Book graph example.

We have already implemented the Matching Engine, a critical component of the system on a Kubernetes cluster. You have been tasked with building the rest of the system. Specifically, we want your help for implementing the critical path of such a stock exchange such that orders are matched as fast as possible. You are also required to implement a reporting dashboard enabling users to observe the behavior of the market in near real-time. Our reputation is very important. Therefore, you must ensure that the exchange is always available and that users do not perceive any slowdown despite the system load. In Figure 2 we show a logical view of the system's architecture.

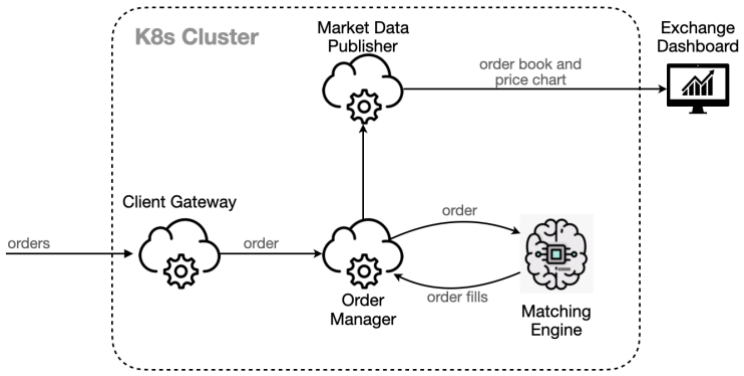

Figure 2. Logical Architecture.

Functional requirements

1- **Client Gateway**. This is a *stateless* service that takes orders (asks or bids), type-checks the order fields and forwards the order to the **Order Manager** service.
2- **Order Manager**. This *stateful* service performs two operations:

a. First, all received orders are stripped from the user_id, trader_type, time and get attached a sequential number. After, orders are sent to the Matching Engine and to the **Market Data Publisher**. When the Matching Engine finds a match (i.e., ask order(s) with lower or equal price to any bid order(s)), it emits two sets of order fills (executed orders). If no match is instantly found, the order is kept in the order book until a match is found.
b. Second, the executions (order fills) emitted by the matching engine are received by the **Order Manager** that sends these executions to the **Market Data Publisher** service.

3- **Market Data Publisher**. This *stateful* service receives orders and executions from the **Order Manager**. It builds and maintains the order book from the original orders and order executions (fills). This service then publishes the necessary data to any subscribed reporting application.
4- **Exchange Dashboard**. This is an instance of a data reporting application that receives data from the **Market Data Publisher**.
   a. It should visualize the order book status in real-time for any of the traded stock symbols that the user can select from a list. Placed orders should be used to increment the number of bars (or their sizes) in the book. Order fills (executions) should be used to decrease the number of bars (or their sizes) on the book.
   b. Is should display a graph of the daily evolution of the average price for asks and bids of the selected stock symbol like Figure 3.

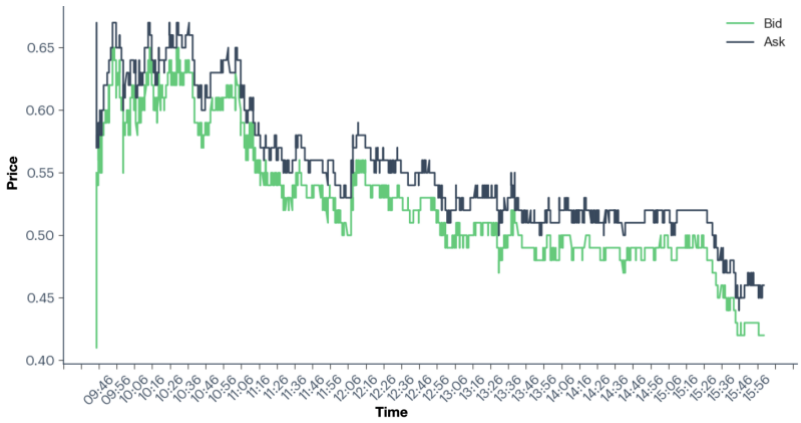

Figure 3. Asks and Bids daily price evolution graph.

Non-functional requirements

1- **Infrastructure**. All services mentioned above, must be implemented and deployed inside a Kubernetes cluster. Notice that you may need to implement additional services to fulfill all requirements.
2- **Availability**. Use Kubernetes abstractions to ensure that the services you need to implement are always available.
3- **Elasticity**. Ensure that the system can horizontally scale due to high traffic during market opening hours and scale down after the markets closes.
4- **Fault tolerance**. At any point, any of the services can crash. However, you must ensure that orders handled by such a service are handled correctly by other peer instances of such a service or a new fresh instance.

Implementation notes

- You will be provided with a JavaScript application that you should use together with a provided dataset to place orders to your system.
- You will be provided with the implementation of the Matching Engine. You can extend this component by reusing the existing abstraction, but you cannot modify its original code.

Practical Information

- This assignment is individual, and you cannot collaborate with other students.
- You are required to use JavaScript/TypeScript for implementing your assignment.
- You should deliver a zip file containing the application's code and all deployment archives.
- Together with the source code of the project, you must submit a project report describing your solution to the assignment. The document should also discuss the choices you made for implementing the assignment's requirements. Moreover, the document must describe the steps required to deploy the application. The project report should not exceed 5 pages.
- The student should prepare a presentation (e.g., PowerPoint) to discuss his/her solution to the assignment during the oral exam. In addition to the presentation, the student could provide a video showcasing the system's behavior, availability and elasticity.
- Plagiarism will not be tolerated.
- You must submit all what was asked above by **January 5th, 2025, at 23h59**.
- Questions about the assignment can be sent to Angel Luis.

Appendix 1

Order Book: The term order book refers to an electronic list of buys (bids) and sells (asks) orders for a specific financial symbol (e.g., stock) organized by price level. As shown in Figure 1, an order book lists the *cumulative number of shares* being bid on or asked at each price point. Each bar in the figure represents the cumulative number of orders at that price point. Prices are sorted such that the lowest ask and highest bids are in the center of the chart.
The bars in the chart of Figure 1 show the cumulative number of shares:

1. **Cumulative bid orders in green bars**:
    - These are the total quantities of shares buyers are willing to buy at or below a given price point.
2. **Cumulative ask orders in red bars**:
    - These represent the total quantities of shares sellers are willing to sell at or above a given price point.

Example of cumulative order calculation:

- If there are 100 shares offered for purchase (bids) at EUR 50 and 150 shares at EUR 52, the **cumulative buy order** (the chart bar) at EUR 50 would be:
    - **100** (shares at EUR 50) **+ 150** (shares at EUR 52) = **250** shares in the bar.

Matching Engine: Given an order book and a new order, for example, to buy(bid) some shares of some particular stock, the matching engine looks into the book for ask orders with an asking price lower or equal to the buy order. If such ask orders are found, a match has occurred, and the matching engine executes the orders by emitting two sets of order executions (order fills), one for the asks and one for the bids, indicating the purchase and selling of some of those shares. The matching engine will keep in the order book the part of an order that cannot be matched due to insufficient asks or bids in the book. For example, an order may want to buy (bid) 25 AAPL at EUR 347. However, if there are only 20 asked AAPL shares at that price point, the matching engine will execute the order partially and will only execute the orders in the book up until 20 shares. The remaining 5 shares of the bid will be added to the order book at the given price point.

More information about the order book and the matching engine can be found in the links below:

- https://en.wikipedia.org/wiki/Order_book
- https://en.wikipedia.org/wiki/Order_matching_system