Інформаційні технології

УДК 000.04

Піпіч Артем Андрійович

магістр комп'ютерних наук

Національного технічного університету України

«Київський політехнічний інститут імені Ігоря Сікорського»

Пипич Артём Андреевич

магистр компьютерных наук

Национального технического университета Украины

«Киевский политехнический институт имени Игоря Сикорского»

Pipich Artem

Master of Computer Science of the

National Technical University of Ukraine

«Igor Sikorsky Kyiv Polytechnic Institute»

ЗАСТОСУВАННЯ ХОРЕОГРАФІЇ В ШАБЛОНІ ПРОЕКТУВАННЯ САГА

ПРИМЕНЕНИЕ ХОРЕОГРАФИИ В ШАБЛОНЕ ПРОЕКТИРОВАНИЯ САГА

THE USE OF CHOREOGRAPHY IN SAGA DESIGN PATTERN

Анотація. *Розглянуто шаблон проектування Сага, варіацію його реалізації за використання хореографії; переваги та недоліки такого підходу.*

Ключові слова: *сервіс, транзакція, подія, публікація події, хореографія.*

Аннотация. *Рассмотрен шаблон проектирования Сага, вариация его реализации при использование хореографии; преимущества и недостатки такого подхода.*

Ключевые слова: *сервис, транзакция, событие, публикация события, хореография.*

Summary. *The design pattern of Saga, the variation of its implementation for the use of choreography was considered; the advantages and disadvantages of this approach..*

Key words: *service, transaction, event, publication of the event, choreography.*

Транзакції є важливою частиною програм. Без них неможливо зберегти узгодженість даних. Один з найбільш потужних типів транзакцій називається двоетапним комітом, суть якого в тому, що коміт першої транзакції залежить від завершення другої. Це корисно, особливо якщо доводиться одночасно оновлювати декілька об'єктів, наприклад підтвердження замовлення та оновлення доступного ліміту одночасно.

Однак, при роботі, наприклад, з мікросервісами, стан речей значно ускладнюється. Кожна служба є окремою системою, що має свою власну базу даних, і вже відсутня можливість використовувати простоту місцевих двофазних комітів для забезпечення узгодженості всієї системи.

За втрати цієї властивості, RDBMS стає досить поганим вибором для збереження, оскільки можна виконати ту саму «одиночну атомну транзакцію», але в десятки разів швидше, просто використовуючи базу даних NoSQL, таку як Couchbase. Саме тому більшість компаній, що працюють з мікросервісами, також використовують NoSQL.

Щоб продемонструвати цю проблему, розглянемо наступну архітектуру мікросервісів на високому рівні системи електронної комерції (Рисунок 1).

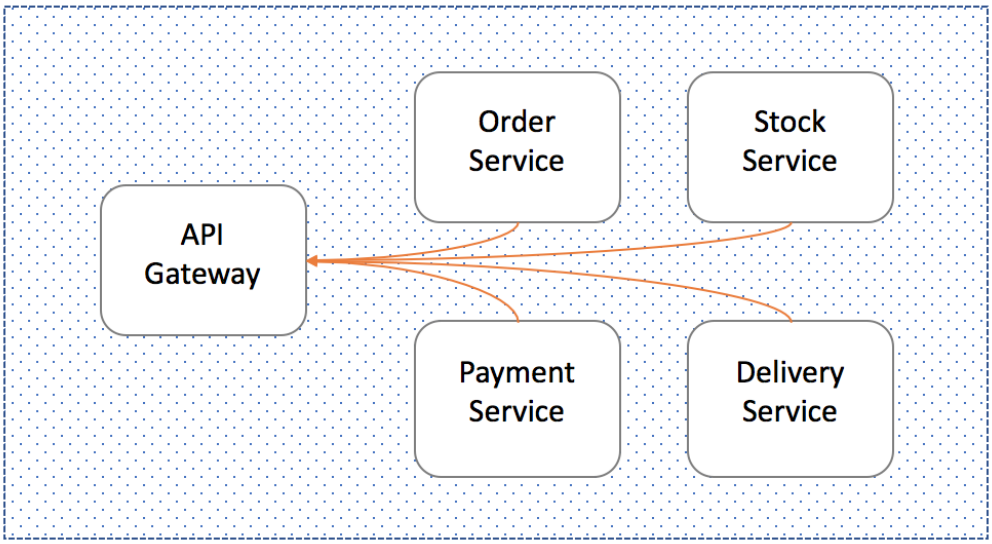

Рис. 1. Архітектура мікросервісів електронної комерції [2]

У наведеному вище прикладі не можна просто розмістити замовлення, здійснити його оплату замовником, оновлювати запас і відправити замовлення у службу доставки - все в єдиній транзакції ACID. Щоб послідовно виконати цю всю послідовність, потрібно буде створити розподілену транзакцію.

Досить розповсюдженим є досвід того, як важко реалізувати щось розподілене, і, на жаль, транзакції не є винятком. Робота з тимчасовими станами, не гарантована консистентність між службами, ізоляція та відкати - це сценарії, які всі слід розглянути на етапі проектування.

На щастя, було розроблено кілька ефективних моделей, на основі 20-річного досвіду здійснення розподілених транзакцій їхніми авторами. Те, про що піде мова в даній статті, називається шаблоном проектування Сага.

Шаблон SAGA

Одна з найбільш відомих моделей для розподілених транзакцій називається Сага. Перший документ про неї був опублікований ще в 1987 році і це зробило патерн досить популярним рішенням на той час.

Сага - це послідовність локальних транзакцій, де кожна з них оновлює дані в межах однієї служби. Перша транзакція ініціюється

зовнішнім запитом, який відповідає певній події в системі, після чого кожен наступний етап ініціюється завершенням попереднього.

Існує кілька різних способів реалізації транзакції із сагами, але два найбільш популярні:

- Події / Хореографія: коли немає центральної координації, кожна служба публікує та слухає події іншої служби та вирішує, чи потрібно застосовувати власну відповідну дію чи ні.
- Команда / Оркестровка: коли сервіс-координатор відповідає за централізацію прийняття рішень саги, а також за послідовність бізнес-логіки.

Слід трохи заглибитися у кожну з реалізацій, щоб зрозуміти, як саме вони працюють.

Події / Хореографія

У підході "Події / Хореографія" перший сервіс виконує транзакцію, після чого публікує свою подію. Ця подія слухається одним чи кількома сервісами, які виконують власні локальні транзакції та публікують (або ж ні) власні нові події.

Розподілена транзакція закінчується, коли останній сервіс виконує свою локальну транзакцію і не публікує жодних подій, або ж подія, що була опублікована, не буде слухатися жодним із учасників саги.

Таким чином, так це виглядатиме в за використання у прикладі з електронної комерції.

Order Service зберігає нове замовлення, встановлює його стан в очікування та публікує подію під назвою ORDER_CREATED_EVENT.

Payment Service слухає ORDER_CREATED_EVENT, сплачує замовлення (локальна транзакція) та публікує власну подію BILLED_ORDER_EVENT.

Stock Service слухає BILLED_ORDER_EVENT, оновлює доступний ліміт, готує продукти, куплені в замовленні, та публікує ORDER_PREPARED_EVENT.

Delivery Service прослуховує ORDER_PREPARED_EVENT, після чого забирає та доставляє товар, так як знає, що він був підготовлений у Stock Service. Зрештою, він публікує ORDER_DELIVERED_EVENT

Нарешті, Order Service прослуховує ORDER_DELIVERED_EVENT і встановлює стан замовлення як завершене.

У вищезгаданому випадку, якщо стан замовлення треба відстежувати, Order Service може просто слухати всі події та оновлювати стан.

Відкат у розподілених транзакціях

Відкат розподіленої транзакції не є аж занадто простим. Зазвичай необхідно ввести ще одну операцію / транзакцію, щоб компенсувати те, що було зроблено раніше.

Припустимо, що Stock Service не виконав свою локальну транзакцію. В такому разі відкат виглядатиме так:

- Stock Service публікує PRODUCT_OUT_OF_STOCK_EVENT;
- Order Service та Payment Service слухають дану подію;
- Payment Service повертає кошти клієнта;
- Order Service встановлює стан замовлення як невдалий

Слід звернути увагу на те, що важливо визначити зовнішній ідентифікатор для кожної транзакції, тому щоразу, коли публікується подія, всі слухачі можуть одразу дізнатися, якої транзакції це стосується.

Переваги та недоліки дизайну Подій / Хореографії Saga

Події / Хореографія є природним способом реалізації моделі Саги; вона проста для зрозуміння, не вимагає багато зусиль для побудови, а всі учасники є слабо пов'язаними (loosely coupled), оскільки не мають

безпосереднього знання один одного. Якщо певна транзакція включає в себе 2-4 кроки, застосування цього патерну може бути доволі доцільним.

Однак цей підхід може швидко стати досить заплутаним, якщо продовжувати включати додаткові кроки до транзакції, оскільки в такому випадку стає важко відстежити, які саме сервіси прослуховують певні події. Окрім того, це також може додати і циклічну залежність між сервісами, оскільки вони в такому разі мають підписатися на події один одного.

І, врешті, тестування було б доволі складним, так як передбачало б імітування поведінки транзакції, в якій повинні працювати всі сервіси.

Література

1. Pattern: Saga / Microservice Architecture. - 2017. - [Електронний ресурс]. - Режим доступу: http://microservices.io/patterns/data/saga.html - 10.04.2018.
2. Saga Pattern How to implement business transactions using Microservices - 2018. - [Електронний ресурс]. - Режим доступу: https://blog.couchbase.com/saga-pattern-implement-business-transactions-using-microservices-part/ - 12.04.2018.