УДК 004.3

**Грядченко А. О.**

# ТЕХНОЛОГІЇ АСИНХРОННОЇ КОМУНІКАЦІЇ У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

Вінницький національний технічний університет

***Анотація***

*У роботі наведено короткий огляд технологій асинхронної комунікації в мікросервісній архітектурі, здійснено огляд використання системи черг та черги повідомлень.*

**Ключові слова:** розподілена система, черга, асинхронна взаємодія.

***Abstract***

*The article provides a brief overview of asynchronous communication technologies in microservice architecture, reviews the use of queue systems and message queues.*

**Keywords:** distributed system, queue, asynchronous interaction.

## Вступ

В епоху інформаційних технологій, коли високошвидкісний доступ до даних став неодмінним для успішної діяльності в багатьох сферах людського життя, значущість ефективного обміну інформацією в системах не можна переоцінити. Це стає ще більш критичним, коли ми говоримо про високопродуктивні та масштабовані системи, які використовують мікросервісну архітектуру.

Мікросервісна архітектура характеризується розподіленням функціональності системи на декілька незалежних сервісів. Ці сервіси, кожен з яких виконує свою власну вузько спеціалізовану роль, повинні працювати разом для виконання складних завдань, що вимагає ефективної та надійної комунікації між ними. [1]

## Основна частина

Асинхронна комунікація є фундаментальним концептом, який лежить в основі мікросервісної архітектури. Вона полягає в тому, що відправник і отримувач інформації не зобов'язані бути у стані готовності одночасно. Це надзвичайно корисно у мікросервісних архітектурах, де декілька незалежних сервісів можуть працювати та обмінюватися даними без необхідності очікувати один на одного. Такий підхід дає змогу системі продовжувати роботу навіть при відмові окремих компонентів, а також дозволяє масштабувати окремі сервіси незалежно один від одного.

Використання асинхроних методів комунікації має низку переваг, у порівняні з синхроними методами комунікації. Серед них можна виділити такі:

- відмінна масштабованість: оскільки компоненти можуть обмінюватися даними без очікування відповіді, можна масштабувати окремі сервіси незалежно один від одного;
- надійність: якщо один з компонентів відмовляє, інші можуть продовжити роботу без перерв, при цьому повідомлення можуть бути опрацьовані пізніше, коли сервіс повернеться до звичайного режиму функціонування;
- поліпшення продуктивності: компоненти можуть обробляти запити у той час, коли інші зайняті, це підвищує загальну продуктивність системи;
- відсутність залежності між сервісами при розгортанні: на період розгортання нової версії сервісу повідомлення можуть зберігатись у черзі та будуть опрацьовані як тільки нова версія буде готова виконувати обробляти повідомлення;
- гнучкість системи: моніторінг черг дозволяє гнучко контролювати навантаження на кожен окремий сервіс та при необхідності повідомляти операторів системи а при наявності системи автоматизованого масштабування збільшувати кількість сервісів для обробки піків навантаження на систему;

- якщо система використовується різними організаціями (multitenant) використання черг дозволяє більш зручно розподілити ресурси між організаціями таким чином, щоб уникнути конкуренцію за ресурси між організаціями та надає можливість розділити загальні витрати на інфраструктуру між організаціями.

При виборі асинхроних методів комунікації у розподіленій системі варто також зважати на недоліки:

- складність проектування: необхідність врахування асинхронної взаємодії може ускладнити процес проектування та розробки;
- підтримка стану: в асинхронних системах може бути важко відслідковувати поточний стан об'єктів, оскільки вони можуть бути в різних станах у різний час;
- відлагодження та тестування: з огляду на асинхронну природу взаємодії, відлагодження та тестування можуть бути складнішими в порівнянні з синхронними системами;
- складність обробки помилок: на відміну від синхронних методів комунікації, де у випадку наявності помилки обробки запиту помилка буде отримана одразу, тоді як для асинхроних методів комунікації необхідно передбачити інші механізми повідомлення про помилки;
- асинхроні методи комунікації зазвичай вимагають використання додаткових компонент у системі, які, власне, і реалзують функціональність черг, що призводить до зростання складності системи та збільшення можливих точок відмови, що у свою чергу вимагає додаткового моніторінгу на та інших експлуатаційних обов’язків.

Одним з найпопулярніших підходів до асинхронної комунікації є використання систем черги повідомлень, таких як RabbitMQ, Apache Kafka або Amazon SQS [2, 3, 4]. Ці системи дозволяють сервісам відправляти повідомлення до черги, а потім інші сервіси можуть отримувати та обробляти ці повідомлення, коли вони готові. Схематично це зображено на рисунку 1.

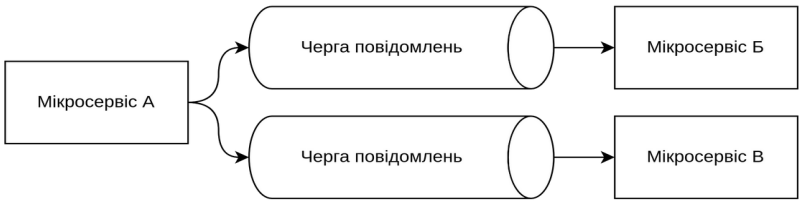


Рисунок 1. Приклад використання систем черг

Системи черги повідомлень надають велику гнучкість, тому що вони дозволяють сервісам працювати в різному темпі та обробляти навантаження відповідно до їх поточних можливостей. Це також допомагає підтримувати високий рівень надійності, оскільки повідомлення не губляться, якщо один з сервісів тимчасово стає недоступним.

У таких системах сервіс може мати з’єданння з багатьма чергами, кожна з яких пов’язує його з іншим сервісом. У такому випадку система стає більш очевидною та зв’язки між різними сервісами більш чіткі. І хоча така система досить гнучка, вона вимагає створення окремої черги під кожний зв’язок з іншим мікросервісом що призводить до додаткових операційних навантажень на керування чергами та їх моніторінг.

Іншим підходом до асинхронної комунікації є використання подійно-орієнтованих архітектур. В таких архітектурах, сервіси генерують події, коли відбувається певна зміна стану або виконується певна дія. Інші сервіси, які підписані на ці події, потім можуть реагувати на них відповідно до власної логіки обробки. Схематично це зображено на рисунку 2.


Рисунок 2. Приклад використання черги подій

Цей підхід дозволяє створювати дуже гнучкі та динамічні системи, в яких сервіси можуть легко адаптуватися до змін у додатку. Подійно-орієнтовані архітектури також часто використовуються для реалізації складних бізнес-процесів, в яких різні кроки можуть виконуватися в асинхронному порядку. Деякі популярні технології, які використовуються для підтримки подійно-орієнтованих архітектур, включають Apache Kafka та EventBridge від AWS [2, 3, 4].

У подійно-орієнтованій архітектурі сервіс створює «подію» яка передається до центральної черги (шини) до якої можуть під'єднуватись інші сервіси. У такому випадку сервіс, який продукує повідомлення та сервіс, який отримує повідомлення під'єднується лише до однієї черги, що спрощує налаштування та керування сервісами. З іншого боку, у такому випадку може бути складно контролювати перелік сервісів, які реагують на події, що може призвести до послаблення контролю за системою.

## Висновок

Технології асинхронної комунікації в мікросервісній архітектурі відіграють критичну роль у підтримці високого рівня надійності, масштабуваності та гнучкості. Вони дозволяють нам проектувати системи, в яких різні сервіси можуть працювати незалежно один від одного, що робить систему загалом більш стійкою до відмов, проте ускладнюють розробку та експлуатацію системи.

Системи черги повідомлень, такі як RabbitMQ, Apache Kafka або Amazon SQS, дозволяють розробникам проектувати архітектури, в яких повідомлення можуть бути оброблені асинхронно. Це створює надзвичайно гнучкі системи, які можуть легко адаптуватися до різних навантажень та умов роботи.

Подійно-орієнтовані архітектури також дають значні переваги, дозволяючи системам реагувати на зміни у відповідних подіях у реальному часі. Ці системи, зазвичай, дуже гнучкі та здатні до швидкої адаптації до нових вимог або змін у бізнес-логіці.

Отже, як системи черги повідомлень, так і подійно-орієнтовані архітектури мають важливі ролі у сучасній мікросервісній архітектурі. Вибір між ними залежить від конкретних вимог та контексту вашого проекту.

## СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. S. Newman, Building Microservices: Designing Fine-Grained Systems, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2021.

2. J. Vanlightly, "RabbitMQ vs Kafka Series Introduction," Jack Vanlightly. [Електронний ресурс]. Режим доступу до ресурсу: https://jack-vanlightly.com/blog/2017/12/3/rabbitmq-vs-kafka-series-introduction.

3. System Design US Blog, "Kafka vs RabbitMQ vs SQS," Medium. [Електронний ресурс]. Режим доступу до ресурсу: https://medium.com/systemdesign-us-blog/kafka-vs-rabbitmq-vs-sqs-70d1bfefa274.

4. S. Tonse, "The Big Little Guide to Message Queues," Sudhir.io. [Електронний ресурс]. Режим доступу до ресурсу: https://sudhir.io/the-big-little-guide-to-message-queues.

***Грядченко Антон Олексійович*** — студент групи 1КІ-22м, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, м. Вінниця, e-mail: antongriadchenko@gmail.com.

***Griadchenko Anton*** — student of group 1CI-22m, Faculty for Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: antongriadchenko@gmail.com