NSDS KAFKA PROJECT

Introduction

In this paper, we present the development and implementation of a Kafka-based project designed to support a microservices architecture for university course management. The project consists of several services that manage users, courses, projects, and registration processes.

This project focuses on separating the frontend and backend through Kafka events. The frontend interacts with the backend by calling API endpoints, which in turn generate Kafka events. These events are then consumed by Kafka consumers, which process them and update internal data structures. This design ensures that each service can operate independently without the need for inter-service communication, while still maintaining the ability to handle distributed tasks and data consistency.

System Design

The system is designed around a microservices architecture, where each service is autonomous and handles its own tasks independently. The frontend invokes APIs, which generate Kafka events that are processed by the backend services asynchronously. This allows each service to focus on its specific responsibilities without needing to interact directly with other services.

1. User Service:
    - Manages user registrations, authentications, and roles (e.g., students, professors, admins).
    - API calls from the frontend, such as user registration, generate Kafka events that are consumed by the User Consumer, which updates the internal state of the service by adding or updating user data.
2. Course Service:
    - Admins can create or delete courses, and students can enroll in them.
    - When a course is created, an API request from the frontend triggers an event, which is consumed by the Course Consumer. The course is then added to the internal service state. Similar processes occur for course deletions and enrollments.
3. Project Service:
    - Manages project creation and submissions. Professors can create projects, and students submit their assignments.
    - When a project is created, submitted, rated the Project Consumer processes the corresponding Kafka event.

4. Registration Service:
    - Tracks course completion for students based on project submissions and grades. When all projects for a course are completed and a student passes, this service records the course completion.
    - Kafka events related to project submissions and grading are processed to determine if a course is completed by the student.

Communication via Kafka:

In this design, services do not communicate with each other. Instead, Kafka events are triggered by frontend actions and consumed by the respective backend services. This approach ensures that the frontend and backend remain loosely coupled, while each service can focus on its core tasks independently. Kafka acts as a message broker between the API endpoints and the backend logic.

Data Structures

Each service maintains its internal state using in-memory data structures like hash maps and lists. Kafka is used to persist event data and ensure that services can recover from failures by reprocessing messages if necessary. Below are the key data structures used in the system:

1. User Data Structure:
    - The UserService stores users in a ConcurrentHashMap<String, User>, where the key is the user ID and the value is a User object. This allows for efficient retrieval and management of user data, supporting concurrent access.
2. Course Data Structure:
    - The CourseService stores courses in a ConcurrentHashMap<String, Course>, where the key is the course ID and the value is the Course object. Each course holds a list of students enrolled in the course.
3. Project Data Structure:
    - Each project represents in a ConcurrentHashMap<String, Integer> the evaluation map, where the student ID is mapped to a vote. Each project holds information about the students who submitted work and the grades they received.
4. Registration Data Structure:
    - The RegistrationService tracks completed courses for each student using a ConcurrentHashMap<String, List<String>>, where the student ID maps to a list of completed course IDs.

System Functioning

The system operates through a series of asynchronous events triggered by frontend actions and communicated to the backend via Kafka. When a user performs an action, such as registering, enrolling in a course, submitting a project, or receiving a grade, the frontend API generates a corresponding Kafka event, which is published to a specific topic. The backend services are structured to listen for events from distinct topics, ensuring that each service processes only the events relevant to its function.

The system uses three main Kafka topics for event handling:

1. user-events: Handles user registration and updates related to students, professors, and admins.
2. course-events: Manages events related to course creation, deletion, enrollment, and course completion.
3. project-events: Captures events related to project creation, submissions, and grading.

Event Handling Breakdown:

- User Registration:
    - When a user registers, the frontend API triggers a registration event that is published to the user-events topic. The User Consumer listens to this topic, processes the event, and updates the User Service with the new user data.
- Course Operations:
    - Actions such as course creation, deletion, student enrollment, and course completion are handled through the course-events topic. For example, when an admin creates a course, the API triggers a createCourse event, which is published to course-events. The Course Consumer listens to this event and updates the internal state of the Course Service by adding the new course. Similarly, enrollment and completion actions are processed through the same topic, ensuring the system is kept up-to-date on course-related changes.
- Project Operations:
    - Project-related actions (e.g., creating, submitting, and grading a project) are captured by the project-events topic. For instance, when a professor creates a new project, a createProject event is triggered and published to project-events. The Project Consumer processes this event. Submissions and grading actions follow a similar flow, ensuring that all relevant project data is maintained.

Running Example

Scenario: A student submits a project and receives a grade.

1. Project Submission:
    - The student submits their project through the frontend, triggering a ProjectSubmitted event via the API.
    - The Project Consumer listens to this event, processes the submission, and updates the internal evaluationMap for that project.
2. Project Grading:
    - The professor grades the project via the frontend, which triggers a ProjectRated event.
    - The Project Consumer processes the event, updates the student's grade in the evaluationMap.
3. Course Completion:
    - After all projects are submitted and graded, the Registration Service checks if the student has passed the course.
    - If the student has passed, the Registration Service marks the course as completed for that student and updates its internal structures.

Kafka Design Choice

Kafka was chosen for its ability to decouple the frontend from the backend services, providing a scalable and fault-tolerant solution. While services do not communicate with each other directly, Kafka ensures that events triggered by frontend actions are processed reliably by the appropriate backend services. This decoupling allows for greater flexibility and scalability in the system’s architecture.