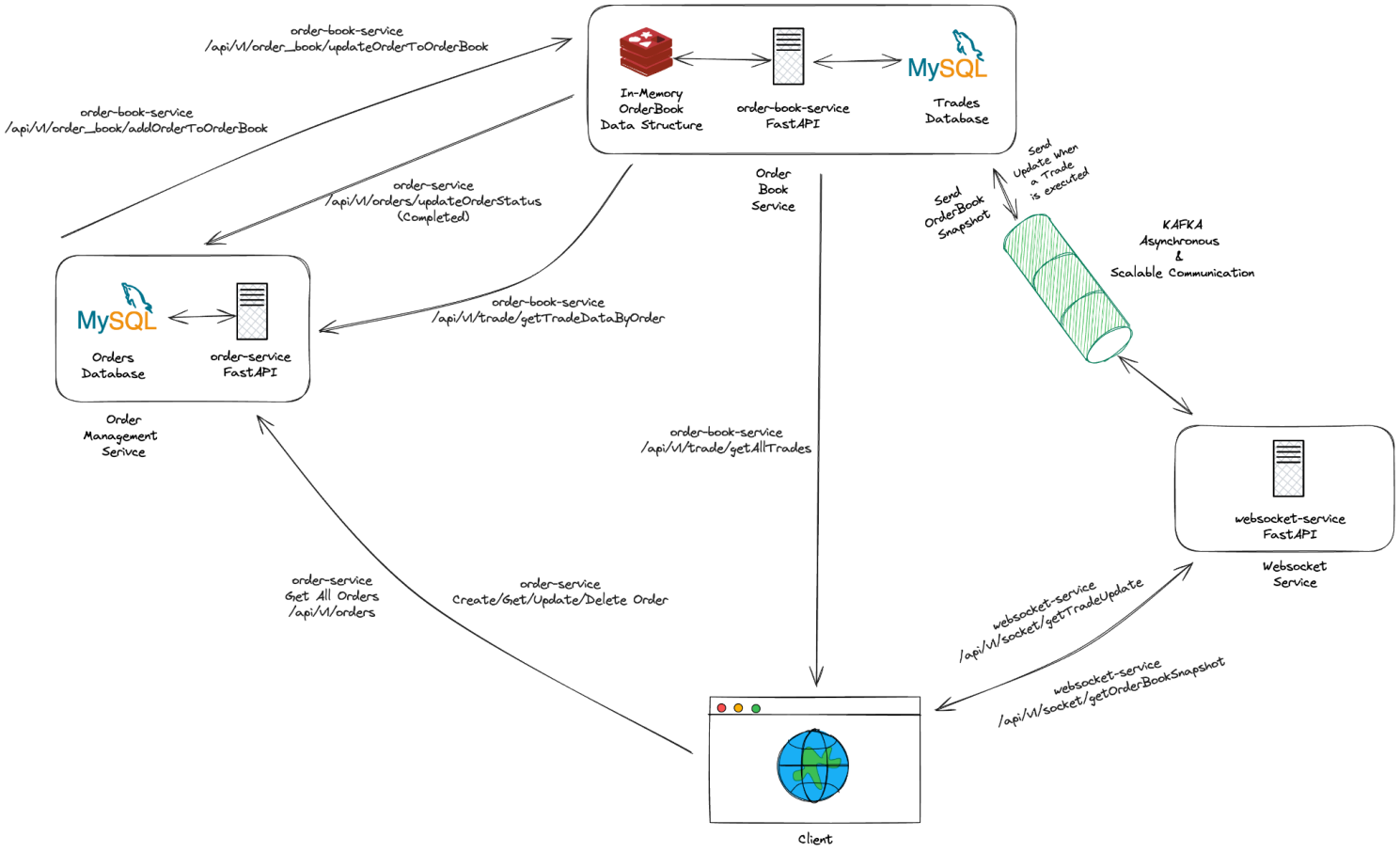
OrderAPI Documentation

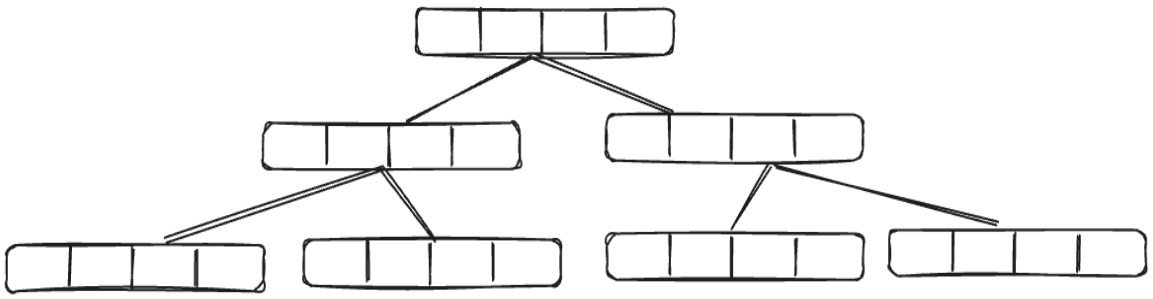
- **Architecture Diagram -**



- **Microservices Details -** There are 3 microservices designed to handle Orders, Order Book/Trades and Web Sockets independently -
    a. **Order Service (order-service) -** This service is primarily responsible for handling client requests related to Orders. The service provides endpoints with POST/PUT/GET/DELETE methods to Create/Update/Read/Delete orders. The order service internally communicates with the Order Book Service, to add/update orders in the Order Book Data Structure, and fetch trade related details for orders.
    b. Order Book Service (order-book-service) - This service is primarily responsible for maintaining the Order Book Data Structure (in memory), providing methods for performing operations like Adding/Removing/Updating orders efficiently. This service also executes all the feasible trades within the order book on addition of a new order, sends update to Order Service and Web Socket service using Kafka (Async) on execution of an order, Order Book Snapshot for Price-Volume related data.
    c. Web Socket Service (web-socket-service) - This service is primarily responsible for providing websocket endpoints to the client, for getting updates on the new trades placed, and Price-Volume related data (Order Book Snapshot). The service consumes data from Kafka topics (Trade Execution and Order Book Snapshot). The Order Book

Service is responsible for pushing data to the Kafka Topics, in case of trade execution. This helps with an event driven architecture, making the services highly scalable and fault tolerant.

- **Order Book Data Structure -** The order book data structure is maintained as a part of the Order Book Service. It facilitates the matching of buy and sell orders based on their prices and quantities.
    a. **Attributes:**
        - **lowestSellingPriceHeap**: A min-heap storing the lowest selling prices.
        - **highestBuyingPriceHeap**: A max-heap storing the highest buying prices.
        - **volumeMap**: A dictionary mapping prices and order sides to corresponding volumes.
        - **queueMap**: A dictionary mapping prices and order sides to queues containing orders at that price.
    b. **Methods:**
        - **__init__**: Initializes the OrderBook object and its attributes.
        - **executeOrder**: Executes a given order by matching it with existing orders in the order book.
        - **addOrderToOrderBook**: Adds a new order to the order book.
        - **removeOrderFromOrderBook**: Removes an order from the order book after it has been executed.
        - **getOrderBookSnapshot**: Retrieves a snapshot of the current order book.
        - **__createTradeObject**: Creates a trade object based on two matching orders.
    c. **Order Book Data Structure - Heap of Queues:**



The **lowestSellingPriceHeap** and **highestBuyingPriceHeap** are integral components of the OrderBook class, serving as data structures to efficiently manage buy and sell orders based on their prices. Each node in these heaps contains a queue of orders sorted by their respective prices.

Structure:

- **lowestSellingPriceHeap**: Utilizes a min-heap where each node represents a queue of orders, with queues sorted in ascending order of order prices.

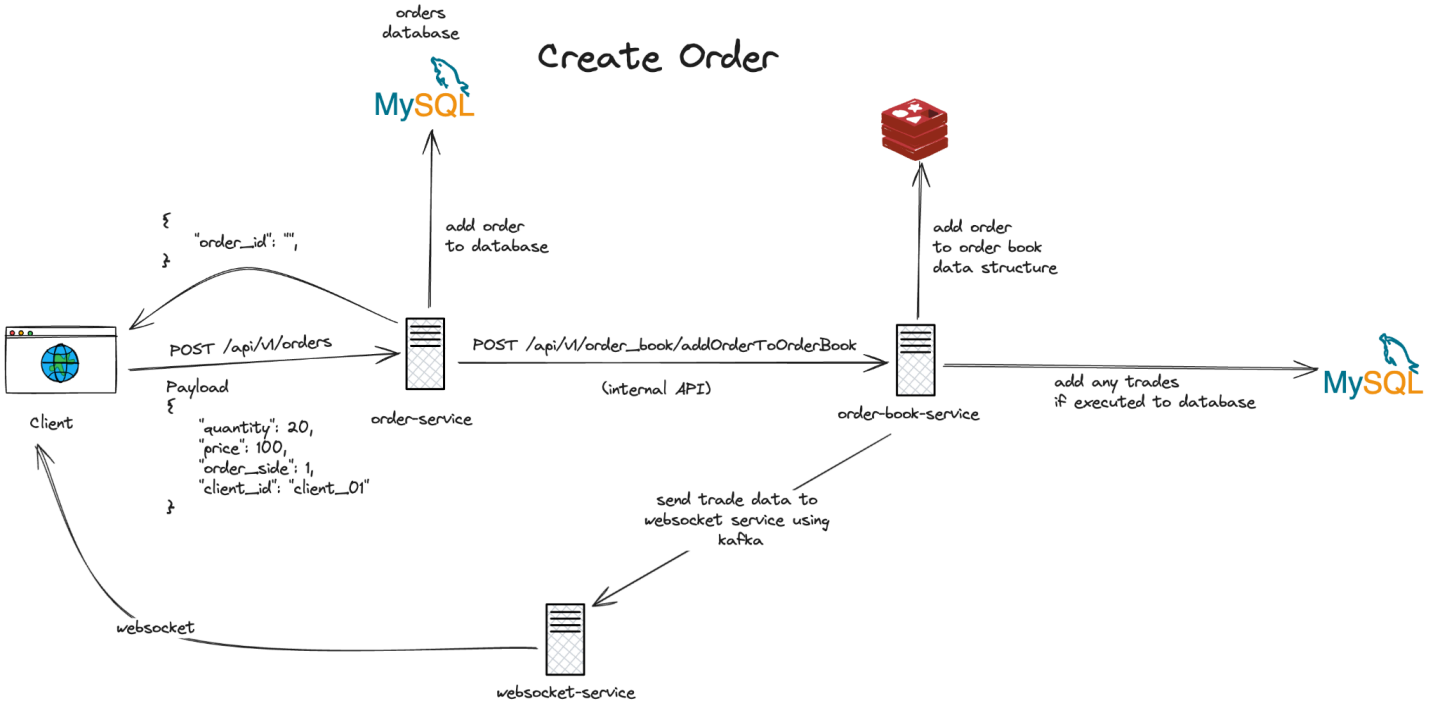
- **highestBuyingPriceHeap**: Implemented as a max-heap storing buy orders, where each node represents a queue of orders, with queues sorted in descending order of order prices.

Efficiency:

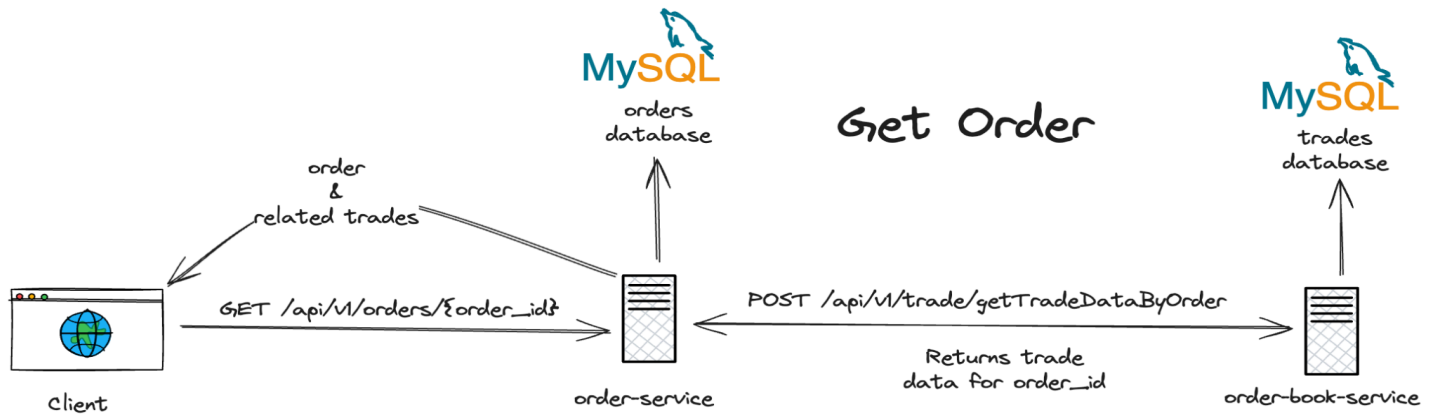
- **Matching Efficiency**: Orders are quickly matched by considering only compatible prices, speeding up the process.
- **Time Complexity**: Operations like insertion and removal have logarithmic time complexity (O(log n)), optimizing search and access.
- **Space Efficiency**: Queues within heap nodes reduce memory usage by grouping orders with the same price.

- **Order Service -** The Order Service API provides endpoints for managing orders within the trading system. These endpoints enable the creation, retrieval, updating, and deletion of orders, along with functionalities to fetch all orders and update their statuses.

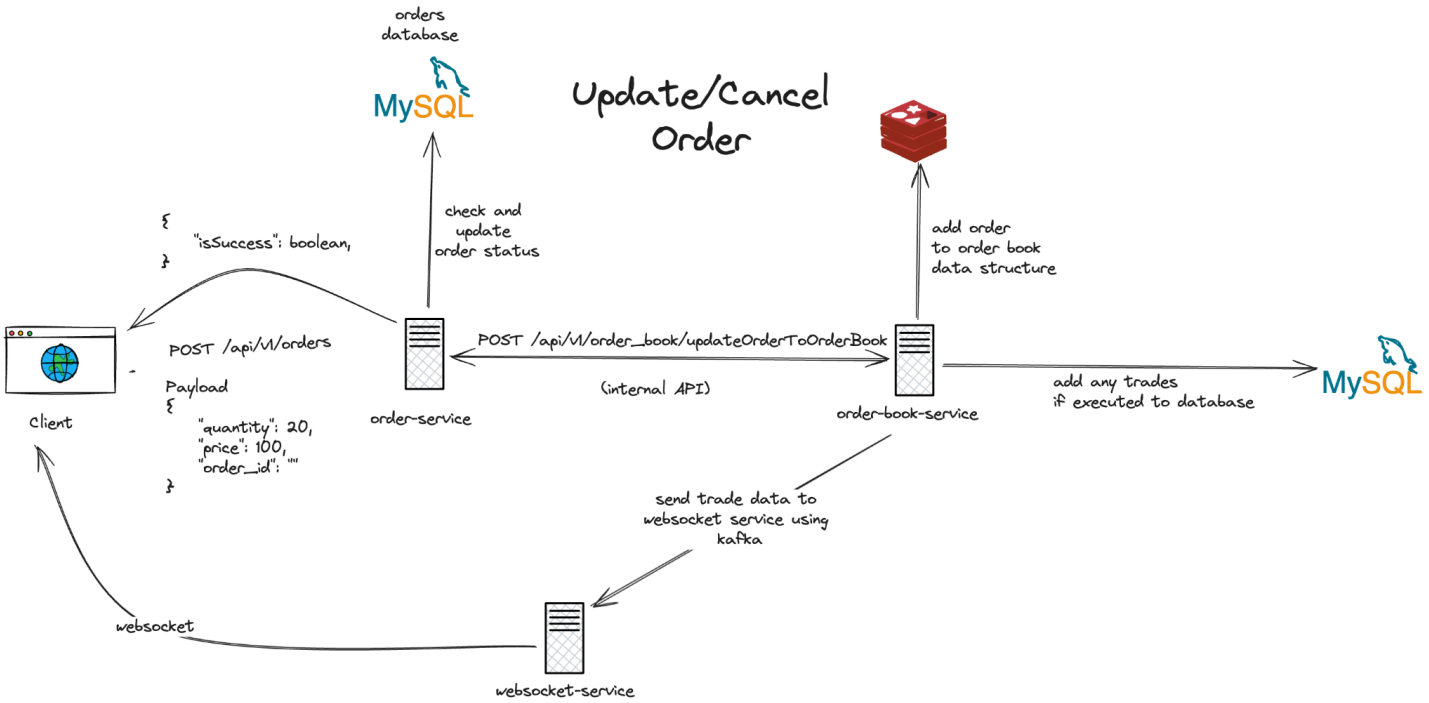
Create Order -



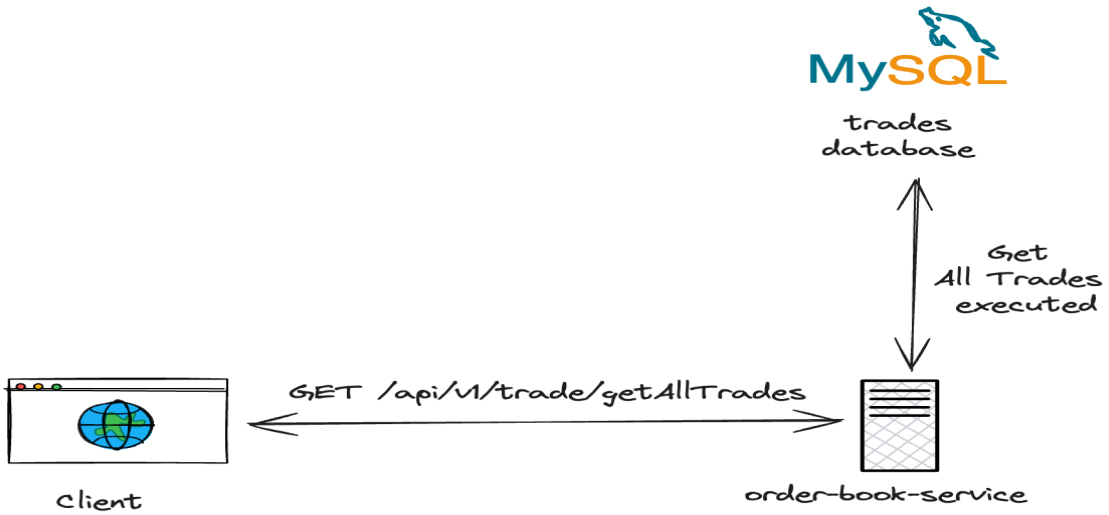
Get Order Details by Order Id -



Update/Cancel Order by Order Id -



Get All Trades -



Web Socket API's