# **Assignment 3**

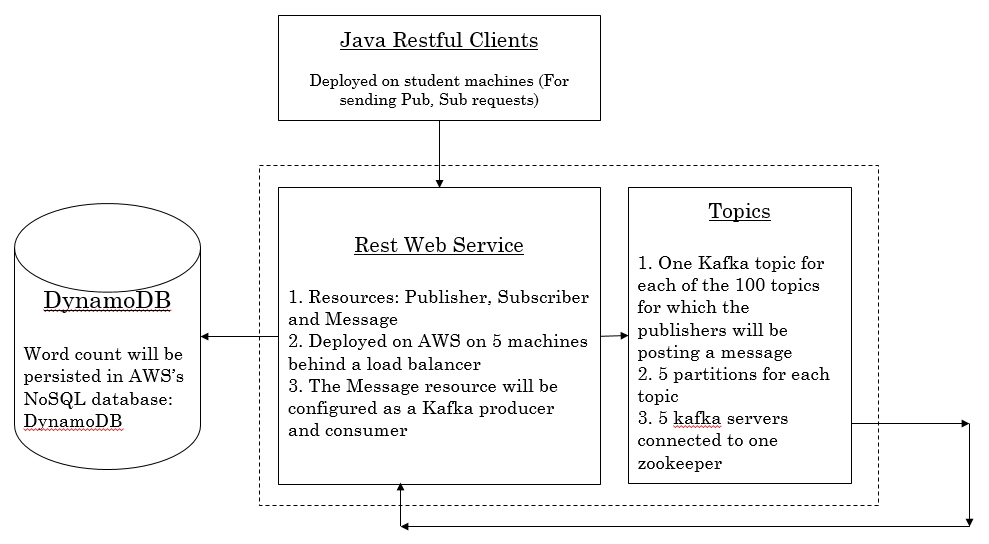
# **Building Scalable Distributed System**

Submitted by-

Nirav Aga

Praneeta Jhanwar

Architecture Design:



We plan to use Kafka for implementing the server. The deciding criteria is as follows:

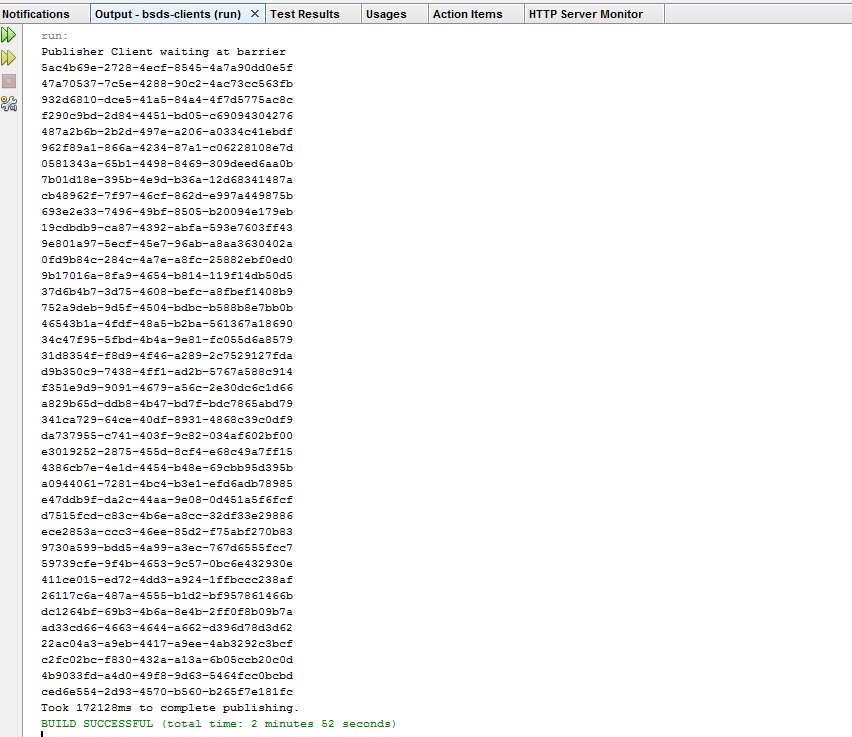
1. Since Kafka is build up on Zookeeper, there will be very less efforts required for synchronization.
2. Kafka is ideal for applications where there are more reads than writes (10:1) and since in our application we have many subscribers reading the same message, Kafka serves as a good fit.

Additional Notes:

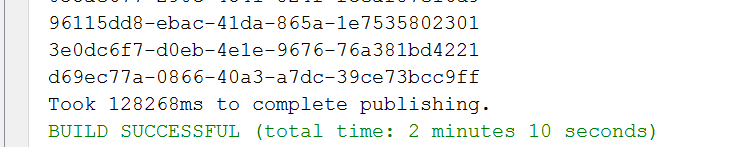
1. Kafka API will provides consumers to read from an offset, and thus we do not need to persist subscriber states in the database.

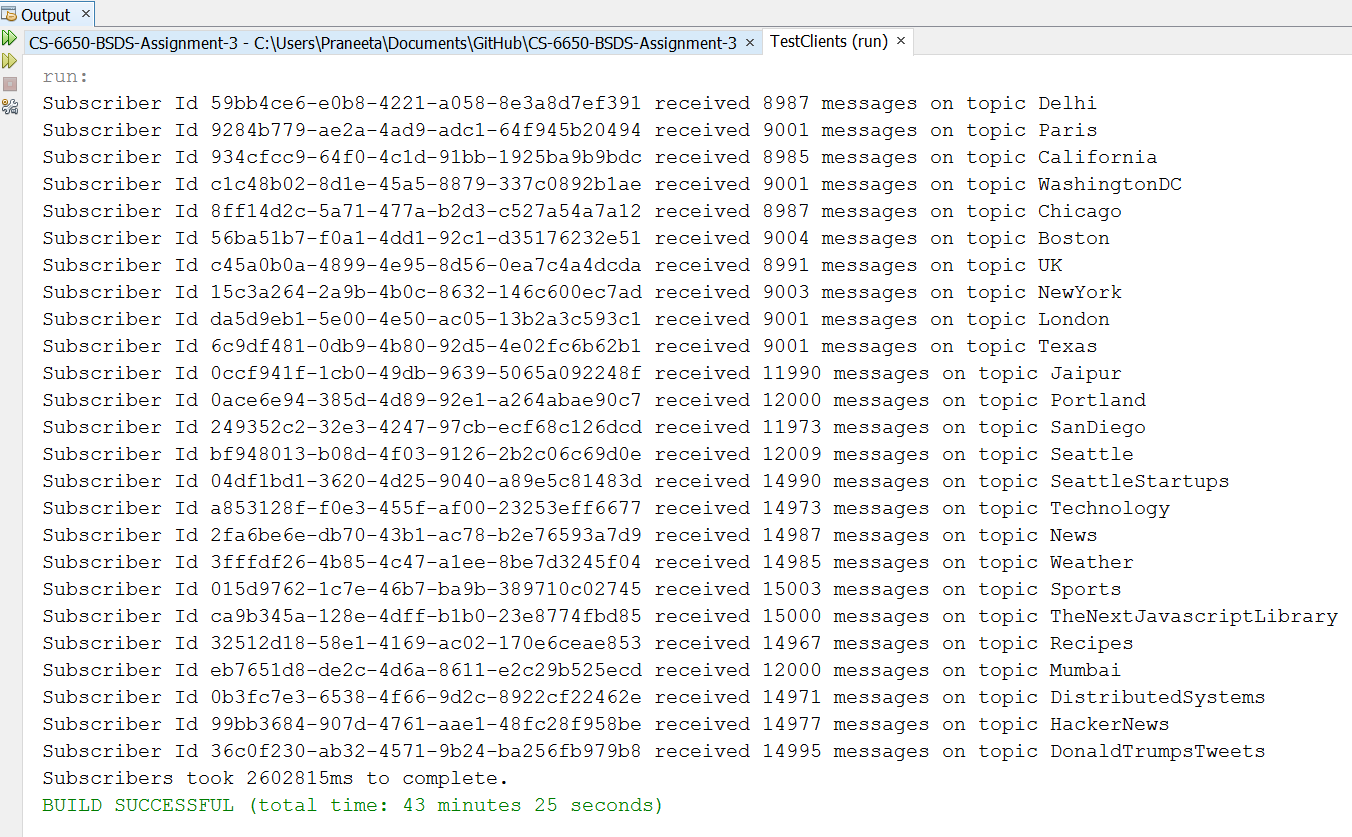
Test Case:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Number of threads | Number of Messages | Runtime | Throughput  (messages/second) |
| Publishers on Machine 1 | 40 | 3000 | 2 min, 52 sec | 697.67 |
| Publishers on Machine 2 | 60 | 3000 | 2 min, 10 sec | 1384.62 |
| Subscribers on Machine 2 | 25 | 300,000 | 43 min, 25 sec | 115.16 |





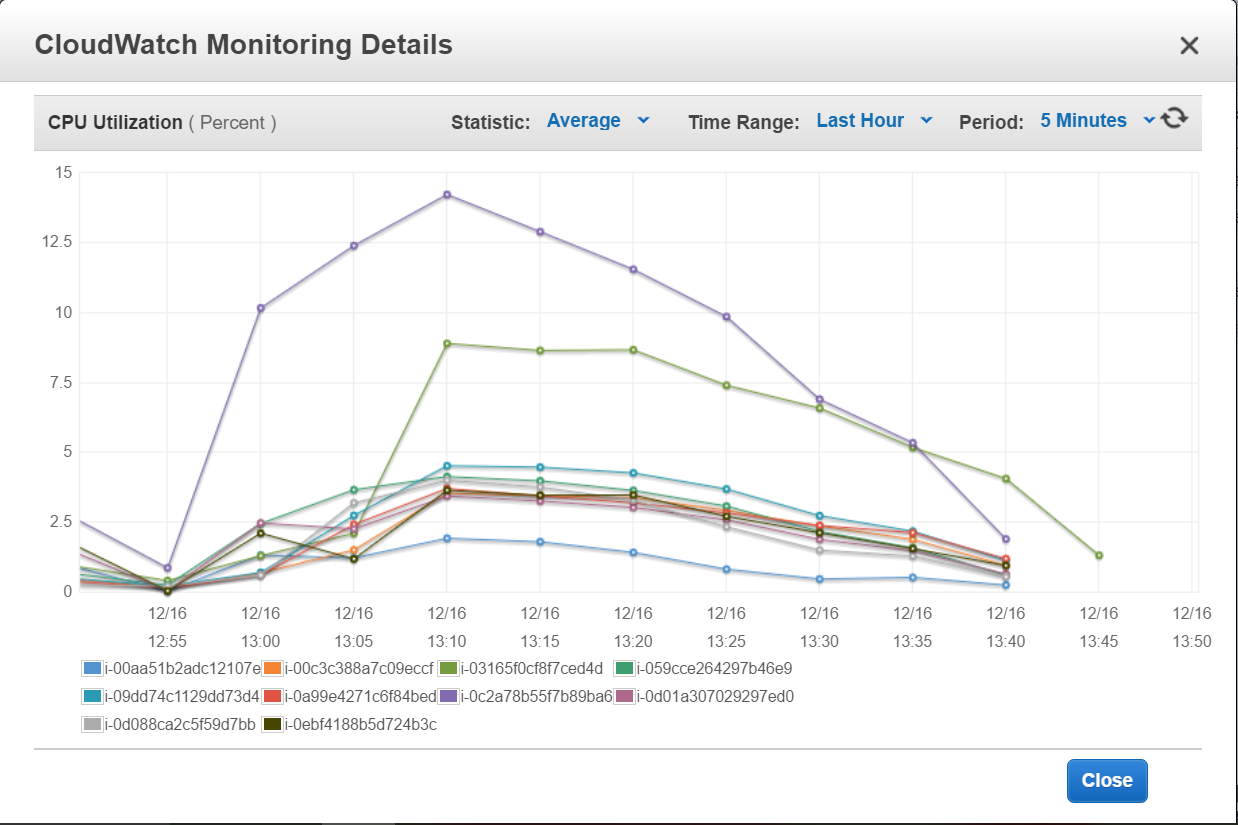




We did not distribute the messages evenly, number of messages published per topic:

|  |  |
| --- | --- |
| Sports | 15000 |
| News | 15000 |
| Technology | 15000 |
| DistributedSystems | 15000 |
| TheNextJavascriptLibrary | 15000 |
| DonaldTrumpsTweets | 15000 |
| SeattleStartups | 15000 |
| HackerNews | 15000 |
| Weather | 15000 |
| Recipes | 15000 |
| Seattle | 12000 |
| Portland | 12000 |
| SanDiego | 12000 |
| Mumbai | 12000 |
| Jaipur | 12000 |
| Chicago | 9000 |
| Boston | 9000 |
| Delhi | 9000 |
| NewYork | 9000 |
| WashingtonDC | 9000 |
| Paris | 9000 |
| London | 9000 |
| UK | 9000 |
| California | 9000 |
| Texas | 9000 |

Following is a picture of the CPU utilization of the 10 machines we used for running our web tier and Kafka servers:



Observations:

1. We observed that only the number of threads was truly reflecting the degree of parallelism. Increasing number of messages just increased the wall times as each subscriber would read the messages sequentially. Thus for a total number of 5000 messages, if we tried 50 pubs with 100 messages each and the other combination of 10 pubs with 500 messages each, the former required less run time, as it had higher degree of parallelism.
2. As can be seen, the load on the machines never increased more than 15%, which shows that our server was well scaled and the load was also well distributed. Our major limitation was that we were not able to send requests from AWS instances and thus used our own laptops to send requests. The network caused huge delays and thus slowed down the subscriber clients.
3. Another limitation of using our own machines was that we could not increase the total number of publisher threads on our respective machines as we kept getting socket connection errors for more number of threads.
4. As per the requirements, we had as many number of subscribers as the topics. Each subscriber was supposed to receive all the messages published for the respective topic. We created 5 partitions for each topic. The publishers were able to send their messages to different partition, which led to higher degree of parallelism for publishers and thus we saw smaller wall times. But since the subscribers had to receive all the messages for a topic, there was no parallelism for subscribers and thus we saw higher wall times for subscribers.