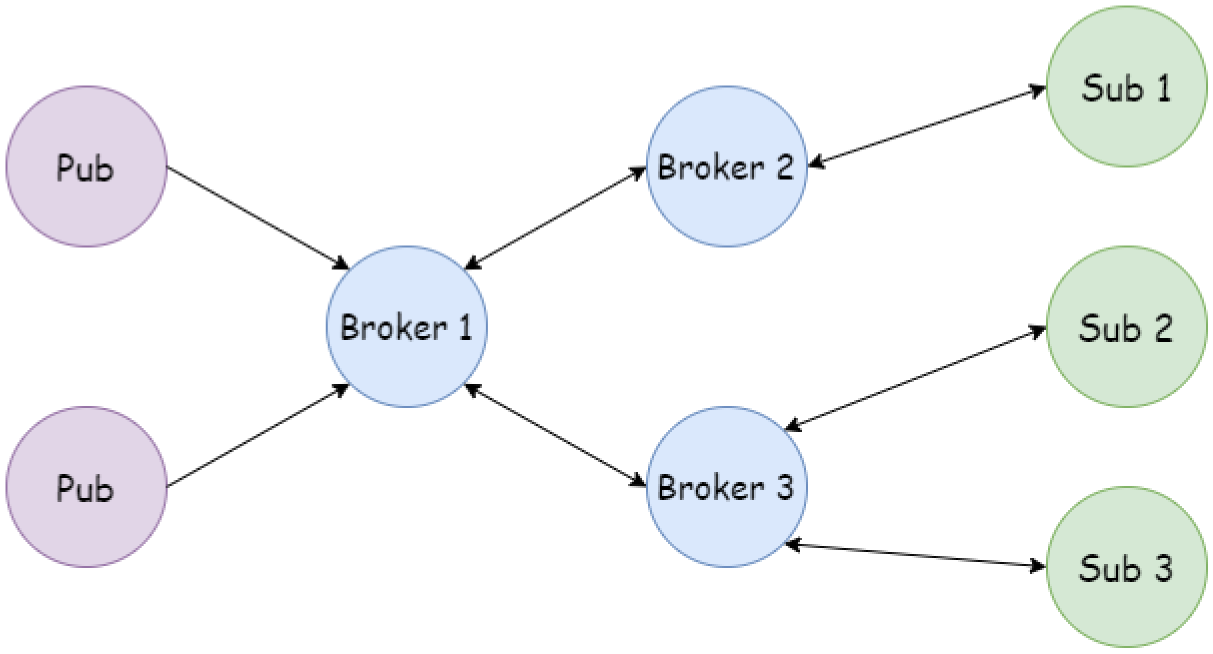
# Event Bases System
# Homework Evaluation

## Architecture & flow

For message passing, we use RabbitMq in order to take advantage of its ability to handle message spikes without overwhelming consumers (back pressure), offering horizontal scalability and traffic data in the form of different metrics and logging observable in the UI interface. Each broker or subscriber has a queue that accepts messages from other sources. At the beginning of the script, subscribers are split randomly among Broker 2 and Broker 3. The 3 existing subscribers generate 3333, 3333, and, respectively, 3334 messages. All the components from the topology (subscribers, publishers, and brokers) function in parallel with a synchronization mechanism only during the setup phase, so publishers wait for the subscribers to finish pushing filters before they start to publish for 3 minutes.



## Messages format:

Messages are converted to bytes using Protobuf models. Each message includes its type (publication or subscription), source, timestamp (for computing latency), and the payload itself. Deserialization is done at the level of each broker, so an equivalent Java class is obtained for hashing and testing the equality during the building of the routing table and matching of different pubs with different subs.

## Broker routing algorithm:

We use identity routing by saving a routing table in the form of a Map<String, HashMap> structure, where keys represent the names of the subs or brokers of interest, a subset of NB U LB, and values represent the unique filters that should be matched by a message in order to be sent to the destination (key). We also use an additional structure, filterToSources, a Map<Subscription, and LinkedHashSet<String>>. The very first time a subscription comes (the sub is not found in filterToSources), it will be sent to all the brokers in NB and will be added to filterToSources. If the subscription comes again from another source, it will be sent to the first source (if the first source is a broker). The third time a subscription comes, it will be added to the routing table if the source does not contain that filter, and the message will not be forwarded to other brokers in the neighborhood.

## Synchronization using a custom orchestrator - Rabbitstrator🐰:

Rabbitstrator is a thread that keeps a ConcurrentHashMap<String, List<Integer>> where keys uniquely represent the broker to be monitored, and the value is the last 16 timestamps recorded from that specific broker. A timestamp is updated at the broker level the very moment it receives a new filter. If there was no change in timestamp in the last 16 checks, the broker ended his routing job. Rabbitstrator also includes support for the recovery of the brokers. A similar approach is taken, considering the latest 16 values of timestamps from the broker; if there is no change in them, it means that the broker crashed.

Different stats are also gathered in a synchronized manner, using ConcurrentHashMap in each broker. Another thread is gathering data, AccuracyWatcher, which monitors the count of the received publications by the brokers and by the subs. If there was no change in the value of the counts for all the brokers and pubs in the last 4 seconds, data transmission is considered to be done. Accuracy is computed using the number of publications received by each thread divided by the number of publications received by Broker 1. Latency is computed using the metadata from each message, then a pooling operation is done.

## Results

These results were obtained by sending publications constantly for 3 minutes and using 3334 subscriptions for each subscriber with every attribute percent at 100%, other than the sign equal, which is set to 25% and 100%.

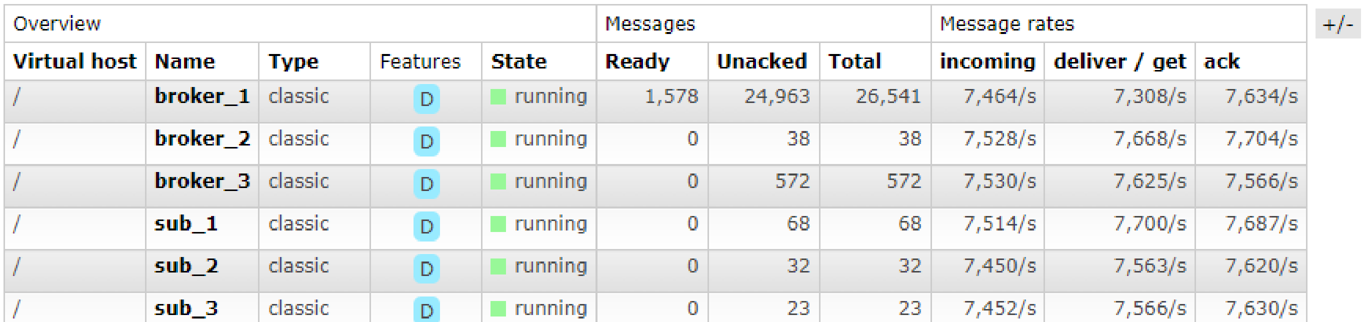
### Node Statistics

| Number of Equals | Node | Latency (ms) | Received | Matching Rate |
|---|---|---|---|---|
| 100% Equal Sign | Sub 1 | 210.896,12 | 2.160.869 | 89.85% |
| | Sub 2 | 210.900,27 | 2.133.949 | 88.73% |
| | Sub 3 | 210.916,34 | 2.133.846 | 88.72% |
| 25% Equal Sign | Sub 1 | 5.719,00 | 2.334.005 | 100% |
| | Sub 2 | 5.859,27 | 2.334.005 | 100% |
| | Sub 3 | 5.858,74 | 2.334.005 | 100% |

| Number of Equals | Node | Received |
|---|---|---|
| 100% Equal Sign | Broker 1 | 2.405.013 |
| | Broker 2 | 2.160.869 |
| | Broker 3 | 2.369.361 |
| 25% Equal Sign | Broker 1 | 2.334.005 |
| | Broker 2 | 2.334.005 |
| | Broker 3 | 2.334.005 |

### RabbitMQ Statistics

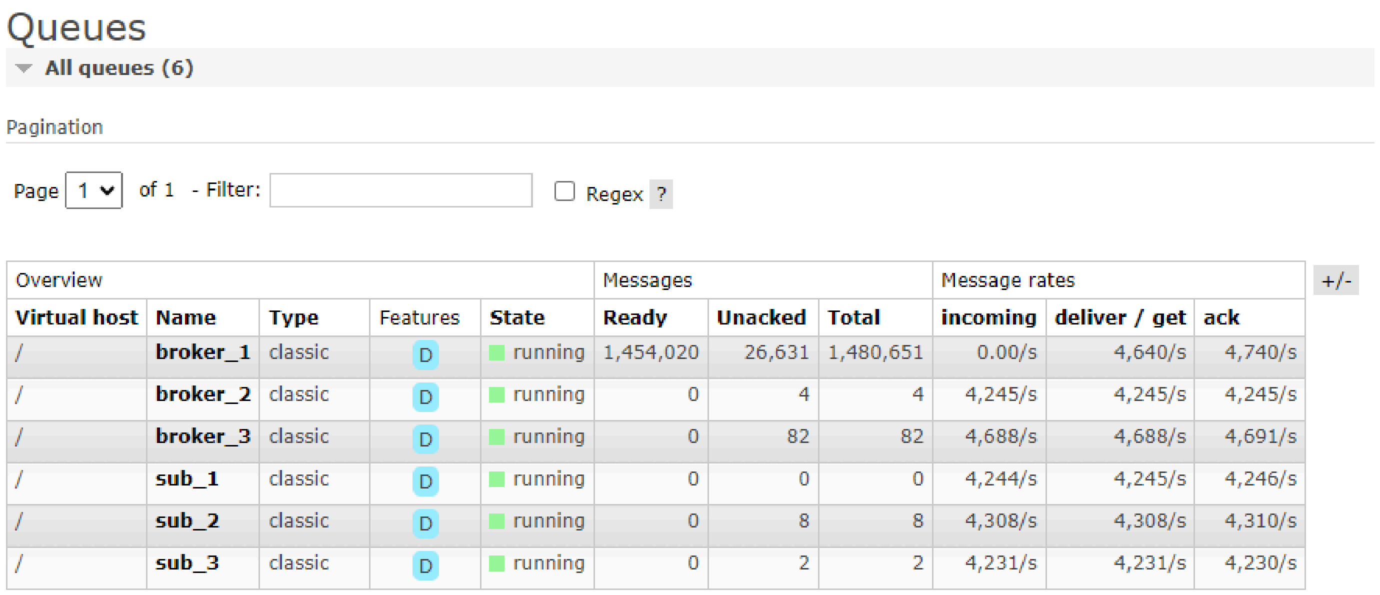
- For 25% number of the equal sign



| Overview | | | | | Messages | | | Message rates | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Virtual host** | **Name** | **Type** | Features | **State** | **Ready** | **Unacked** | **Total** | **incoming** | **deliver / get** | **ack** |
| / | **broker_1** | classic | D | 🟩 running | 1,578 | 24,963 | 26,541 | 7,464/s | 7,308/s | 7,634/s |
| / | **broker_2** | classic | D | 🟩 running | 0 | 38 | 38 | 7,528/s | 7,668/s | 7,704/s |
| / | **broker_3** | classic | D | 🟩 running | 0 | 572 | 572 | 7,530/s | 7,625/s | 7,566/s |
| / | **sub_1** | classic | D | 🟩 running | 0 | 68 | 68 | 7,514/s | 7,700/s | 7,687/s |
| / | **sub_2** | classic | D | 🟩 running | 0 | 32 | 32 | 7,450/s | 7,563/s | 7,620/s |
| / | **sub_3** | classic | D | 🟩 running | 0 | 23 | 23 | 7,452/s | 7,566/s | 7,630/s |

+/-

- For 100% number of the equal sign



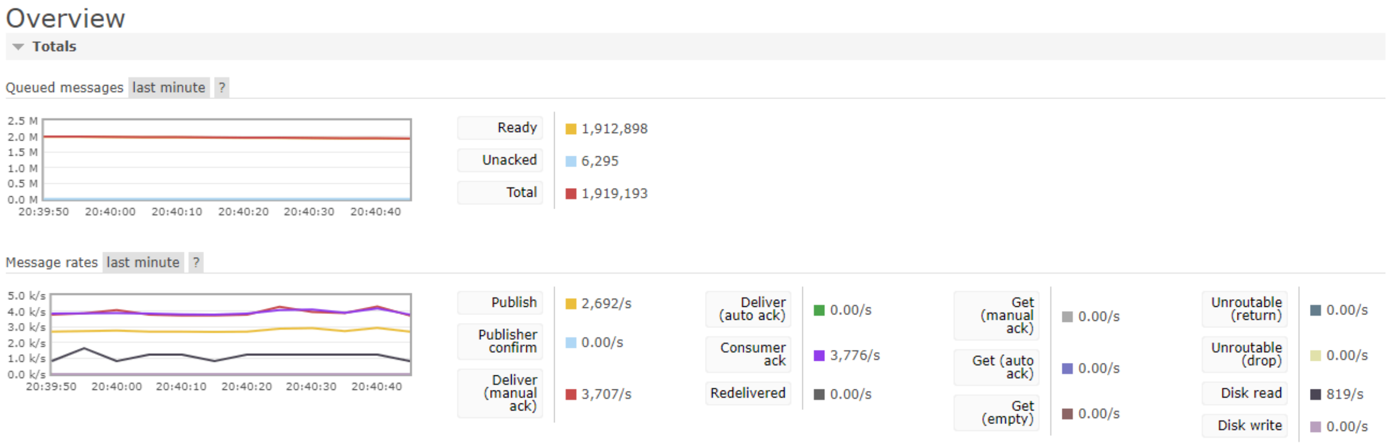
Queues

All queues (6)

Pagination

Page 1 of 1 - Filter: Regex ?

| Overview | | | | | Messages | | | Message rates | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Virtual host | Name | Type | Features | State | Ready | Unacked | Total | incoming | deliver / get | ack |
| / | broker_1 | classic | D | 🟩 running | 1,454,020 | 26,631 | 1,480,651 | 0.00/s | 4,640/s | 4,740/s |
| / | broker_2 | classic | D | 🟩 running | 0 | 4 | 4 | 4,245/s | 4,245/s | 4,245/s |
| / | broker_3 | classic | D | 🟩 running | 0 | 82 | 82 | 4,688/s | 4,688/s | 4,691/s |
| / | sub_1 | classic | D | 🟩 running | 0 | 0 | 0 | 4,244/s | 4,245/s | 4,246/s |
| / | sub_2 | classic | D | 🟩 running | 0 | 8 | 8 | 4,308/s | 4,308/s | 4,310/s |
| / | sub_3 | classic | D | 🟩 running | 0 | 2 | 2 | 4,231/s | 4,231/s | 4,230/s |



Overview

Totals

Queued messages last minute ?

| | |
|---|---|
| Ready | 1,912,898 |
| Unacked | 6,295 |
| Total | 1,919,193 |

Message rates last minute ?

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Publish | 2,692/s | Deliver (auto ack) | 0.00/s | Get (manual ack) | 0.00/s | Unroutable (return) | 0.00/s |
| Publisher confirm | 0.00/s | Consumer ack | 3,776/s | Get (auto ack) | 0.00/s | Unroutable (drop) | 0.00/s |
| Deliver (manual ack) | 3,707/s | Redelivered | 0.00/s | Get (empty) | 0.00/s | Disk read | 819/s |
| | | | | | | Disk write | 0.00/s |