Networked Software for Distributed Systems
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano

Project 5: Apache Kafka

Carlo Laurenti Argento
Gianluca Scanu
Federico Zanon

A.Y. 2023-2024

Contents

Introduction

Assignment

In this project, you are to implement the online services that support university courses adopting continuous evaluation. The application consists of a frontend that accepts requests from users and a backend that processes them. There are three types of users interacting with the service: (1) students enroll for courses, submit their solutions for projects, and check the status of their submissions; (2) admins can add new courses and remove old courses; (3) professors post projects and grade the solutions submitted by students. The backend is decomposed in four services, following the microservices paradigm: (1) the users service manages personal data of registered students and professors; (2) the courses service manages courses, which consists of several projects; (2) the projects service manages submission and grading of individual projects; (3) the registration service handles registration of grades for completed courses. A course is completed for a student if the student delivered all projects for that course and the sum of the grades is sufficient.

Technologies

The primary technology involved is Apache Kafka. The project also involves building a frontend service for the end user. This functionality has been implemented with a simple command line terminal which communicates with the services through socket connections.

Implementation

Design

The back-end of the application follows the **Service Oriented Architecture** (SOA): the system has been decomposed into several micro-services: `UserService`, `CourseService`, `ProjectService` and `RegistrationService`. Clients interact with these services through a front-end interface, while the micro-services communicate with each other by exchanging events in the form of JSON messages over Kafka topics, following the event-driven architecture paradigm. Each service maintains its state using *Map* data structures. Whenever a service updates its state, it publishes a record to a Kafka topic, enabling other services to update their states accordingly through a Kafka consumer. Additionally, these records serve as integral components of a fault-recovery system. In the event of a service crash and subsequent loss of its state, each service initiates a recovery process upon execution. This process involves retrieving all records from the Kafka topics and reinstating them into the respective data structures.

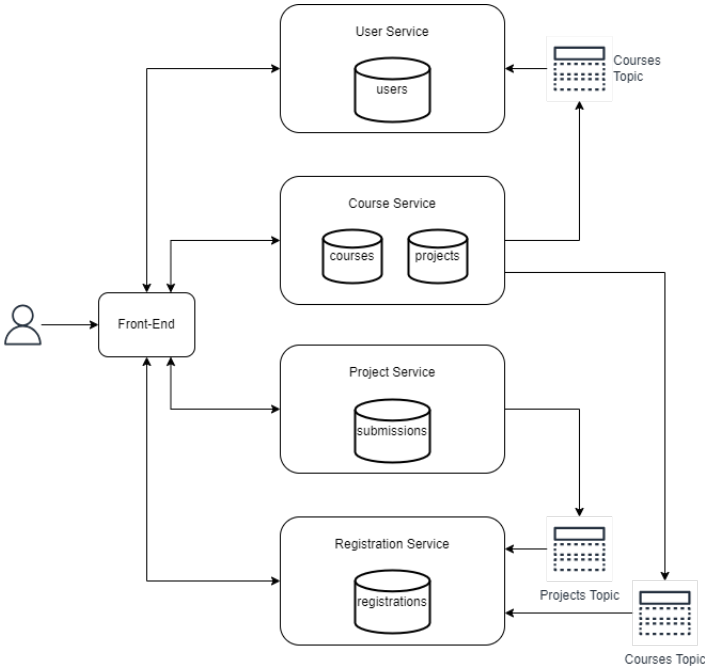

Figure 2: High-level design of the application.

Assumptions

- Kafka topics cannot be lost.
- Services can crash at any time and lose their state, when this happens the client is disconnected to avoid inconsistencies.
- It's possible to access admin functionalities by simply type "admin", this because it's not required to create a login functionality for it.
- A professor can post and grade on each existing course.
- Students can re-submit a new solution for the same project and try to have a better mark as long as the course isn't registered.
- Removing a course doesn't mean deleting the record on Kafka but changing a boolean value to make it "old" and invisible for the client.
- Students can see their registered courses on the Student Page as soon as they login to the application.

Front-end

The front-end of the application has been implemented as a simple command line terminal which uses socket connections to communicate with the services. To better visualize the application functionalities, the interface has been divided into four pages: Home Page, Student Page, Professor Page and Admin Page. There are three types of user that interact with the system and each one can do the following actions:

- **Student:**
  - **Enroll for Courses:** Interacts with the `UserService` to enroll in a course. Sends a request to the back-end to show available courses and enrolls in the desired course.
  - **Submit Solutions for Projects:** Interacts with the `ProjectService` to submit solutions for projects. Requests the back-end to show available projects for enrolled courses and then submits solutions for the available projects.
  - **Check Submission Status:** Interacts with the `ProjectService` to verify if one of the submitted solutions was graded by a professor.
- **Professor:**
  - **Post Projects:** Interacts with the `ProjectService` to post new projects to which the students can submit solutions to.
  - **Grade Solutions:** Interacts with the `ProjectService` to fetch new submissions and to grade them.

- **Admin:**
  - **Add Courses:** Interacts with the `CourseService` to add new a course.
  - **Remove Courses:** Interacts with the `CourseService` to remove a courses.
- **Common Functionality:**
  - **Registration:** Interacts with the `UserService` to register new users, by sending a registration request to the back-end.
  - **Login:** Interacts with the `UserService` for user authentication. Sends a login request to the back-end.
  - **Logout:** Logs out the user and returns to the Home Page.

UserService

The `UserService` class represents a micro-service designed to handle user registration and login. The service must also manage users' personal data, including details regarding their enrolled courses. For this reason, the service runs a thread called `UserConsumer` where a consumer is subscribed to the *courses* topic and continuously pull records in order to updates the internal state of the service. This class includes functionalities related to registration, login and course enrollment. Below are key features:

- **New User:** The `newUser` method register a new user by adding it to the local database and publishing it on the *users* topic.
- **Authenticate User:** The `authenticateUser` method is used by a user to login into the application. It checks if the credentials exist and are correct.
- **Show User Courses:** The `showUserCourses` method retrieves all the courses a student is enrolled in, including the details.
- **Enroll Course:** The `enrollCourse` method is used by a student to enroll in a course. It adds the selected course to the student's list of enrolled courses. It also publishes a record to update the user's information.

CourseService

The `CourseService` class represents a micro-service designed to handle courses and their projects. Since the state is composed by only courses and projects information, the service doesn't require interaction with the other services. This class includes functionalities related to adding, removing, and showing courses and related projects. Below are key features:

- **New Course:** The `newCourse` method is used by an admin to create a new course specifying the name, the number of cfu and the number of projects. It adds the course to the local database and publishes the course on the *courses* topic.

- **Remove Course:** The `removeCourse` method is used by an admin to remove a course by labeling it as *old*. It removes the course from the internal state and updates the record on *courses* topic. To avoid the visualization of old courses, when *CourseService* recovers its state from Kafka topic it ignores all records referring to old courses.
- **Show All Courses:** The `showAllCourses` method retrieves all the courses existing in the local state.
- **New Project:** The `newProject` method is used by a professor to add a project to a specific course. It adds the project to the local database and publishes the project on the *projects* topic.
- **Show Course Projects:** The `showCourseProjects` method retrieves all the projects existing for a specific course.

ProjectService

The `ProjectService` class represents a micro-service designed to handle project submissions. Since the state is composed by only submission information, the service doesn't require interaction with the other services. This class includes functionalities related to submitting, viewing, and grading project submissions. Below are key features:

- **Submit New Solution:** The `submitNewSolution` method processes a new solution which is uploaded by a student. It adds the submission to the local database and publishes the submission on the *submissions* topic.
- **Show New Submissions:** The `showNewSubmissions` method is used by the professor to retrieve new, ungraded, submissions for a specified project.
- **Update Submission Grade:** The `updateSubmissionGrade` method allows a professor to grade a solution by updating its grade attribute. It also publishes the updated submission on the *submissions* topic.
- **Check Submission Status:** The `checkSubmissionStatus` method is used by a student to retrieve all submissions and check the grades.

RegistrationService

The `RegistrationService` class represents a micro-service designed to handle registered courses. This is the most critic service because it needs information from `CourseService` and from `ProjectService` to check whether a course is completed for a student and, in that case, register it. To retrieve data from these services, it runs two parallel threads: `RegistrationConsumer` and `RegistrationProducer`. This class also stores all the registrations in its internal state and a user can request to see them. Below are key features:

- **RegistrationConsumer:** The `RegistrationConsumer` class is a parallel thread where a consumer is subscribed to the *courses* topic and continuously pull records in order to updates the internal state of the service.
- **RegistrationProducer:** The `RegistrationProducer` class is a parallel thread where a consumer is subscribed to the *submissions* topic and continuously pull records. When a new graded submission is detected, the thread uses the information related to the course of this submission to retrieve all the projects of this course. Then it searches for all the submissions made by the student and related to these projects. If all the projects have been completed by the student and the sum of the grades is sufficient, that course can be registered for that student: the service creates a new registration to the local database and publishes the registration on the *registrations* topic.
- **Show User Registrations:** The `showUserRegistrations` is a method which retrieves all the registrations for a specific student.

Conclusions

Testing

The four services are designed to run independently, so they can run on different distributed machines and can communicate with each other through a Kafka broker. It's important to set the IP address and port of the machine running the services and the Kafka broker, it can be easily done with the `ConfigUtils` class.

Bibliography

- Designing event driven systems book:
  `https://www.confluent.io/designing-event-driven-systems/`
- Apache Kafka official documentation:
  `https://kafka.apache.org/documentation/`