РОЗРОБКА ТЕХНОЛОГІЇ ДЛЯ ЗАБЕЗПЕЧЕННЯ МІЖСЕРВІСНОЇ КОМУНІКАЦІЇ У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Голуб Д.К.

Харківський національний університет
радіоелектроніки, кафедра системотехніки, Україна.

E-mail: dmytro.holub1@nure.ua

Abstract

The report discusses the development process of a library for the Node.js server framework, which allows convenient and efficient management of internal service endpoints in a microservice architecture, supports multiple message brokers, and implements the request-reply and fire-and-forget principles for messaging within the infrastructure. The idea and software implementation of such a library consists of a Core component, which is an interface that the developer directly uses to interact with the library functions, a provider interface that is implemented for a number of message brokers (and can be implemented by the developer to extend the functionality and connect brokers that the library does not provide by default), and providers - classes that implement business logic specific to a particular message broker.

Мікросервісна архітектура, на сьогодні, де-факто є стандартом розробки комплексних, високонавантажених та великих за предметною галуззю проектів, вона дозволяє розбити функціонал додатку на незалежні модулі, що полегшує обслуговування та доповнення функціоналу, масштабування застосунку та дозволяє використовувати у окремому сервісі саме ті технології, які вимагає предметна галузь[1]. Разом з перевагами сервіс-орієнтованої парадигми, вона має і недоліки, серед яких – складнощі обміну повідомленнями між розподіленими модулями.

Дуже часто, коли створюється новий проект, команда бекенд-розробки з нуля пише новий модуль для комунікації між сервісами, або клонує аналогічний модуль, написаний раніше. Це викликає проблему неконсистентності кодових баз, складнощі підтримки проектів та забирає час, коли розробник міг би реалізовувати корисну бізнес-логіку на реалізацію службових функцій проекту.

Мета розробки – створити зручний інструмент, що дозволяє швидко та ефективно імплементувати функції комунікації між сервісами у сервіс-орієнтованій архітектурі. Розроблювана бібліотека повинна надавати розробнику зручне API, що дозволяє реєструвати ендпоінти, робити запити на інші сервіси та бути незмінним відносно брокеру повідомлень, що розробник буде використовувати, що дозволить розробнику не тільки швидко та ефективно перейти до імплементації основної логіки додатку, а й у разі необхідності, без суттєвої зміни кодової бази змінити технологію обміну повідомленнями.

У рамках доповіді розглядається процес розробки бібліотеки для серверного фреймворку Node.js[2], що підтримує ряд найбільш поширених брокерів повідомлень[3] – NATS, Apache Kafka, RabbitMQ та Redis Streams, дозволяє реєструвати ендпоінти, middle- та postware та реалізує принципи request-reply та fire-and-forget для обміну повідомленнями у межах інфраструктури. Ідея та програмна реалізація такої бібліотеки складається з Core компоненту, що є інтерфейсом, який розробник напряму використовує для взаємодії з функціями бібліотеки, інтерфейсу провайдера, що імплементовано для ряду брокерів повідомлень (та може бути імплементовано розробником для розширення функціоналу та підключення брокерів, що бібліотека не надає за замовченням), та провайдерів - класів, що реалізують специфічну для конкретного брокеру повідомлень бізнес-логіку.

Спрощена діаграма класів розроблюваної бібліотеки наведена на рисунку 1, вона демонструє основні компоненти програмного рішення. Клас FluffyCore є Core класом, що надає розробнику інтерфейс для роботи з бібліотекою, клас IProvider вказує методи, які класи-провайдери мають

імплементувати, це методи встановлення зв'язку з брокером повідомлень, налаштування логування, підписки, відправлення повідомлення, відповіді на повідомлення та відправки запиту. У якості підтримуваних за замовченням брокерів повідомлень було обрано NATS, Apache Kafka, RabbitMQ та Redis Streams, як найпоширеніші брокери повідомлень у мікросервісних архітектурах. Низькорівнева логіка для роботи з цими брокерами повідомлень реалізована у класах-провайдерах: NATSProvider, KafkaProvider, RabbitMQProvider та RedisStreamsProvider відповідно. Клас Message призначений для уніфікації структури повідомлень. Класи Pipeline, PendingHandler та APIError призначені для внутрішньої логіки бібліотеки.

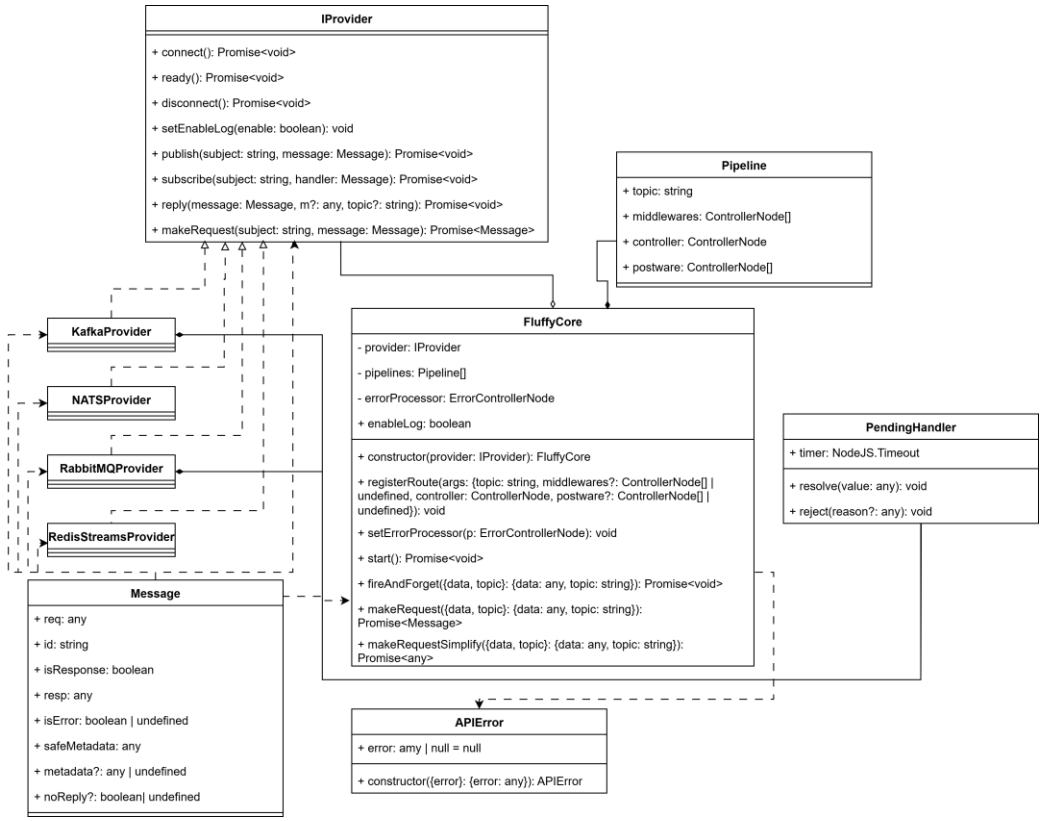

Рис. 1. Спрощена UML Class Diagram компонентів бібліотеки

Навантажувальні тести над розроблюваною бібліотекою проводились у режимі request-reply у багатопоточному режимі. Тестовий стенд включав в себе ендпоінт, створений за допомогою описаного програмного рішення та десяти «клієнтів», що звертались до цього ендпоінту. В рамках одного «клієнта» запити виконувались послідовно, а в рамках усієї інфраструктури, всі клієнти виконували свої запити одночасно. Навантажувальний тест було виконано локально на ОС Windows 10 та апаратному забезпеченні з характеристиками 16GB ОЗП та CPU Intel Core I7 8th gen. Тестування проводилось на кількостях запитів 100, 1000, 10000, 100000, 1000000. Для кожного тесту було записано час його виконання та розраховано показник RPS (запити у секунду), що є критичним показником у розподілених системах реального часу. Результати експерименту наведено у таблиці 1.

Таблиця 1. Оцінка пропускної здатності (у RPS) з розробленої бібліотекою для різних брокерів повідомлень

Requests count / Message broker	100	1000	10000	100000	1000000	RPS (avg)
	Requests per second (RPS)					
NATS	1052	2385	3343	3200	3065	2609
Apache Kafka	332	472	730	906	856	659
Redis Streams	895	876	1154	1150	1215	1058
RabbitMQ	1098	2606	4275	4134	3790	3181

Розроблене програмне рішення допоможе проектам зручніше, швидше та ефективніше розробляти серверну частину проекту та мінімізує необхідність розробляти свій механізм роботи з брокером повідомлень для кожного проекту знову і знову. Оцінка пропускної здатності показала чудові результати, рішення з такою високою пропускною здатністю вирішить потреби більшості комерційних та академічних проектів.

Технології: Розробка бібліотеки проводилася із використанням мови програмування TypeScript, серверного фреймворку Node.js, брокерів повідомлень, які найчастіше використовують при розробці розподілених систем: NATS, Apache Kafka, RabbitMQ та Redis Streams, та відповідних до них клієнтів – nats, kafkajs, amqplib та ioredis, що добре зарекомендували себе та є нативними рішеннями для роботи з вказаними брокерами повідомлень. Розробка виконувалась із використанням інтегрованого середовища розробки JetBrains WebStorm.

Висновки: Запропоноване рішення дозволяє ефективно та зручно імплементувати міжсервісну взаємодію у мікросервісних середовищах виконуючи роль проміжного слою, що дозволяє розробнику сконцентруватись на реалізації бізнес-логіки додатку та не відволікатись на підтримку логіки міжсервісної комунікації. Бібліотека також дозволяє досить легку зміну брокера повідомлень, що може стати в нагоді при використанні суміжної кодової бази на різних проектах та дозволить підібрати брокер, спираючись на реальні потреби проекту.

Література

1. Richardson C. Microservices Patterns: With examples in Java. 1st ed. Manning. 2018. 520 p.
2. Node.js Documentation. [Електронний ресурс]. Доступ до ресурсу: https://nodejs.org/docs/latest/api/
3. Newman, S. (2017). Building microservices: Designing fine-grained systems. O'Reilly Media, Inc..