CMPE273: Enterprise Distributed Systems

Lab 2 Kafka And MongoDB

Due: 30th October 2019

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, Redux, MongoDb, Passport, JWT token, AWS EC2

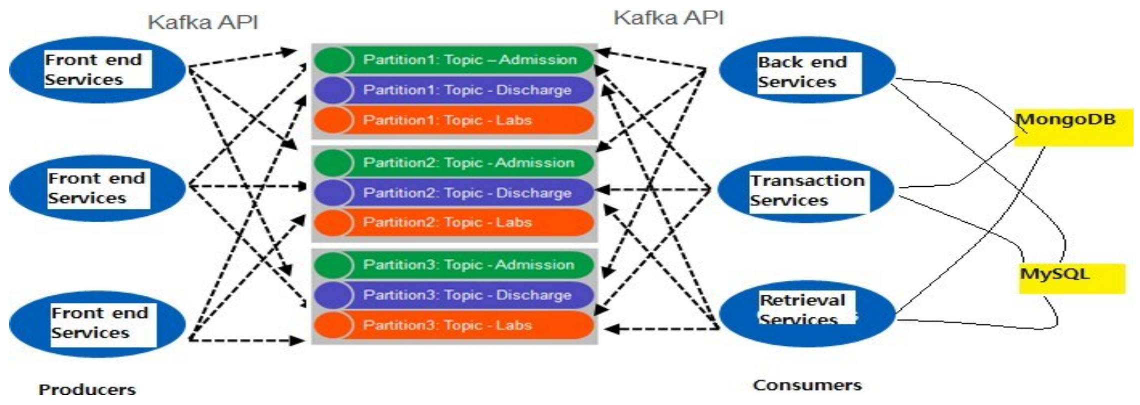
Grading

- 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
- Code Structure & Clean Code: -10pts
- Deployment (including Kafka & Mongo):5pts
- Questions:5pts
- Testing:4pts
- Client:
  - Redux:4pts
  - Additional Features:2pts
- Server
  - Mongo:2pts
  - Kafka:5pts
  - Authentication:3pts
  - Functionality:5pts

Total: 35 pts

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.



- Implement your own **connection pooling** as discussed in the lecture.
- 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

Grubhub Application

Server - demonstrate RESTful Services (15 pts)

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

This server should perform the following tasks:

a) Basic **Users (Buyer/Restaurant Owner)** functionalities:

1. Sign up new Buyer (Name, Email and password)
2. Sign up new Owner (Name, Email, password, Restaurant name, Restaurant Zip Code)
3. Sign in existing user
4. Sign out.
5. Profile of Buyer (Profile Image, Name, Email, Phone Number)
5. Profile of Owner (Profile Image, Name, Email, Phone Number, Restaurant Name, Restaurant Image, Cuisine)
6. Users can update Profile anytime.

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

b) **Owner:**

1. **Home Page/Orders:**
   a. Owner should be able to manage orders
   b. Owner should be able to see every order for his restaurant.
   c. Order details should have:
      1) Ordered Person’s name
      2) Ordered Person’s Address
      3) Item details:
         a) Items
         b) Quantity
         c) Price
      4)Status of the Order
         a) New
         b) Preparing
         c) Ready
         d) Delivered

d. Owner should be able to cancel the orders.
e. All the delivered items should be separated as old orders.

2. **MENU:**
   a. Owner should be able to add update sections (Breakfast, Lunch, Appetizers) to the menu.
   b. Owner should be able to add items to these sections.
      1) Name of item
      2) Description of item
      3) Image of item
      4) Price of item
   c. He should be able to delete or update sections.
   d. He should be able to delete or update items in a section.
   e. He should be able to see items divided based on sections in the menu view.

b) **Buyer:**

1. **Home Page:**
   d. Buyer should be able to search an item based on item name and go to search view.
2. **Search Page:**
   a. Buyer should be able to see the search results of restaurants offering that item.
   b. Buyer should be able to further filter results using cuisine.
   c. Buyers should be able to select a restaurant and go to details view.
3. **Details Page:**
   a. Buyer should be able to see items divided into sections
   b. Buyer should be able to select an item choose quantity and add it to cart.
   c. Buyer can purchase multiple items from a single restaurant at a time.
   d. The total amount should be calculated and shown to the buyer in the cart.
   e. Buyer should be able to order the items from the cart.
4. **Past Orders Page:**
   a. Buyer should be able to see all the past orders he placed to different restaurants.
5. **Upcoming Orders Page:**
   a. Buyer should be able to see all the upcoming orders and track their status as posted by the restaurant owner.
   b. Draggable order cards should be implemented.(New)

d) Messaging feature (New)

a) Messaging feature should be added to the grub hub application.
b) Owners should be able message a buyer who placed order.
c) A buyer should be able message an owner of restaurant from which he

ordered.

e) Pagination (new)

a) Pagination should be added to restaurant search, item listing and other suitable views.

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

Client -
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.

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

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.