CMPE273: Enterprise Distributed Systems

Lab 2 Kafka And MongoDB

Due: April 8, 2017

This lab covers designing and implementing distributed service oriented application using Kafka. This lab is graded based on 30 points and is an individual effort (no teamwork allowed)

Prerequisites:

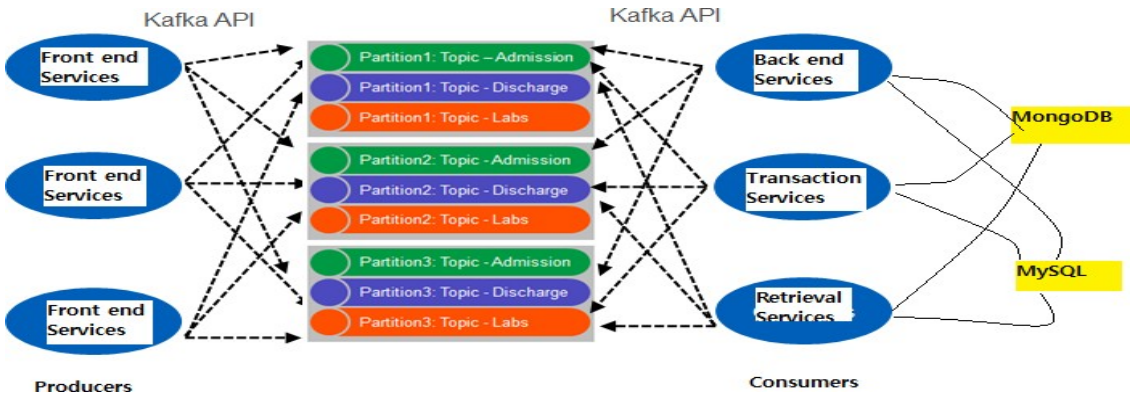
- You should be able to run Kafka sample example.
- You should have prior knowledge of JavaScript, default Sessions

Grading

- Submissions received at or before the class on the due date can receive maximum
- Late assignments will be accepted, but will be subject to a penalty of -5 points per day late

The Assignment

- You will be developing a client and server
- Separate Node into two parts connected via message queues.
  Design "backend services" as consumer and "frontend services" as producer as shown below diagram.



- Use MongoDB as the database. Sessions should be stored in **MongoDB**.
- Passwords need to be encrypted (use **passport**).
- Client and Server should communicate via **Kafka** Streams.
- On, or before the due date, you have to turn in the following:
  - Code listing of client and server
  - Document with architecture of the Kafka interaction in your client/server application, system design description and screenshots

Freelancer Application

Server - demonstrate RESTful Services (8 pts)

The next node.js based server you need to develop is the "Prototype of Freelancer application". Everyone should create the account on Freelancer and see how it functions.

This server should perform the following tasks:

a) Basic **Users** functionalities:

1. Sign up new user (Name, Email and password)
2. Sign in existing user
3. Sign out.
4. Profile (Profile Image, Name, Email, Phone Number, About Me, skills)
5. Users can update Profile anytime.

To use the system, a user must login first to the system. Password must be encrypted.

b) **Post Project** Functionality:

1. All Users can post project as an employer (Project Title, Project Description, File Upload, Skills Required, Budget Range)

c) **Home**

1. Users can see a list of all open projects. (Project *Name*, Description, Skills Required, *Employer*, Budget Range, Number of Bid yet, Bid Now)
2. There are two types of List: (a) All Projects (b) Relevant Projects (least 3 skills of freelancer match with project technology stack).
3. Include search bar where you can search project based on technology stack or Project name. Add pagination max 10 projects per page.

d) **Details View**:

1. Project Details. (Project Name, Description, Files, Skills, Budget Range, Average Bid)
2. All users can bid on the project. (Bid, Period in days)
3. List of All bids. (Profile image, Freelancer Name, Bid Price, Period in days, Hire Button only visible to an employer of the project). Include sorting filter for Bid (Ascending or Descending).
4. Hire: When an employer hires someone, the freelancer should be informed by email. A Project should be assigned to that freelancer.
5. After hiring the freelancer, Details view modify such way that only project details and submission panel (File upload, textbox) is there. [For Freelancer]
6. After hiring the freelancer, Details view modify such way that only project details, "Make Payment", and submission panel (employer can get solution) is there. [For Employee]
7. After making payment, Project should be closed. Transaction History available to both freelancer and employer. *(Note. Bid price should be deducted from employer account and added to freelancer account.)*

e) **Dashboard**
   1. List of all projects you have bid on. (Project Name, Employer, Avg. Bid, your Bid, status of project).
   2. List of all Projects you have published as employer. (Project Name, Average Bid, Freelancer Name, Estimate Project completion Date, status)
   3. Include search bar for searching project by Name. Add pagination max 10 projects per page. Add filter on status.

f) **Transaction Manager**
   - Display total fund with below two features:
     - Add Money
     - Withdraw money
   - Display incoming and outgoing transaction history.
   - Produce Pie chart which is showing incoming and outgoing income.

g) Should perform **DB provided** for database access.

The Service should take care of exception that means validation is extremely important for this server. **Proper exception handling and prototype like actual Freelancer application would attract good marks.**

Client - [*4 pts*]
A client must include all the functionalities implemented by the web services. Develop the Client using HTML5 and ReactJS-Redux. A Simple, attractive and Responsive client attracts good marks.

Note: Every field in an entire project must have validation. User's Name (Navigate to Profile) and Project Name (Navigate to Project Details view) must have hyperlinks.

Hosting - [*8 pts*]
- You must host your freelance lab to cloud.
- You can user either Heroku or Amazon Web Service to host your client and server.
- You can use MLab for your MongoDB database.

Testing of the server should be done using JMeter and Mocha.
Mocha is a node.js testing framework.

1. **Following tasks to be tested using JMeter**: **(2 Points)**
   Test the server for **100, 200, 300, 400 and 500 concurrent users (a)** without connection pooling **(b)** DB provided connection pooling and. **Draw the graph with the average time, your analysis of the graph on why, why not and how in the report.**

2. **Following tasks to be tested using Mocha**: **(2 Point)**
   Implement five randomly selected REST web service API calls using Mocha. **Display the output in the report.**

Create a private repository on the GitHub or bitbucket to manage source code for the project. Add a description for every commit. A description should consist of a one-line overview on what is committed. Include GitHub/bitbucket project link with access credentials in your report.
Regular commits to your repository are mandatory. Include GitHub /bitbucket commit History to your report.
(Penalty for not including commit history would be 3 points).

Questions (6 pts)

1. Compare passport authentication process with the authentication process used in Lab1.
2. Compare performance with and without Kafka. Explain in detail the reason for difference in performance.
3. If given an option to implement MySQL and MongoDB both in your application, specify which data of the applications will you store in MongoDB and MySQL respectively

Deliverables Required:

- Submissions shall include **source code only** for each client/server pair
- Project directory must include the group ID/Name (e.g., Lab1-caffeine)
- Archive the project, and report into one archive file (e.g., zip)
- Do not submit binaries, .class files, or supporting libraries (e.g., junit.jar, javaee.jar) (including them would be **3 points** deduction).
- Include the Readme file to document the steps to run the application.
- **All the dependencies should be added into package.json file.**
- **Project report**
    - Introduction: state your goals, purpose of system,
    - System Design: Describe your chosen system design
    - Results: Screen image captures of each client/server pair during and after run.
    - Performance: What was performance? Analyze results and explain why are you getting those results.
    - The answers to the questions.

  For example:

  Smith is submitting a project. You have provided the following files and source directory:

    - smith-lab2-report.doc
    - lab2/ *(do not send class or jar files)*
- zip –r Lab1-smith.zip Lab2

Submission

- **On-line submission**: shall include all source zipped with your last names (ex. Lab2-Smith.zip) and your report (smith_lab2_report.doc). Submissions shall be made via Canvas.