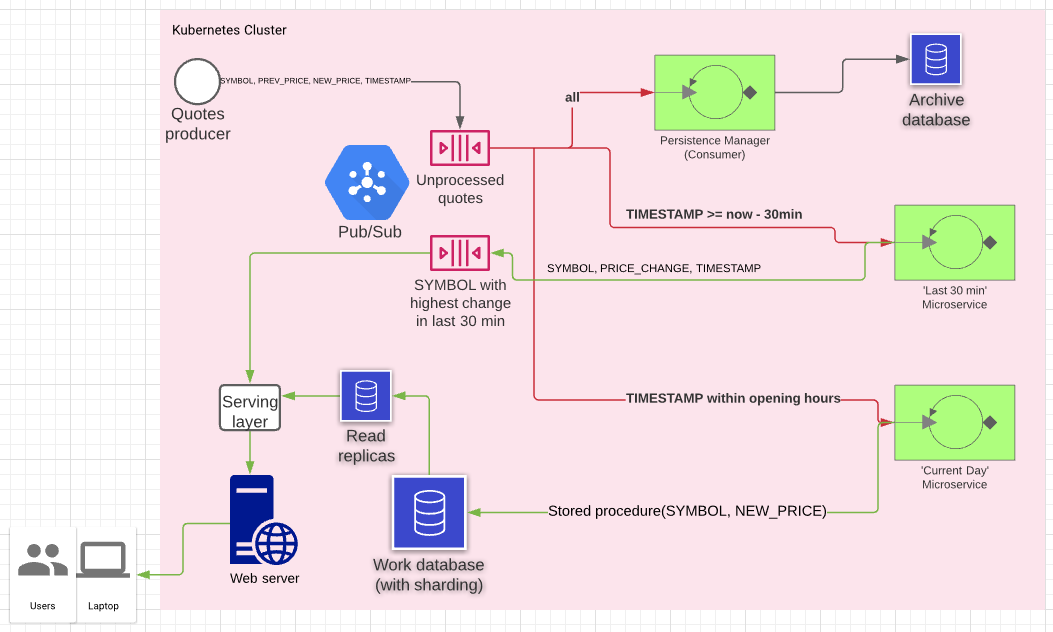
# Requirements

|  |  |  |
| --- | --- | --- |
| **Non-Functional** |  | **Functional** |
| **~ 10,000 quotes**/sec =  **~ 600,000 quotes**/min \*  ~ 25 B / event =  ~ **15 MB/min** |  | Save all quotes to storage. |
|  | Save each minute the stock symbol with highest value change in last 30 minutes. |
|  | Save daily change and min/max prices of each stock. |
|  |  | Service to generate history graph of stock changes. |

# Architecture diagram

[](https://lucid.app/documents/view/f32a6c64-aa3e-4cd7-8100-eb994e6f1f3c)

# 'Last 30 minutes' microservice

|  |
| --- |
| Map<String, StockLast30MinManager> allSymbols = .. |

|  |
| --- |
| processLastMinutes (30);  ScheduledExecutorService executor = Executors.newSingleThreadScheduledExecutor();  Runnable periodicTask = new Runnable () {  public void run () {  processLastMinutes (1);  }  };  executor.scheduleAtFixedRate(periodicTask, 0, 1, TimeUnit.MINUTES); |

|  |
| --- |
| processLastMinutes (ushort **minutes**) {  consume {SYMBOL, NEW\_PRICE} while current time – TIMESTAMP <= **minutes** {  allSymbols[SYMBOL].save(NEW\_PRICE);  }  String highestChangeSymbol = '';  float highestChangeValue = 0;  allSymbols.forEach((symbol, manager) =>  float currentHighestChangeValue = manager.highestChange();  if (abs(currentHighestChangeValue) > abs(highestChangeValue)) {  highestChangeSymbol = symbol;  highestChangeValue = currentHighestChangeValue;  }  } |

## Class StockLast30MinManager



|  |
| --- |
| * Each node in the linked list on the right contains TIMESTAMP and a pointer to a node in an AVL tree on the left. * Each node in the AVL tree contains a price.   void save(**newPrice**){  Insert newPrice to the tree, and add a node to the H(ead) of the list with the current time and a pointer to the new node in the tree.  Scan the list from the T(ail) and for each node that is older than 30 minutes delete the node in both the tree and the list, and balance the tree (O(logN)).  }  float highestChange(){  return the max – min values of the tree (O(logN)).  } |

# 'Current Day' microservice

|  |
| --- |
| **Work database:** (with horizontal sharding by SYMBOL)    **Day start**: (~1 min before opening hour), for all symbols:  ***day\_start\_price*** 🡨 ***day\_end\_price***  **Store procedure**: (parameters: **SYMBOL**, **NEW**\_**PRICE**)  ***day\_end\_price*** 🡨 **NEW\_PRICE**  ***day\_min\_price*** 🡨 Min (***day\_min\_price,*** **NEW\_PRICE**)  ***day\_max\_price*** 🡨 Max (***day\_max\_price,*** **NEW\_PRICE**)  ***timestamp*** 🡨 current time  ***price*** 🡨 NEW\_PRICE |

# Tech stack

For an on-prem solution I would go with Kafka for the pub/sub and any database that supports sharding and read replicas.

For a cloud solution I would choose any comparable managed service, depending on the cloud vendor (further research will be required).