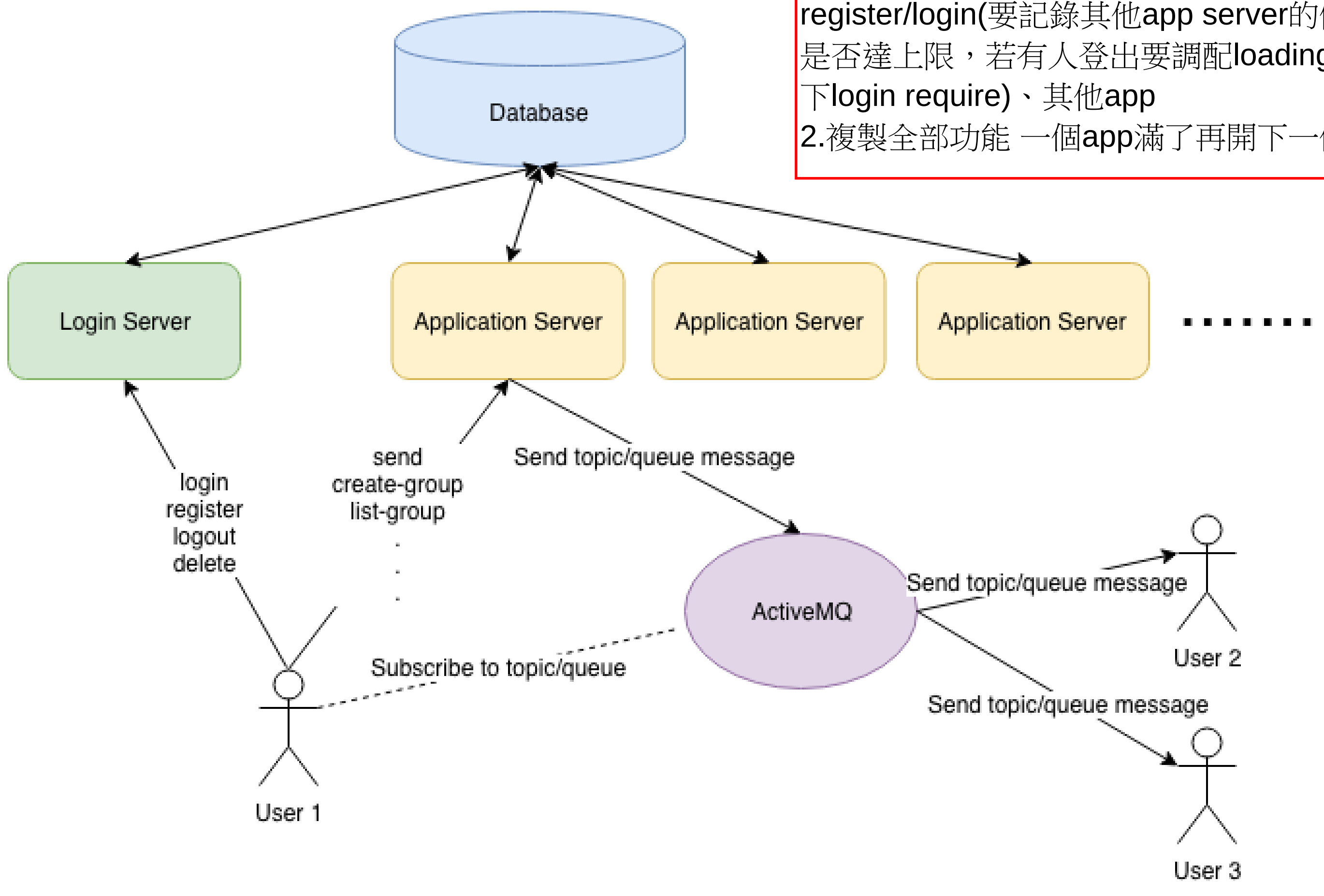
Introduction of Network Programming 2018 Autumn
Homework 5 - Distributed Server

Description

In this assignment, you are going to extend your program in homework 2, 3 and 4 in order to implement a distributed server on Amazon Web Services.

Example Architecture

1.用功能切割
register/login(要記錄其他app server的使用狀況ex人數
是否達上限，若有人登出要調配loading ex 請client重新
下login require)、其他app
2.複製全部功能 一個app滿了再開下一個



Login server deals with 4 commands, login, register, logout and delete. Application server serves the other commands as previous projects.

This is just an example architecture for your reference. If you have a better design, then you don’t have to follow this architecture.

Requirement

For client side, like previous projects, client program will help user to store their access token. Therefore, users only need to type their username.

In this project, you have to implement servers on aws. There are two kinds of servers, login server and application server. Login server keeps waiting for users’ connections. Once a user register and login, login server will assign an application server to provide services. One application server serves **ten users** at most. If there is no available application server for the upcoming users, login server should launch a new instance to operate as an application server. It is suggested that application server should be automatically terminated when no user use it. Do not make too many instances running and remember to terminate the instances that is not used anymore. When users want to logout or delete, login server should deal with these two commands.

Login server: keep listening to the connections, deal with register, login, logout, delete commands, launch and manage application servers, assign application server to users

Application server: provide services(invite, list-invite, accept-invite, list-friend, post, receive-post, send, etc)

ActiveMQ: work as project 4

General

- All the commands in project 2 & 3 & 4 must work as before.
- You have to setup an ActiveMQ server to forward the messages.
- You may have to modify the **login** feature to integrate with the message broker. For the failing cases, output the same message as project 2 & 3.
- You need to handle all the successful request and failing request as previous projects.
- Your client program must output the correct message according to different commands and different situations.
- All servers must implement on AWS.
- You server should reply the correct message according to the **priorities**.
- Output format of receiving message from other users
  - When USER_A send message “HELLO WORLD” to USER_B, your client program should show the message with following format to USER_B.
    - <<<USER_A->USER_B: HELLO WORLD>>>
  - When USER_A send message “HELLO WORLD” to GROUP_A, your client program should show the message with following format with all the group members, including USER_A.
    - <<<USER_A->GROUP<GROUP_A>: HELLO WORLD>>>

Grade (100%)

- login server - (35%)
- application server - (35%)
- management for every instance - (20%)
- oral demo - (10%)
- If you don’t implement servers on AWS, you won’t get any point.

Demo

It’s your responsibility to prepare the demo environment. TA will not help you prepare the environment, including the ActiveMQ and AWS. You can either bring your laptop, desktop, or using remote control to run the server.

When it’s your turn to demo, we will ask you to download the version you uploaded to new-e3.

1. Launch server (ip and port may change during demo): ./server 0.0.0.0 8888
2. Run with testcase: ./client TARGET_IP TARGET_PORT < testcase > output

Please learn how to run the above commands in your shell, no matter on Windows, Mac, or Linux.

Submit

Please upload a zip file called “hw5_{$student_id}.zip” that includes your source code. Submission that don’t follow the rule will get 20% punishment on the grade.

Demo time will be announced before the deadline. You are not allowed to modify your code after demo. Please submit your code on time.
If you have any questions, please ask your questions on course forum(https://e3new.nctu.edu.tw/)

Reference

1. JSON format (https://zh.wikipedia.org/wiki/JSON)
2. C socket (http://man7.org/linux/man-pages/man2/socket.2.html)
3. Python socket (https://docs.python.org/3/library/socket.html)
4. C JSON (https://github.com/json-c/json-c)
5. Python JSON (https://docs.python.org/3/library/json.html)
6. C++ ODB (https://www.codesynthesis.com/products/odb/)
7. Python peewee (http://docs.peewee-orm.com/en/latest/)
8. Activemq (http://activemq.apache.org/)
9. Boto3 (https://boto3.amazonaws.com/v1/documentation/api/latest/index.html)
10. AWS Developer (https://aws.amazon.com/tw/developer/)