DOI 10.34185/1562-9945-1-144-2023-13
УДК 004.75

Д.І. Дружинін, В.В. Герасимов

АСИНХРОННА КОМУНІКАЦІЯ МІКРОСЕРВІСІВ

Анотація. У зв'язку зі зростаючим попитом мікросервісної архітектури гостро стоїть питання асинхронної комунікації між сервісами. В роботі розглянуті особливості асинхронної комунікації, типи та патерни асинхронної комунікації та основні можливі виклики при проектуванні комунікації між сервісами. Наведені відмінності асинхронної від синхронної комунікації. Дано опис деяких популярних засобів, які застосовуються при комунікації мікросервісів, зроблено висновки щодо найбільш ефективних областей їх застосування.
Ключові слова: мікросервіс, асинхронна комунікація, брокер повідомлень, черга повідомлень, публікації/підписки, HTTP, AMQP, Kafka, Scala, RabbitMQ.

Постановка проблеми. Все більше і більше бізнес-проектів використовують мікросервісну архітектуру, де очікується широке масштабування і довга еволюція проекту з залученням великої кількості розробників. Це пов'язано із гнучкістю масштабування, можливістю безперервного розгортання і полегшенням обслуговування. За такої архітектури різні команди можуть працювати над сервісами, не впливаючи на робочі процеси в цілому, що неможливо за інших архітектурних стилів. Однією з найбільших проблем при переході на архітектуру на основі мікросервісів є зміна механізму зв'язку. Оскільки мікросервіси розподілені, вони взаємодіють один з одним за допомогою міжсервісної взаємодії на рівні мережі. Кожен мікросервіс має свій власний екземпляр та процес. Отже, служби повинні взаємодіяти з використанням протоколів міжсервісного зв'язку, таких як HTTP, gRPC або протоколу брокерів повідомлень AMQP.

Аналіз останніх досліджень та публікацій. Спектр питань, які стосуються публікацій, присвячених питанням комунікації мікросервісів та забезпечення його якості, досить широкий. Так, у [1] наведено результати дослідження патернів комунікації в мікросервісній архітектурі щодо їх особливостей, переваг і недоліків, продуктивності, зроблено висновки щодо сфер їх застосування.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

У роботі [2] наведено результати дослідження синхронної та асинхронної комунікації мікросервісної архітектури.

Основна частина. Клієнт і служби можуть спілкуватися за допомогою багатьох різних типів спілкування, кожен з яких націлений на різні сценарії та цілі. Ці типи комунікацій можна розділити на дві осі.

Перша вісь визначає, є протокол синхронним чи асинхронним.

Синхронний протокол [3]. HTTP — це синхронний протокол. Клієнт відправляє запит і чекає відповіді від сервісу. Цей протокол не залежить від виконання коду клієнта, який може бути синхронним (потік заблоковано) або асинхронним (потік не заблоковано) і відповідь зрештою досягне зворотного виклику. Важливим моментом тут є те, що протокол (HTTP/HTTPS) є синхронним і код клієнта може продовжувати виконання завдання лише тоді, коли він отримує відповідь сервера HTTP.

Асинхронний протокол [4]. Інші протоколи, такі як AMQP (протокол, який підтримується багатьма операційними системами та хмарними середовищами), використовують асинхронні повідомлення. Код клієнта або відправник повідомлення зазвичай не чекає на відповідь. Він просто надсилає повідомлення.

Синхронна комунікація підходить, якщо ваш зв'язок здійснюється лише між кількома мікросервісами. Але у випадку, коли кілька мікросервісів повинні викликати один одного та чекати довгих блокуючих операцій, повинен використовуватись асинхронний зв'язок. Інакше ця залежність і сполучення мікросервісів створить вузьке місце та серйозні проблеми архітектури. Оскільки мікросервіси — це розподілена система, яка працює на кількох процесах, сервіси повинні бути ізольовані один від одного якомога більше і взаємодіяти між собою за допомогою мережевих протоколів зв'язку, таких як синхронні протоколи HTTP, gRPC або асинхронні AMQP.

Якщо у вас є кілька мікросервісів, які повинні взаємодіяти між собою, і ви хочете взаємодіяти з ними без будь-якої залежності, повинен використовуватись асинхронний зв'язок на основі повідомлень. Клієнтська мікрослужба надсилає повідомлення до систем посередника повідомлень і не чекає відповіді. Повідомлення містять дані і надсилаються через асинхронні протоколи, такі як AMQP та через системи брокерів повідомлень, такі як Kafka та RabbitMQ.

Типи асинхронного обміну повідомленнями [5]. Існує два типи асинхронного обміну повідомленнями. Комунікація на основі повідомлень з одним

приймачем, яку назвали моделлю «брокер повідомлень» або «черга повідомлень», та комунікація на основі повідомлень із кількома приймачами, яку назвали моделлю «публікації/підписки».

Брокер повідомлень або черга повідомлень. Цей зв’язок в основному призначений для здійснення зв’язку «один-на-один» або «точка-точка». Якщо ми надсилаємо по одному запиту конкретному сервісу, і ця операція займає багато часу, тоді добре використовувати цей асинхронний зв’язок з одним отримувачем.

Зв'язок на основі моделі публікації/підписки. Ця комунікація в основному призначена для виконання механізмів публікації/підписки, які мають кілька отримувачів. Таким чином, у цій комунікації видавець публікує повідомлення, котре обробляється кількома сервісами, які підписалися на це повідомлення в системі посередника повідомлень. Ці операції публікації/підписки повинні вимагати наявності інтерфейсу шини подій для публікації подій для будь-якого підписника. Завдяки цій комунікації видавець не залежить від підписників.

Ці шаблони публікації/підписки реалізуються за допомогою шини подій. І шина подій також може мати реалізацію з системами посередників обміну повідомленнями, які підтримують асинхронну комунікацію та модель публікації/підписки, як Kafka та RabbitMQ.

Різниця між чергою повідомлень і системою обміну повідомленнями публікації/підписки. Брокери повідомлень — це програмні модулі, які дозволяють програмам, службам і системам спілкуватися та обмінюватися інформацією. Брокери повідомлень роблять це шляхом відправки повідомлень формальними протоколами обміну повідомленнями, дозволяючи взаємозалежним службам безпосередньо «розмовляти» одна з одною, навіть якщо вони написані різними мовами або працюють на інших платформах.

Посередники повідомлень перевіряють, маршрутизують, зберігають і доставляють повідомлення визначеним одержувачам. Брокери діють як посередники між іншими додатками, дозволяючи відправникам надсилати повідомлення, не знаючи місцезнаходження підписників, чи активні вони чи ні, чи навіть скільки їх існує.

В свою чергу публікація/підписка — це шаблон розповсюдження повідомлень, який дозволяє видавцям публікувати кожне повідомлення, яке вони хочуть.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Kafka [6] — це платформа розподіленої потокової передачі подій із відкритим вихідним кодом, яка забезпечує високу пропускну здатність. Написана на Java та Scala Kafka — це шина повідомлень публікації/підписки, орієнтована на потоки та передачі великого масиву даних. Замість того, щоб покладатися на чергу повідомлень, Kafka додає повідомлення до журналу та не видаляє їх, доки підписник не прочитає їх або не буде досягнуто ліміту збереження.

Архітектурні терміни та внутрішні операції Kafka:

- Mirror Maker: одним із найважливіших елементів Kafka є реплікація, яка гарантує, що повідомлення публікуються та обробляються навіть у випадку, якщо брокер стикається з проблемою.
- ZooKeeper [7]: діє як зв'язок між підписниками та брокером Kafka і зберігає такі дані про координацію, як конфігурація, розташування та деталі стану.
- Видавці надсилають або публікують повідомлення до Kafka topic, створеної на брокері Kafka, в синхронному або асинхронному режимі.
- Kafka topic — це категорії, які використовуються для групування повідомлень і мають назви, унікальні для всього кластеру Kafka.
- Підписники: сервіси, які підписалися на Kafka topic і витягують з неї повідомлення. За замовчуванням підписники зберігають повідомлення в ZooKeeper. Однак Kafka також дозволяє зберігати дані на додаткових платформах зберігання, які використовуються програмами для обробки онлайн-транзакцій (OLTP).

Kafka використовує підхід "pull-based", дозволяючи користувачам запитувати пакети повідомлень із певних вибірок. Користувачі можуть групувати повідомлення для підвищення пропускної здатності та ефективної доставки повідомлень.

Хоча Kafka поставляється лише з клієнтом Java, він пропонує адаптер SDK, що дозволяє програмістам використовувати свою унікальну системну інтеграцію. Також зростає каталог проектів екосистем спільноти та клієнтів із відкритим кодом.

Kafka найкраще використовувати для потокової передачі від пункту A до B, не вдаючись до складної маршрутизації, але з максимальною пропускною здатністю. Він також ідеально підходить для визначення джерела подій, обробки потоку та виконання моделювання змін у системі як послідовності подій. Kafka також підходить для обробки даних у багатоступеневих конвеєрах.

RabbitMQ [8] — це розподілений брокер повідомлень із відкритим кодом, який забезпечує ефективну доставку повідомлень у складних сценаріях маршрутизації. Його називають «розподіленим», тому що RabbitMQ зазвичай працює як кластер вузлів, де черги розподілені між вузлами — тиражуються для високої доступності та відмовостійкості.

Архітектурні терміни та внутрішні операції RabbitMQ:

- Підписник: він підписується на чергу та підключається до сервера брокера.
- Посередник: програми можуть обмінюватися інформацією та спілкуватися одна з одною через посередника.
- Прив'язка: повідомляє посереднику, які черги розповсюджують повідомлення. Крім того, прив'язка вказує посереднику фільтрувати повідомлення, які дозволено додавати до черги для певних типів посередника.

RabbitMQ використовує модель push-повідомлень і контролює навантаження користувачів через налаштований ліміт вибірки повідомлень. Ця модель є ідеальним підходом для обміну повідомленнями з малою затримкою між відправкою повідомлення та обробкою. Він також добре працює з архітектурою на основі черги повідомлень.

RabbitMQ використовує AMQP 0.9.1 і використовує плагіни, щоб запропонувати додаткові протоколи, такі як AMQP 1.0, HTTP, STOMP і MQTT. RabbitMQ офіційно підтримує Elixir, Go, Java, JavaScript, .NET, PHP, Python, Ruby, Objective-C, Spring і Swift. Він також підтримує різні інструменти для розробників і клієнти за допомогою плагінів спільноти.

Розробники використовують RabbitMQ для обробки високонавантажених і надійних фонових завдань, а також інтеграції та взаємодії між мікросервісами та всередині них. Програмісти також використовують RabbitMQ для виконання складної маршрутизації до підписників та інтеграції кількох програм і служб за допомогою нетривіальної логіки маршрутизації.

Відмінності між RabbitMQ і Kafka. Ці структури обміну повідомленнями підходять до обміну повідомленнями з абсолютно різних точок зору, і їхні можливості надзвичайно відрізняються.

Потік даних. RabbitMQ використовує окремий, обмежений потік даних. Повідомлення створюються та надсилаються видавцем і отримуються підписником. Apache Kafka використовує необмежений потік даних, при цьому пари ключ-значення безперервно надходять у призначений topic.

Використання даних. RabbitMQ найкраще підходить для транзакційних

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

даних, таких як формування та розміщення замовлень, а також запити користувачів. Kafka найкраще працює з оперативними даними, такими як операції процесу, статистика аудиту та журналювання, а також активність системи.

Обмін повідомленнями. RabbitMQ надсилає повідомлення користувачам. Ці повідомлення видаляються з черги після їх успішної обробки. Kafka використовує повідомлення, які залишаються в черзі до закінчення часу зберігання.

Модель дизайну. RabbitMQ використовує модель розумного брокера/примітивного підписника. Брокер постійно доставляє повідомлення підписникам і відстежує їхній статус. Кафка використовує модель примітивного брокера/розумного підписника. Kafka не відстежує повідомлення, які прочитав кожен користувач та зберігає всі повідомлення протягом встановленого періоду часу.

Топологія. RabbitMQ використовує топологію черги обміну — повідомлення направляються до різних прив'язок черги для використання підписником. Kafka використовує топологію публікації/підписки, надсилаючи повідомлення через потік до Kafka topics.

Масштабованість і резервування. RabbitMQ використовує циклічну чергу для повторення повідомлень. Щоб збільшити пропускну здатність і збалансувати навантаження, повідомлення розподіляються між чергами. Крім того, це дозволяє багатьом користувачам одночасно читати повідомлення з різних черг.

Масштабованість і резервування забезпечуються розділами Kafka. Розділ продубльований у багатьох брокерів. У випадку, якщо один із брокерів виходить з ладу, клієнта все одно може обслуговувати інший брокер.

Якщо ми зберігаємо всі розділи в одному посереднику, наша залежність від цього посередника зростатиме, що є небезпечним і збільшує ймовірність його збою. Крім того, розподіл розділів значно покращує пропускну здатність.

Таблиця 1

Відмінності між RabbitMQ і Kafka

Параметр	RabbitMQ	Kafka
Продуктивність	4 000 ÷ 10 000 повідомлень в секунду	1 мільйон повідомлень в секунду
Зберігання повідомлень	На основі підтвердження	На основі політики (наприклад, 30 днів)
Тип даних	Транзакційний	Оперативний
Споживчий режим	Розумний брокер/примітивний підписник	Примітивний брокер/розумний підписник
Топологія	Тип обміну: прямий, розгорнутий, тематичний, на основі заголовка	На основі публікації/підписки
Розмір корисного навантаження	Жодних обмежень	Обмеження за умовчанням 1 Мб
Випадки використання	Прості варіанти використання	Великі дані/висока пропускна здатність

Видалення повідомлення. Щоб видалити повідомлення з черги, RabbitMQ повинен надати підтвердження успішного отримання. Повідомлення повертаються до черги після неуспішної обробки і зберігаються підписником після успішної обробки.

Kafka зберігає повідомлення протягом визначеного часу.

Пріоритет повідомлень. У RabbitMQ повідомленням можна надати пріоритет. У Kafka всі повідомлення мають однаковий пріоритет.

Бібліотеки та підтримка мовами програмування. RabbitMQ підтримує Python, Ruby, Elixir, PHP, Swift, Go, Java, C, Spring, .Net і JavaScript.

Kafka підтримує Node js, Python, Ruby та Java.

Порядок повідомлень. RabbitMQ не підтримує порядок повідомлень. Kafka використовує topics для розрізнення повідомлень, а ZooKeeper відстежує зсув, щоб його міг використовувати будь-який підписник, який бажає прочитати topic.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Таблиця 2

Обробка повідомлень з RabbitMQ та Kafka

Параметр	RabbitMQ	Kafka
Гарантія доставки	Так	Так
Упорядкування повідомлень	Ні	Так
Пріоритети повідомлень	Так	Ні
Тривалість життя повідомлення	Повідомлення видаляються після обробки.	Повідомлення зберігаються протягом визначеного часу

Безпека та операції. І RabbitMQ, і Kafka надають вбудовані інструменти для керування безпекою та операціями. Крім того, обидві платформи пропонують інструменти сторонніх розробників для моніторингу. І Kafka, і RabbitMQ підтримують керування доступом на основі ролей (RBAC) і автентифікацію на рівні простої автентифікації та безпеки (SASL). У Kafka навіть можна керувати політиками безпеки через інтерфейс командного рядка (CLI).

Продуктивність. Може бути важко кількісно оцінити продуктивність через великий вплив коду, який взаємодіє з цими службами, та наявного апаратного забезпечення. Навіть такі речі, як швидкість мережі, пам’яті та диска, можуть значно вплинути на продуктивність служби. Незважаючи на це, RabbitMQ і Kafka оптимізовані для максимальної ефективності.

Висновок. У зв'язку зі зростаючим попитом мікросервісної архітектури дуже гостро стоїть питання асинхронної комунікації між сервісами. Оскільки мікросервіси є складною структурою, яка складається з незалежно розроблених і розгорнутих сервісів, комунікація між ними може стати вузьким місцем, тому розробники повинні бути обережними, обираючи інструменти асинхронного зв’язку.

Розглянуто найпопулярніші open source інструменти для асинхронного зв’язку між сервісами — **RabbitMQ** та Kafka. Хоча RabbitMQ і Kafka іноді взаємозамінні, їх реалізації дуже відрізняються одна від одної. Як результат, ми не можемо розглядати їх як члени однієї категорії інструментів; один — брокер повідомлень, а інший — розподілена потокова платформа.

RabbitMQ краще використовувати, коли нам потрібно: розширені та гнучкі правила маршрутизації, контроль часу повідомлень (керування закінченням

терміну дії повідомлення або затримкою повідомлення). Розширені можливості обробки помилок у випадках, коли підписники не можуть обробити повідомлення.

Kafka використовується, коли потрібне суворе впорядкування повідомлень, зберігання повідомлень протягом тривалого часу, включаючи можливість повторного відтворення минулих повідомлень та масштабування, коли традиційних рішень недостатньо.

Ми можемо реалізувати більшість варіантів використання, використовуючи обидві платформи. Однак архітектор проекту повинен вибрати найбільш підходящий інструмент для роботи. В окремих випадках під час розробки складних програмних систем може виникнути спокуса реалізувати всі необхідні варіанти використання повідомлень, використовуючи одну платформу, в той час як використання обох платформ може надати багато переваг.

ЛІТЕРАТУРА

1. Karabey Aksakalli, Işıl; Çelik, Turgay; Can, Ahmet Burak; Teki̇nerdoğan, Bedir. Deployment and communication patterns in microservice architectures : A systematic literature review. The Journal of Systems & Software. Volume 180, October 2021, 111014. DOI: https://doi.org/10.1016/j.jss.2021.111014

2. Benyamin Shafabakhsha, Robert Lagerströmb and Simon Hacksb. Evaluating the Impact of Inter Process Communication in Microservice Architectures. 8th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2020). URL: https://ceur-ws.org/Vol-2767/07-QuASoQ-2020.pdf

3. Design interservice communication for microservices.
URL: https://learn.microsoft.com/en-us/azure/architecture/microservices/design/interservice-communication

4. Communication in a microservice architecture.
URL: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture

5. Mehmet Ozkaya. Microservices Communications. URL: https://medium.com/design-microservices-architecture-with-patterns/microservices-communications-f319f8d76b71

6. Apache Kafka. URL: https://kafka.apache.org

7. Apache ZooKeeper. URL: https://zookeeper.apache.org

8. Messaging that just works — RabbitMQ. URL: https://www.rabbitmq.com

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

REFERENCES

1. Karabey Aksakalli, Işıl; Çelik, Turgay; Can, Ahmet Burak; Teki̇nerdoğan, Bedir. Deployment and communication patterns in microservice architectures : A systematic literature review. The Journal of Systems & Software. Volume 180, October 2021, 111014. DOI: https://doi.org/10.1016/j.jss.2021.111014
2. Benyamin Shafabakhsha, Robert Lagerströmb and Simon Hacksb. Evaluating the Impact of Inter Process Communication in Microservice Architectures. 8th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2020). URL: https://ceur-ws.org/Vol-2767/07-QuASoQ-2020.pdf
3. Design interservice communication for microservices.
URL: https://learn.microsoft.com/en-us/azure/architecture/microservices/design/interservice-communication
4. Communication in a microservice architecture.
URL: https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/communication-in-microservice-architecture
5. Mehmet Ozkaya. Microservices Communications. URL: https://medium.com/design-microservices-architecture-with-patterns/microservices-communications-f319f8d76b71
6. Apache Kafka. URL: https://kafka.apache.org
7. Apache ZooKeeper. URL: https://zookeeper.apache.org
8. Messaging that just works — RabbitMQ. URL: https://www.rabbitmq.com

Received 16.02.2023.
Accepted 20.02.2023.

Asynchronous communication of microservices

More and more business projects use microservice architecture, where large scale and long evolution of the project with the involvement of many developers are expected. This is due to the flexibility of scaling, the possibility of continuous deployment, ease of maintenance, and different teams can work on services without affecting the work processes as a whole, which is impossible with other architectural styles. Since microservices are a complex structure consisting of independently designed and deployed services, communication between them can become a bottleneck, so we must be careful when considering asynchronous communication tools.

The most popular open-source tools for asynchronous communication between RabbitMQ and Kafka services are considered. Although RabbitMQ and Kafka are sometimes used interchangeably, their implementations are very different from each other. As

a result, we cannot consider them as members of the same instrument category; one is a message broker, and the other is a distributed streaming platform.

RabbitMQ is best used when we need: advanced and flexible routing rules, message timing control (managing message expiration or message delay). Advanced fault handling capabilities in cases where consumers are likely to be unable to process messages (temporarily or permanently), simple implementations for consumers.

Kafka is used when strict ordering of messages is required, the storage of messages for long periods of time, including the ability to replay past messages, and the ability to achieve high scale when traditional solutions are insufficient.

We can implement most use cases using both platforms. However, the project architect must choose the most appropriate tool for the job. In making this choice, we must consider differences, as noted above. In other cases, when developing complex software systems, it may be tempting to implement all of the necessary messaging use cases using one platform when there are many advantages to using both platforms.

Keywords: microservice, asynchronous communication, message broker, message queue, publish/subscribe, HTTP, AMQP, Kafka, Scala, RabbitMQ.

Герасимов Володимир Володимирович – доцент кафедри електронних обчислювальних машин ДНУ.
Дружинін Денис Ігорович – аспірант кафедри електронних обчислювальних машин ДНУ.

Gerasymov Volodymyr Volodymyrovych — Associate Professor of Computer Systems Engineering Department of DNU
Druzhynin Denys Igorovych – PhD Student of Computer Systems Engineering Department of DNU.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)