Facebook-like Service

Niraj Dedhia ; Shweta Yakkali

1. INTRODUCTION

The aim of the project was to develop a Facebook-like social networking service on top of the cloud infrastructure. The social networking Facebook-like service provides the functionalities of Login/Logout, Sign Up/ Register, Send text message to friends and receive the receive and send image or any other files to friends and receive them. The front-end of the application was developed using the Flask Web Framework in Python. For the Database we used MySQL for storing all the information about users, their friends, messages sent and received in a distributed network of virtual containers. The project provides the functionalities of Load-Balancing and Security which is discussed in detail in the successive sections.

2. ALGORITHM DESCRIPTION

The algorithm or the flow of execution of the application on a request to deliver the response is as in figure(1). The execution flow based on the two functionalities implemented is as follows:

At Load Balancer:
On receipt of request from user:
- Perform round robin to select container
- Forward request to selected container

On reply from Container:
- Convert JSON String in to dictionary
- Generate dyanamic web pages by feeding the results
- Send this web pages to user's machine

At Container:
On receipt of request from load balancer:
- calculate hash value for user name
- select respective DB based on hash value
- Perform operations and fetch the result
- Convert results in to JSON string
- Forward it to Load balancer

At User:
User can ask for following services to load balancer:
- Login/Logout
- Register
- Send Image/Messages
- Receive Image/Messages
- View profile

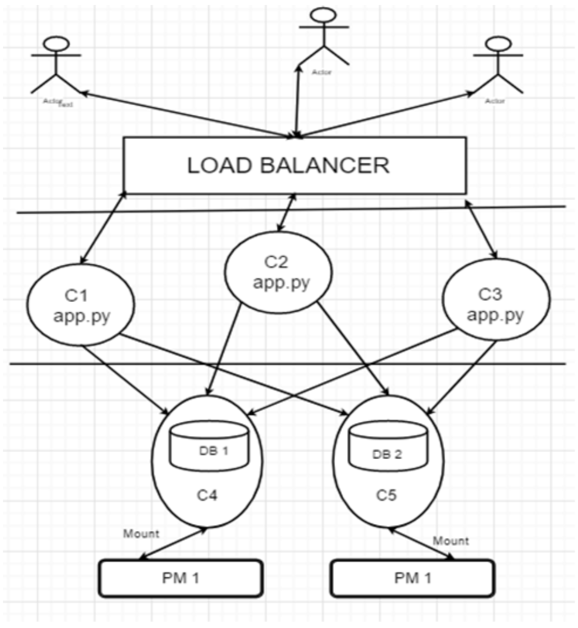


Figure 1: System Architecture

2.1 Load Balancing

In this stage, the incoming requests of login, register, send message or send image/file issued by the users, is handled by the Load Balancer as seen in figure(1). The load balancer handles the incoming requests by implementing the Round Robin technique. A global counter of requests is maintained in the load balancer which takes a modulus of the counter of the current request number issued by user(s) with the number of active containers in the system and transfers the request issued by the user, to the specific container based on output of the modulus operation. The output of the modulus operation is one of the active containers and load balancer now directs the request to this container. This can be seen in Figure(2) and Figure(3) where requests get transferred to different containers at 8001 and 8003 in a round robin fashion.

The respective container on receiving the request from the load balancer processes the request issued by the user. To do this, the container invokes the appropriate method which calls the relevant distributed database. The invoking of the appropriate Database which has the user information depends on the output of a hash on the user name. For example, user with user name *shweta* with be mapped to one of the two containers and say another user with user name *raj* will be mapped to another Database since the hash is done on the user name. This hashing can be seen in Figure(4).The details of the user will be retrieved from the respective Database for processing the relevant request of authentication during login, addition of information during register, view friends and send messages/image to them. Thus, data is not maintained in a centralized manner and is distributed in several databases.

2.2 Security

The next feature implementation was Security inorder to make the system profile secure and not prone to attacks. The functionalities of Authentication during Login, Password Hashing for prevention against Man-In-The-Middle Attack and Secret Key Generation for prevention against Replay attack were implemented.

Authentication:

- To access the service, users must register.

- Users must have username and password to access their profile page.

- Users require correct credentials, to re-login after logging out.

Password Hashing:

- When a user registers, the password provided is hashed at the load balancer and passed to one of the containers for storing the information in Database.

- By doing this, Man-In-The-Middle Attack is prevented.

- In flask, API calls are made instead of socket programming so if any anonymous user tries to access the API directly using the password the attacker will never be able to get an access as we are storing hash value of password.

- To hack the system, anonymous user must be aware of hash function and which would be bit difficult to know since we have implemented our own hash function.

Secrete Key Generation:

- Initially we used session cookie to maintain the session for the user.

- As we proceeded further, we decided to implement token based secure algorithm. For every user who logs in, we create a Random Secret Key.

- This key remains alive until the user is logged in and we use this key to forward the requests back and forth between the load balancer and containers.

- Session variable is also used to keep track of userâĂŹs presence.

3. EXPERIMENTAL CONFIGURATIONS

The Experimental configurations done are mentioned as follows:

1. Initially, we tried to have a loose coupling in the system i.e. have the Database and the application container run in separate machines but this required the use of the command *docker expose*, which was inaccessible for us. To overcome this, we had the database and the application container run on the same machine and connected them to serve the incoming requests by using the *docker –link command* to connect all the databases and the containers.

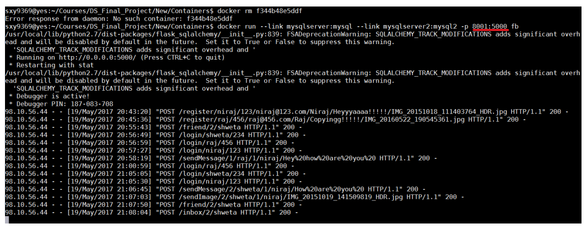
2. The next configuration was to have the developed API return the response obtained from the database and convert it into a dictionary and then convert the dictionary to a JSON object which is parsed by using *Jinja* to render the results of the request as a response to the user.

3. Also, we have used Mount command, to store the database created on the Physical machine to avoid loss of data in case of container failure.

4. The images stored for the respective profile and the images sent and received are stored in the virtual container.

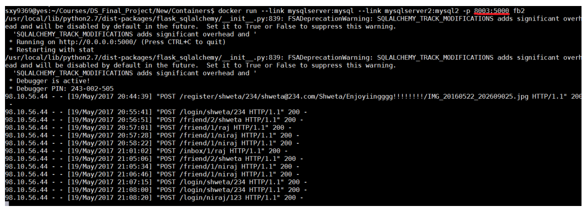
4. RESULTS

The results for the functionalities implemented are as follows:



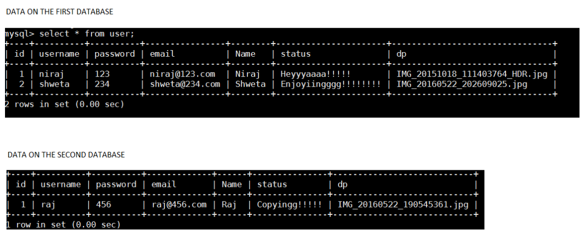
```
sxy9369@yes:~/Courses/DS_Final_Project/New/Containers$ docker rm f344b48e5ddf
Error response from daemon: No such container: f344b48e5ddf
sxy9369@yes:~/Courses/DS_Final_Project/New/Containers$ docker run --link mysqlserver:mysql --link mysqlserver2:mysql2 -p 8001:5000 fb
/usr/local/lib/python2.7/dist-packages/flask_sqlalchemy/__init__.py:839: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overh
ead and will be disabled by default in the future.  Set it to True or False to suppress this warning.
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
/usr/local/lib/python2.7/dist-packages/flask_sqlalchemy/__init__.py:839: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overh
ead and will be disabled by default in the future.  Set it to True or False to suppress this warning.
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
 * Debugger is active!
 * Debugger PIN: 187-083-708
98.10.56.44 - - [19/May/2017 20:43:20] "POST /register/niraj/123/niraj@123.com/Niraj/Heyyyaaaa!!!!!/IMG_20151018_111403764_HDR.jpg HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:45:36] "POST /register/raj/456/raj@456.com/Raj/Copyingg!!!!!/IMG_20160522_190545361.jpg HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:55:43] "POST /friend/2/shweta HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:56:49] "POST /login/shweta/234 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:56:59] "POST /login/raj/456 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:57:27] "POST /login/niraj/123 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:58:19] "POST /sendMessage/1/raj/1/niraj/Hey%20how%20are%20you%20 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:00:59] "POST /login/raj/456 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:05:05] "POST /login/shweta/234 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:05:30] "POST /login/niraj/123 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:06:45] "POST /sendMessage/2/shweta/1/niraj/How%20are%20you%20 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:07:03] "POST /sendImage/2/shweta/1/niraj/IMG_20151019_141509819_HDR.jpg HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:07:50] "POST /friend/2/shweta HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:08:04] "POST /inbox/2/shweta HTTP/1.1" 200 -
```

Figure 2: Requests Routed to Port 8001



```
sxy9369@yes:~/Courses/DS_Final_Project/New/Containers$ docker run --link mysqlserver:mysql --link mysqlserver2:mysql2 -p 8003:5000 fb2
/usr/local/lib/python2.7/dist-packages/flask_sqlalchemy/__init__.py:839: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overh
ead and will be disabled by default in the future.  Set it to True or False to suppress this warning.
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
/usr/local/lib/python2.7/dist-packages/flask_sqlalchemy/__init__.py:839: FSADeprecationWarning: SQLALCHEMY_TRACK_MODIFICATIONS adds significant overh
ead and will be disabled by default in the future.  Set it to True or False to suppress this warning.
  'SQLALCHEMY_TRACK_MODIFICATIONS adds significant overhead and '
 * Debugger is active!
 * Debugger PIN: 243-002-505
98.10.56.44 - - [19/May/2017 20:44:39] "POST /register/shweta/234/shweta@234.com/Shweta/Enjoyiingggg!!!!!!!!/IMG_20160522_202609025.jpg HTTP/1.1" 200
 -
98.10.56.44 - - [19/May/2017 20:55:41] "POST /login/shweta/234 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:56:51] "POST /friend/2/shweta HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:57:01] "POST /friend/1/raj HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:57:28] "POST /friend/1/niraj HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 20:58:22] "POST /friend/1/niraj HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:01:02] "POST /inbox/1/raj HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:05:06] "POST /friend/2/shweta HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:05:34] "POST /friend/1/niraj HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:06:46] "POST /friend/1/niraj HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:07:15] "POST /login/shweta/234 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:08:00] "POST /login/shweta/234 HTTP/1.1" 200 -
98.10.56.44 - - [19/May/2017 21:08:20] "POST /login/niraj/123 HTTP/1.1" 200 -
```

Figure 3: Requests Routed to Port 8003



DATA ON THE FIRST DATABASE

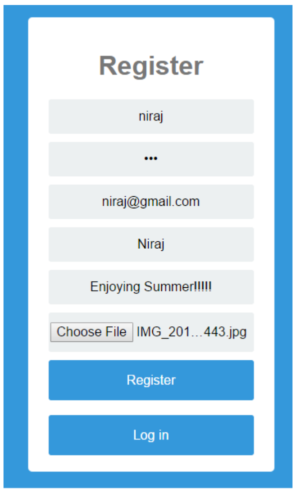
```
mysql> select * from user;
+----+----------+----------+----------------+--------+----------------------+--------------------------------+
| id | username | password | email          | Name   | status               | dp                             |
+----+----------+----------+----------------+--------+----------------------+--------------------------------+
|  1 | niraj    | 123      | niraj@123.com  | Niraj  | Heyyyaaaa!!!!!       | IMG_20151018_111403764_HDR.jpg |
|  2 | shweta   | 234      | shweta@234.com | Shweta | Enjoyiingggg!!!!!!!! | IMG_20160522_202609025.jpg     |
+----+----------+----------+----------------+--------+----------------------+--------------------------------+
2 rows in set (0.00 sec)
```

DATA ON THE SECOND DATABASE

```
+----+----------+----------+-------------+------+---------------+----------------------------+
| id | username | password | email       | Name | status        | dp                         |
+----+----------+----------+-------------+------+---------------+----------------------------+
|  1 | raj      | 456      | raj@456.com | Raj  | Copyingg!!!!! | IMG_20160522_190545361.jpg |
+----+----------+----------+-------------+------+---------------+----------------------------+
1 row in set (0.00 sec)
```

Figure 4: Distributed Database using Hashing

1.<u>New User:</u> For a new user registry, as seen in Figure(5), on filling out the profile information, the new user would be registered to a database based on the hash of the user name.


Register

niraj

•••

niraj@gmail.com

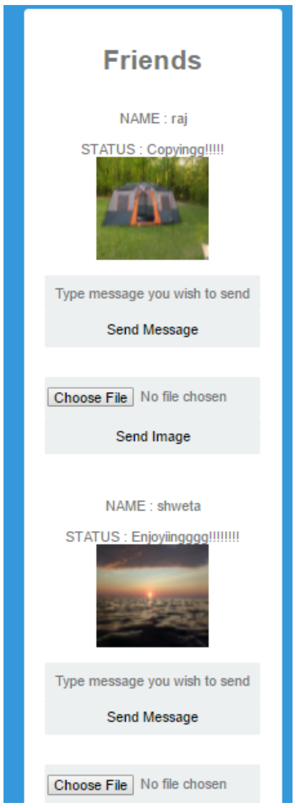
Niraj

Enjoying Summer!!!!!

Choose File IMG_201...443.jpg

Figure 5: Register Page

2.Profile of User: On logging in, the user can see their profile, the image uploaded, their status and their respective friends to whom they can send the message and image/files. All this data is obtained from the database, converted into a JSON object and rendered on the front end using Jinja and HTML. The profile of user *raj* is shown in Figure(6).


Friends

NAME : raj

STATUS : Copyingg!!!!!

Type message you wish to send

Choose File No file chosen

NAME : shweta

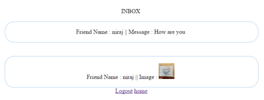
STATUS : Enjoyiingggg!!!!!!!

Type message you wish to send

Choose File No file chosen

Figure 6: Friends, Status and Sending Image/Text

3.Inbox: The Inbox of a User is as seen in Figure(7). The messages and the images received from friends can be seen here.


INBOX

Friend Name : niraj || Message : How are you

Friend Name : niraj || Image :

Figure 7: Inbox of the user

5. CONCLUSION

As described in the previous sections, the application implements the functionalities of Load-Balancing and Security and also provides some support for Fault Tolerance. As the system uses API calls and JSON object transfer, the application is secure and security attacks are prevented. Load Balancing is done by using Round Robin Technique and Data is segmented by using a Hash function.

The whole system is Distributed but functions by having the essence of a Centralized System and maintains Transparency in terms of location, concurrency as the users are not aware from which database the request is processed and also they are not aware as to how many other requests are being served simultaneously.

6. REFERENCES

1. https://rominirani.com/docker-tutorial-series-part-8-linking-containers-69a4e5bf50fb
2. http://flask.pocoo.org/docs/0.12/tutorial/
3. https://www.youtube.com/watch?v=ZVGwqnjOKjk
4. https://en.wikipedia.org/wiki/ApacheCassandra
5. https://realpython.com/blog/python/token-based-authenticatio
with-flask/
6. LaTex Read-only link: https://www.overleaf.com/read/xfcyfyrbh