Design:

The components of this project are -

- The Web Tier - contains the REST web services
- The EJB Tier - which contains
    - the session beans for the publisher, subscriber, message and word count
    - entity beans for the publisher, subscriber, message and word count.
- The Database - Derby using JPA and JQL.
- The Remote Interface - serves as an API for the EJB which is used by clients to interact with the EJB.

- **Web Tier -**
    - **Publisher Service -** Contains a POST method to register the publisher and return the id of the publisher. Consumes the topic as a parameter and returns a long integer which is the id of the publisher.
    - **Subscriber Service -** Contains a POST method to register the subscriber and return the id of the subscriber. Consumes the topic as a parameter and returns a long integer which is the id of the subscriber.
    - **Message Service -** Contains 2 methods - one POST to publishContent with the messageContent and publisherId as a query parameter and returns void. The other GET to getLatestContent with the subscriberId as a Query Parameter which returns the message String.
    - **TopNWords Service -** Contains a GET method that takes N as an input parameter and returns a comma separated string of the top N popular words and their counts.

s

- **EJB Tier -**
    - **Entity Beans -**
        - Publisher Entity **-** id, topic
        - Subscriber Entity - id, topic, lastSeenMessageId
        - Message Entity - id, message, topic
        - WordCount Entity - id, word, count
    - **Session Beans -**
        - Publisher Storage Bean - Creates and Persists the publisher entity on registration
        - Subscriber Storage Bean - Creates and Persists the subscriber entity on registration
        - Message Storage Bean - Creates and Persists the message entity when publish content is called with a message.

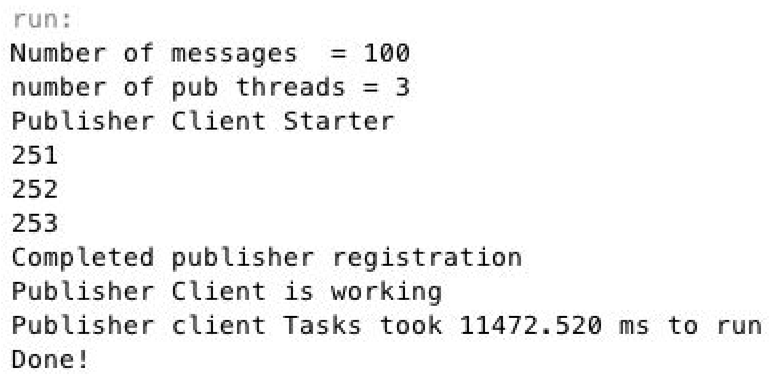
        - WordCount Storage Bean - Creates a new word count entity if word doesn’t exist, else the word count for the word is fetched and incremented.
    - **Message Driven Bean -**
        - Word Count Message Driven Bean - Implements the Message Listener and processes messages in the queue by implementing onMessage(). Invokes a singleton session bean that splits the message into words and updates the word count.

Results

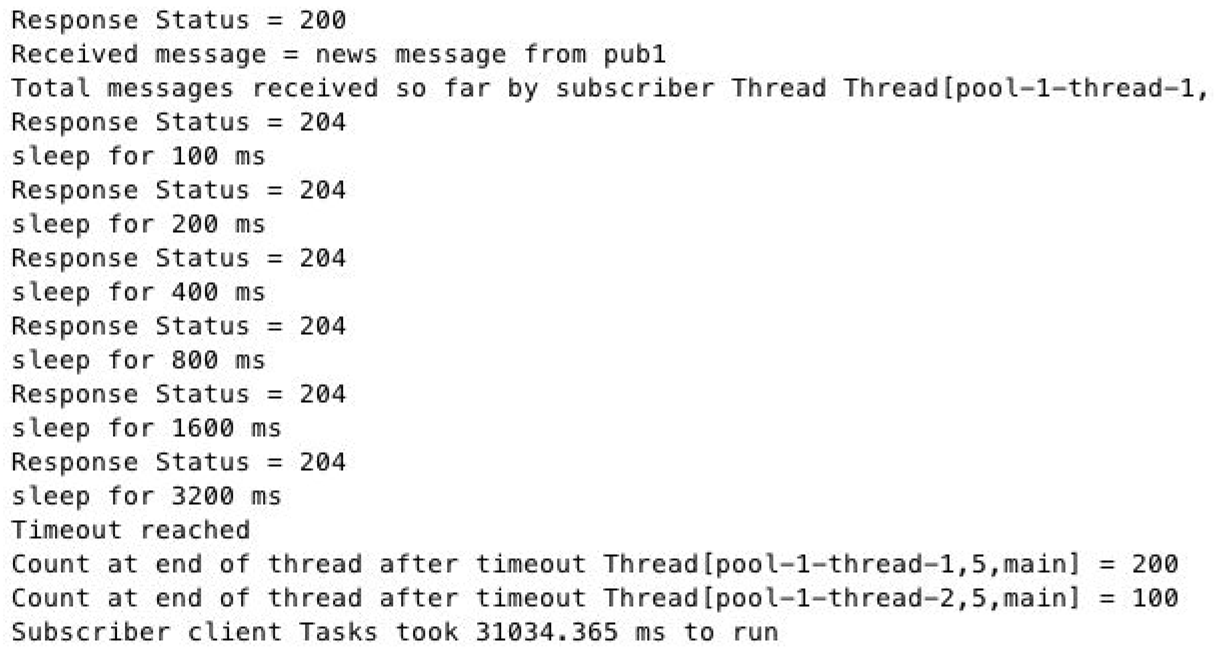
Step1: Basic Functional Correctness:
Implemented the CA Server as HTTP/REST server. On running 3 CPs publishing 100 messages on 2 topics “news” and “sports”, and 2 subscribers subscribing to each “news” and “sports”.
Subscriber 1 received 2000 messages and subscriber 2 received 1000 messages.
Time taken by publisher = 7.03seconds
Time taken by subscriber = 11.4 seconds

Step 2: Add Term Counts:

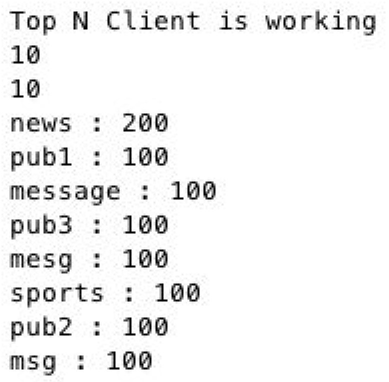
1. After adding term counts and calling top 50 words, the result is as shown -



```
run:
Number of messages  = 100
number of pub threads = 3
Publisher Client Starter
251
252
253
Completed publisher registration
Publisher Client is working
Publisher client Tasks took 11472.520 ms to run
Done!
```

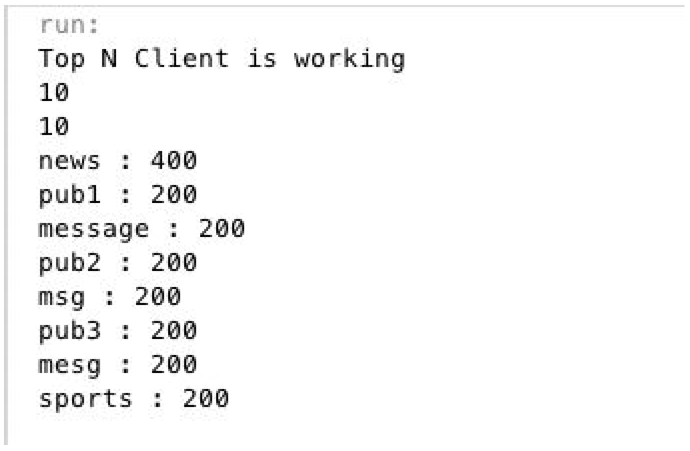


```
Response Status = 200
Received message = news message from pub1
Total messages received so far by subscriber Thread Thread[pool-1-thread-1,
Response Status = 204
sleep for 100 ms
Response Status = 204
sleep for 200 ms
Response Status = 204
sleep for 400 ms
Response Status = 204
sleep for 800 ms
Response Status = 204
sleep for 1600 ms
Response Status = 204
sleep for 3200 ms
Timeout reached
Count at end of thread after timeout Thread[pool-1-thread-1,5,main] = 200
Count at end of thread after timeout Thread[pool-1-thread-2,5,main] = 100
Subscriber client Tasks took 31034.365 ms to run
```



```
Top N Client is working
10
10
news : 200
pub1 : 100
message : 100
pub3 : 100
mesg : 100
sports : 100
pub2 : 100
msg : 100
```

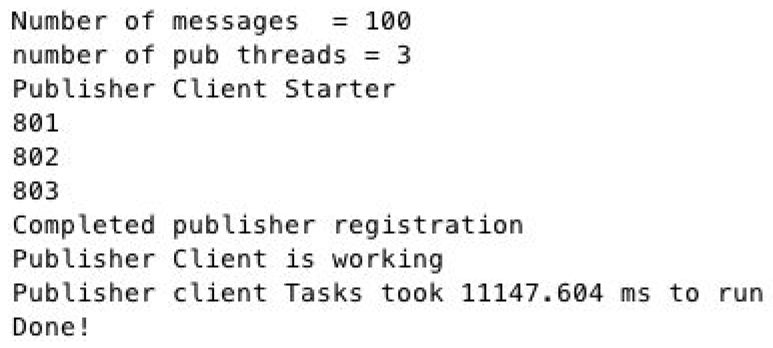
2. After running previous experiment and re-publishing 100 messages on each topic, the word counts have doubled as shown -



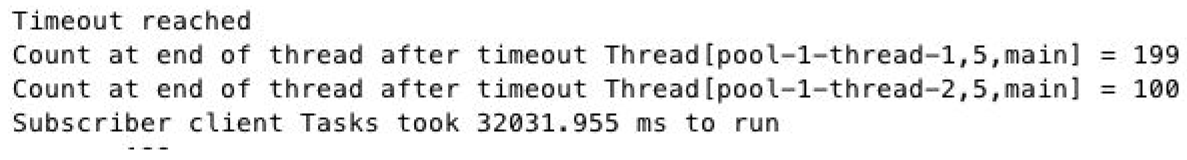
```
run:
Top N Client is working
10
10
news : 400
pub1 : 200
message : 200
pub2 : 200
msg : 200
pub3 : 200
mesg : 200
sports : 200
```

Step 3: Deploying on AWS:
After successfully deploying on AWS and repeating the above experiment,
Time taken by publisher = 11.08 seconds
Time taken by subscriber = 21.2 seconds

Step 4: Stress testing Part 1:
Created a TopN Client that runs 20 threads that repeatedly Calls the Top(N) HTTP request with a random N from 1 to 100 and Sleeps for 1 second.
On running topN along with the publisher and subscriber,
Time taken by publisher = 11.14 seconds
Time taken by subscriber = 26.2 seconds



```
Number of messages  = 100
number of pub threads = 3
Publisher Client Starter
801
802
803
Completed publisher registration
Publisher Client is working
Publisher client Tasks took 11147.604 ms to run
Done!
```



```
Timeout reached
Count at end of thread after timeout Thread[pool-1-thread-1,5,main] = 199
Count at end of thread after timeout Thread[pool-1-thread-2,5,main] = 100
Subscriber client Tasks took 32031.955 ms to run
```

Step 5: Stress testing Part 2:
Tested the CA with 20 CPs and 10 CSs running concurrently each publishing 100 messages and simultaneously running the TopNClient.
Time taken by publisher = 12.1 seconds
Time taken by subscriber = 52.2 seconds

```
run:
Number of messages  = 100
number of pub threads = 20
Publisher Client Starter
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
Completed publisher registration
Publisher Client is working
Publisher client Tasks took 12101.027 ms to run
Done!
```

```
Count at end of thread after timeout Thread[pool-1-thread-3,5,main] = 202
Count at end of thread after timeout Thread[pool-1-thread-7,5,main] = 202
Count at end of thread after timeout Thread[pool-1-thread-2,5,main] = 202
Subscriber client Tasks took 52075.216 ms to run
Count at end of thread after timeout Thread[pool-1-thread-4,5,main] = 202
Count at end of thread after timeout Thread[pool-1-thread-9,5,main] = 202
Count at end of thread after timeout Thread[pool-1-thread-10,5,main] = 202
Count at end of thread after timeout Thread[pool-1-thread-8,5,main] = 202
Count at end of thread after timeout Thread[pool-1-thread-1,5,main] = 202
Count at end of thread after timeout Thread[pool-1-thread-6,5,main] = 202
Count at end of thread after timeout Thread[pool-1-thread-5,5,main] = 202
pub3 : 1818
mesg : 1818
dogs : 202
finance : 202
movies : 202
cats : 202
pets : 202
books : 202
makeup : 202
sports : 202

Done!
```