УДК 004.02, 004.455.2
DOI: https://doi.org/10.53920/ITS-2023-2-2

Костянтин Олександрович ТКАЧЕНКО,
кандидат економічних наук, доцент,
доцент кафедри інформаційних технологій,
Державний університет інфраструктури та технологій
ORCID ID: 0000-0003-0549-3396
Владислав Юрійович ЯКИМЕНКО,
магістрант кафедри інформаційних технологій,
Державний університет інфраструктури та технологій
ORCID ID: 0009-0007-4752-1259

ДЕЯКІ АСПЕКТИ ВИКОРИСТАННЯ АСИНХРОННОЇ КОМУНІКАЦІЇ У МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

У великих мікросервісних системах асинхронна комунікація є важливим інструментом для забезпечення гнучкості, масштабованості та стійкості. Така комунікація дозволяє мікросервісам працювати разом без необхідності негайної (постійної) взаємодії та сприяє створенню архітектури, що є відмінної від так званих «монолітних» систем. Мікросервіси, використовуючи асинхронну комунікацію, можуть функціонувати незалежно один від одного. Кожен мікросервіс може взаємодіяти з іншими сервісами через асинхронні події або повідомлення, незалежно від рівнів їхньої доступності або стану, в якому вони знаходяться.

Метою статті є дослідження та аналіз деяких аспектів використання асинхронної комунікації в мікросервісній архітектурі для визначення оптимальних підходів та рекомендацій щодо її впровадження в сучасних інформаційних системах. В роботі розглянуто можливості підвищення масштабованості, надійності та продуктивності мікросервісних додатків за допомогою асинхронних підходів до здійснення комунікації між сервісами. Проведено аналіз відповідних прикладів практичного застосування запропонованого підходу та підтверджено важливість і актуальність проблем, що розглядаються, в сучасному програмному інжинірингу.

Використання результатів проведеного аналізу та розроблених сервісів буде корисне для різних категорій користувачів – розробни-

ків відповідного програмного забезпечення (зокрема, програмістів і архітекторів програмного забезпечення). Вони можуть використовувати у своїх проєктах для оптимізації комунікації між сервісами запропонований авторами підхід. Для менеджерів та технічних лідерів програмних проєктів отримані результати проведеного аналізу та запропонований підхід сприятимуть кращому розумінню того, як асинхронна комунікація може вплинути на продуктивність і стабільність мікросервісних додатків. Ці аспекти вказаними категоріями користувачів та розробників можуть враховуватися при прийнятті рішень щодо архітектури своїх проєктів. Запропонований підхід має все необхідне для подальшого розвитку і впровадження в сфері мікросервісної архітектури та асинхронної комунікації.

Ключові слова: *асинхронна комунікація, мікросервіси, архітектура програмних застосунків, брокери повідомлень, високонавантажені системи, хмарні обчислення, хмарні технології.*

Kostiantyn TKACHENKO
PhD of economical sciences, associate professor,
associate professor at the department
of information technologies,
State University of Infrastructure and Technology
Vladyslav YAKYMENKO
Undergraduate at the department of information technologies,
State University of Infrastructure and Technology

SOME ASPECTS OF USING ASYNCHRONOUS COMMUNICATION IN MICROSERVICES ARCHITECTURE

In large microservices systems, asynchronous communication is an important tool to ensure flexibility, scalability, and resilience. Such communication allows microservices to work together without the need for immediate (constant) interaction and contributes to the creation of an architecture that differs from so-called «monolithic» systems. Microservices, using asynchronous communication, can operate independently of each other. Each microservice can interact with other services through asynchronous events or messages, regardless of their availability levels or states.

The purpose of this article is to explore and analyze the features and advantages of using asynchronous communication in microservices

architecture to determine optimal approaches and recommendations for its implementation in modern information systems. The paper examines the possibilities of improving the scalability, reliability, and performance of microservices applications through asynchronous communication approaches. An analysis of relevant practical examples of the proposed approach is conducted, confirming the importance and relevance of the issues under consideration in modern software engineering.

The use of the results of the analysis and the developed services will be beneficial for various categories of users, including developers of corresponding software (in particular, programmers and software architects). They can incorporate the approach proposed by the authors into their projects to optimize communication between services. For managers and technical leaders of software projects, the results of the analysis and the proposed approach will contribute to a better understanding of how asynchronous communication can affect the performance and stability of microservices applications. These aspects can be taken into account when making decisions regarding the architecture of their projects. The proposed approach has everything necessary for further development and implementation in the field of microservices architecture and asynchronous communication.

Keywords: *asynchronous communication, microservices, software architecture, message brokers, high-performance systems, cloud computing, cloud technologies.*

Постановка проблеми. В мікросервісній архітектурі [1], де програмні додатки поділяються на невеликі незалежні сервіси, асинхронна комунікація [2] стає необхідною складовою при вирішенні цілої низки важливих і актуальних проблем та практичних завдань, усунення наслідків викликів, що виникають.

В цій статті розглянуто та обгрунтовано необхідність використання асинхронної комунікації у мікросервісній архітектурі та здійснення аналізу існуючих проблем, з якими можна стикнутися при її впровадженні.

Актуальність асинхронної комунікації в мікросервісній архітектурі не викликає сумнівів, а її необхідність обумовлена, зокрема, такими факторами як:

- необхідність забезпечення незалежності мікросервісів (кожен сервіс повинен функціонувати самостійно);

- надання можливості сервісам взаємодіяти без блокування та залежностей;
- необхідність підвищення рівнів масштабованості та продуктивності, щоб сервіси мали можливість обробляти події асинхронно та реагувати на них відповідно до навантаження;
- необхідність оптимізації роботи з великим обсягом даних (зменшувати, зокрема, час очікування результатів).

При використанні асинхронної комунікації в мікросервісній архітектурі можуть виникати, зокрема такі проблеми, як:

- складність налагодження та налаштування, оскільки події відбуваються асинхронно і важко відстежити послідовність подій;
- контроль за станом системи та забезпечення консистентності [3], оскільки асинхронні події можуть виникати в непередбачуваний момент;
- складність управління повідомленнями, коли у великих системах асинхронна комунікація може призвести до значного збільшення кількості повідомлень, що потребує ефективних засобів відповідного управління та моніторингу цими повідомленнями;
- необхідність додаткових інфраструктурних рішень, коли для використання асинхронної комунікації слід розгорнути, зокрема, такі додаткові інфраструктурні компоненти, як брокери повідомлень [4] або системи черг [5].

Таким чином дослідження та аналіз використання асинхронної комунікації в мікросервісній архітектурі є актуальною проблемою, вирішення якої може призвести до розробки відповідних оптимальних підходів і рекомендацій щодо подолання цієї проблеми та забезпечення ефективної роботи мікросервісних систем.

Аналіз останніх досліджень і публікацій. Дослідження в сфері асинхронних комунікацій та мікросервісної архітектури належать до перспективних напрямків розвитку сучасного програмного інжинірингу, оскільки мікросервіси набувають популярності, і важливо розуміти, як вирішувати проблеми, пов'язані з асинхронною комунікацією.

В [6] досліджуються принципи та шаблони реалізації подійно-орієнтованих (ситуаційно-орієнтованих) мікросервісів, що використовують асинхронні комунікації.

Визначення ключових проблем, з якими можна стикнутися при впровадженні асинхронної комунікації в мікросервісній архітектурі, включаючи питання щодо керування станом, налагодження і безпеки, розглянуто в [7]. В цій роботі запропоновані й шляхи щодо вирішення визначених проблем та досягнення успішної імплементації асинхронної комунікації в мікросервісній архітектурі.

В [8] розглядається конкретний технічний стек реалізації подійно-орієнтованих мікросервісів з використанням Spring Boot (фреймворку на основі Java з відкритим кодом, розробленого компанією Pivotal Software) [9] і Apache Kafka (розподіленого сховищі подій і платформи для їх багатопотокової обробки) [10].

Різноманітні шаблони для мікросервісної архітектури, включаючи асинхронну комунікацію, розглянуто в [11]. Надані в роботі приклади реалізації вказаних шаблонів мовою програмування Java сприяють більш глибокому розумінню принципів та паттернів для мікросервісних додатків

Проведений аналіз обумовив переконаність та впевненість авторів у важливості асинхронної комунікації у мікросервісній архітектурі та обгрунтував необхідність подальших досліджень та розробки оптимальних практичних рішень для вирішення викликів запропонованого підходу.

Мета статті. Метою цієї наукової статті є проведення аналізу та дослідження проблем, пов'язаних з використанням асинхронної комунікації в мікросервісній архітектурі, визначення оптимальних стратегій і підходів для вирішення цих проблем та розробки програмного забезпечення.

Досягнення цієї мети забезпечується вирішенням, зокрема, наступних завдань:

- проведення аналізу проблем, пов'язаних з використанням асинхронної комунікації в мікросервісній архітектурі, визначення переваг та недоліків використання такої комунікації;
- визначення класів задач, які можуть бути вирішені більш ефективно за допомогою асинхронної комунікації в мікросервісній архітектурі;

- дослідження інструментарію, програмного забезпечення та технологій, адекватних потребам та можливостям асинхронної комунікації в мікросервісній архітектурі, що дозволяють збирати, аналізувати та використовувати відповідні дані для підвищення ефективності мікросервісних додатків;
- формування практичних рекомендацій (для розробників та архітекторів програмного забезпечення) щодо ефективного впровадження асинхронної комунікації в мікросервісних додатках

Мета і завдання статті спрямовані на просування інноваційних підходів та технологій, які можуть підтримати стійкий розвиток області, що розглядається, в сучасному програмному інжинірингу.

Виклад основного матеріалу дослідження. Мікросервісна архітектура виникла як реакція на недоліки монолітних додатків і стала популярною завдяки своїй гнучкості та здатності розширюватися.

Однак разом з перевагами мікросервісів з'явилися виклики, пов'язані з взаємодією між ними. Асинхронна комунікація та брокери повідомлень виникли для вирішення цих викликів і забезпечення надійної та ефективної комунікації між мікросервісами.

Мікросервіси мають функціонувати незалежно один від одного. Асинхронна комунікація дозволяє їм взаємодіяти без необхідності блокування та очікування відповідей в реальному часі. Це забезпечує вищу ступінь незалежності та розділення обов'язків між сервісами.

У великих системах з багатьма сервісами синхронна комунікація може призвести до зайвого очікування відповідей. Асинхронні повідомлення дозволяють сервісам продовжувати роботу, не очікуючи негайної відповіді. Мікросервіси часто функціонують в розподіленому середовищі, де доволі часто можуть виникати затримки та відмови.

Асинхронна комунікація допомагає забезпечити надійну взаємодію між сервісами, незалежно від їх доступності чи стану. На рис. 1 зображено типову архітектуру мікросервісного застосунку, який використовує асинхронний тип комунікації між своїми компонентами.

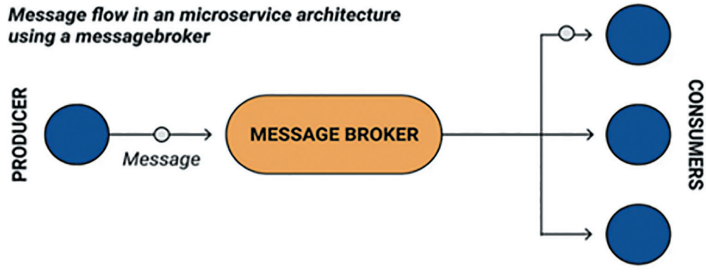

Рис. 1. Приклад використання брокера повідомлень в мікросервісній архітектурі

Джерело: [12]

Серед можливих сценаріїв використання для асинхронної комунікації та брокерів повідомлень слід виділити, зокрема такі, як:

1. Обробка замовлень та транзакцій. В онлайн-торгівлі асинхронна комунікація може використовуватися для обробки замовлень, підтвердження оплати та оновлення статусу доставки. Кожен етап операції може бути окремим сервісом, які взаємодіють асинхронно, щоб забезпечити швидку та надійну обробку інформації.

2. Обробка подій користувачів. Соціальні мережі та додатки задля спільної роботи використовують асинхронну комунікацію для реагування на дії користувачів. Наприклад, коли користувач залишає коментар, система може відправити повідомлення іншим користувачам без очікування відповіді.

3. Збір та обробка даних в реальному часі. Аналітичні системи можуть використовувати асинхронну комунікацію для збору та обробки даних в реальному часі. Дані можуть надходити з різних джерел і асинхронно оброблюватися для створення відповідних аналітичних звітів.

4. Управління конфігурацією. У системах, де потрібно розподіляти конфігурацію та оновлення параметрів, асинхронна комунікація може допомогти впроваджувати зміни без перезавантаження сервісів.

5. Інтеграція зі сторонніми сервісами. При інтеграції зі сторонніми сервісами, такими як платіжні системи чи поштові сервіси, асинхронна комунікація дозволяє ефективно взаємодіяти зі сторонніми API [13] та обробляти відповіді асинхронно.

Розглянемо підходи до реалізації асинхронної комунікації в мікросервісній архітектурі:

1. Тип комунікації «Сповіщення» (Publish-Subscribe, Pub-Sub) [14] є одним із найпоширеніших методів асинхронної комунікації в мікросервісній архітектурі. Він полягає у тому, що мікросервіси публікують події чи повідомлення в централізованому брокері повідомлень, а інші сервіси можуть підписатися на ці події та отримувати їх для подальшої обробки. Типову архітектуру мікросервісних застосунків використовуючих Pub-Sub зображено на рис. 2. Розглянемо більш детально «Сповіщення» у мікросервісній архітектурі, звертаючи увагу, зокрема, на те, що:

- мікросервіси, що публікують в брокері повідомлень власну наявну інформацію та події, якими можуть користуватися іншими сервісами. Наприклад, сервіс, що відповідає за обробку замовлень, може публікувати подію про нове замовлення. Інші мікросервіси можуть підписатися на ті типи подій, які мають для них інтерес. Наприклад, сервіс оповіщень може підписатися на події про нові замовлення, щоб надсилати сповіщення своїм користувачам. На один і той же тип події може бути підписано декілька різних сервісів (наприклад, кілька сервісів можуть підписатися на подію про оновлення профілю користувача, і реакція в кожному сервісі може бути різною);
- важливим у запропонованому підході є наявність централізованого брокеру повідомлень, наприклад, таких як Apache Kafka, RabbitMQ [15]. Брокер відповідає за збереження подій та їх розсилку, що забезпечує надійну доставку подій та розділення між виробниками (сервіси, що публікують події) і споживачами (сервіси, що підписані на події);
- є можливість більш чітко розділити обов'язки між сервісами (сервіси, які публікують події, спеціалізуються на своїх функціях, а сервіси-споживачі реагують на події, до яких в них є інтерес, без необхідності знати деталі роботи інших сервісів), що сприяє створенню реалізацій в реальному часі, бо події можуть бути оброблені майже миттєво після їх публікації.

Рис. 2. Приклад архітектури Publish-Subscribe
Джерело: [16]

2. Тип комунікації через HTTP/REST API [17] з асинхронними запитами дозволяє мікросервісам взаємодіяти асинхронно, використовуючи протокол HTTP/REST. Такий тип комунікації, також відомий як Webhook [18], зображено на рис. 3. Цей підхід відрізняється від звичайної синхронної комунікації через REST тим, що він дозволяє мікросервісам надсилати запити та отримувати відповіді асинхронно, коли це необхідно.

У цьому підході мікросервіси можуть надсилати HTTP-запити до інших сервісів, не отримуючи відповіді відразу. Замість блокуючого очікування відповіді, сервіс, що надсилає запит, може продовжувати свою роботу без очікування результату. Це особливо корисно, коли обробка запиту може зайняти багато часу. Сервіс, який отримує такий запит, може обробляти його відокремлено та відправляти відповідь тоді, коли вона готова. Отримувач може підписатися на отримання відповіді та отримати її, коли вона буде доступною.

Такий підхід особливо корисний для операцій, які можуть тривати довго, наприклад, завантаження великих файлів або складні обчислення. Сервіс може прийняти асинхронний запит, розпочати операцію і повідомити про її завершення тоді, коли вона вже виконана. В асинхронних запитах стан запиту може бути збережений для його подальшого використання. Наприклад, сервіс може надіслати асинхронний запит на обробку великої кількості даних, а потім періодично запитувати стан обробки і отримувати відповіді.

Цей тип асинхронної комунікації може бути використаний для реалізації систем сповіщень та подій. Мікросервіси можуть надсилати асинхронні запити для підписки на певні події та отримувати сповіщення про їх настання.

Відкладені запити допомагають уникнути блокування і звільняють в системі ресурси для виконання інших завдань. Це може підвищити продуктивність та ефективність системи. Використання асинхронних запитів дозволяє системі бути більш гнучкою і оперативно реагувати на зміни та навантаження.

Webhook сприяє ефективній взаємодії мікросервісів, особливо при обробці довгих операцій або некритичності точного часу, що надається на формування відповіді. Використання Webhook є доцільним при створенні гнучких і реактивних інформаційних систем.

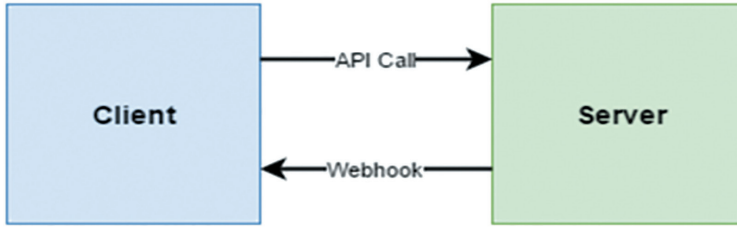

Рис. 3. Приклад HTTP/REST API з асинхронними запитами
Джерело: [18]

3. WebSocket [19] – протокол для асинхронної двосторонньої комунікації між клієнтом і сервером через одне з'єднання TCP/IP [20]. Він розроблений спеціально для вебдодатків, які потребують низького рівня затримки та високого рівня ефективності при обміні даними в режимі реального часу.

Для обміну даними між клієнтом та сервером через WebSocket слід спочатку встановити початкове з'єднання та виконують handshake [21] (так зване «рукостискання»). Handshake –процес взаємного підтвердження підтримки WebSocket і обрання версії протоколу, побудованого на принципі постійного з'єднання, що відрізняє його від традиційного HTTP, де кожен запит відкриває та закриває нове з'єднання. Постійне з'єднання дозволяє клієнту і серверу спілкуватися в реальному часі без затримок, шо пов'язані

з відкриттям/закриттям з'єднань. Діаграма послідовності операцій для Websocket зображена на рис. 4.

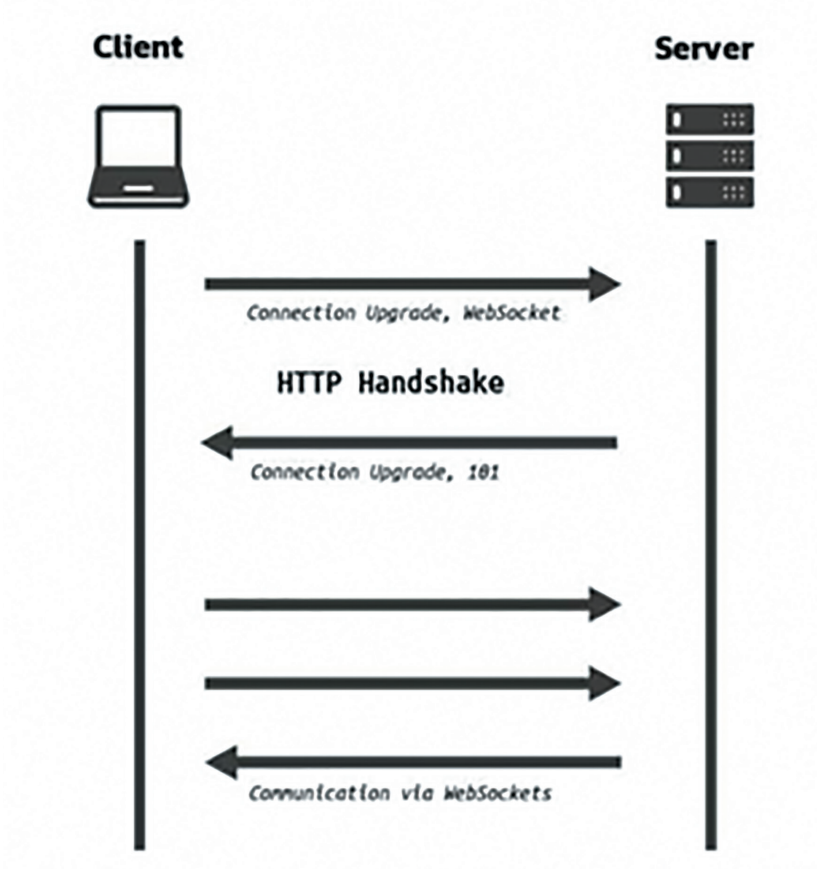

Рис. 4. Приклад використання WebSocket

Джерело: [19]

WebSocket, зокрема:

- підтримує асинхронний обмін даними в обидва напрямки (клієнт -> сервер та сервер -> клієнт), тобто і клієнт, і сервер можуть ініціювати відправку даних без очікування запиту від іншої сторони;

- підтримує передачу як бінарних, так і текстових даних, що робить його універсальним для різноманітних застосувань (від передачі текстових повідомлень до передачі зображень, відео та інших бінарних даних);
- має менший обсяг налагодження порівняно з HTTP через зменшення накладних витрат використання заголовків та оптимізований протокол обміну даними.

WebSocket ідеально підходить для додатків, які потребують режиму реального часу, таких як онлайн-чати, сповіщення, моніторинг. Він дозволяє миттєво відправляти та отримувати оновлення без необхідності постійних запитів. Для його використання у вебдодатках розробники можуть користуватися спеціалізованими бібліотеками та фреймворками, такими як Socket.IO, Spring WebSockets, або WebSocket API в браузері.

Важливим при виборі способу комунікації між мікросервісами є популярність підходу, яка дозволяє використовувати широкий спектр сучасних бібліотек та фреймворків. Все це сприяє прискоренню процесу розробки програмного продукту, що є важливим для підприємств, основним джерелом доходу яких є відповідні інформаційні системи.

Наведемо приклади різних підходів до використання популярної технології асинхронної комунікації в мікросервісній архітектурі – Apache Kafka.

На рис. 5 наведено приклад імперативного підходу до взаємодії з брокером повідомлень, з використанням нативного клієнту для Kafka.

```
while (running) {
  ConsumerRecords<K, V> records = consumer.poll(Long.MAX_VALUE);
  process(records); // application-specific processing
  try {
    consumer.commitSync();
  } catch (CommitFailedException e) {
    // application-specific rollback of processed records
  }
}
```

Рис. 5. Приклад використання нативного клієнта Apache Kafka на мові програмування Java

Джерело: авторська система

На рис. 6 зображено приклад такої ж операції – отримання повідомлень з Kafka топіку, але вже з використанням фреймворку Spring Boot.

```
7    @Component
8    @KafkaListener(id = "multiGroup", topics = "multitype")
9    public class MultiTypeKafkaListener {
10
11       @KafkaHandler
12       public void handleGreeting(Greeting greeting) {
13           System.out.println("Greeting received: " + greeting);
14       }
15
16       @KafkaHandler
17       public void handleF(Farewell farewell) {
18           System.out.println("Farewell received: " + farewell);
19       }
20
21       @KafkaHandler(isDefault = true)
22       public void unknown(Object object) {
23           System.out.println("Unkown type received: " + object);
24       }
25
26   }
```

Рис. 6. Приклад використання бібліотеки Spring Kafka

Джерело: авторська система

В прикладі на Рис. 6 використовується декларативний підхід, і завдяки цьому можна перенести реалізацію технічних аспектів взаємодії з брокером до фреймворку і зосередитися вже безпосередньо на вирішенні бізнес-проблем, що суттєво спрощує і прискорює розробку вебзастосунків.

Конфігурацію підключення до брокера повідомлень можна визначити в окремих файлах налаштувань що дозволяє розмежувати код застосунку від конфігураційних файлів, дозволяючи зберігати ці конфігурації окремо або використовувати анотації [23].

Sping Kafka забезпечує спрощення процесу розробки завдяки, зокрема:

- підтримці транзакцій, коли бібліотека надає підтримку транзакцій, що дозволяє виконувати атомарні операції над Kafka та іншими джерелами даних;
- спрощеному тестуванню через надання засобів тестування коду (для впевненості в правильності коду), що використовується Kafka;

- інтеграції з компонентами Spring Framework, такими як Spring Boot, Spring Cloud, і Spring Integration [22]; це дозволяє легко інтегрувати Kafka у вебдодатки та мікросервіси, використовуючи всю потужність Spring.

Висновки та пропозиції. В статті досліджено та проаналізовано особливості та переваги використання асинхронної комунікації в мікросервісній архітектурі. Було визначено основні аспекти цього типу комунікації, відзначено їх важливість для розробників, архітекторів, менеджерів проектів і науковців.

Таким чином, грунтуючись на результатах проведеного дослідження, можна зробити, зокрема, наступні висновки:

- асинхронна комунікація важлива для мікросервісів, забезпечуючи гнучкість, масштабованість та стійкість системи, дозволяючи мікросервісам працювати незалежно один від одного;
- використання асинхронної комунікації надає можливість для зниження негативних наслідків затримок у взаємодії між мікросервісами, коли замість блокування процесу до завершення запиту, сервіс може продовжувати свою роботу і опрацьовувати інші запити;
- асинхронна комунікація сприяє підвищенню надійності та масштабованості мікросервісів, дозволяючи легко додавати та видаляти сервіси, збільшуючи стійкість до відмов та розподіленої обробки завдань.

Крім того, розробляючи проєкти з використанням асинхронної комунікації у мікросервісних системах, слід, зокрема:

- виконати оцінювання специфічних потреб та вимог до проєкту, враховуючи фактори масштабованості, стійкості та швидкості взаємодії;
- здійснити вибір технології реалізації асинхронної комунікації, враховуючи вимоги до проєкту;
- забезпечити моніторинг і керування асинхронною комунікацією для відстеження стану та надійності системи.

ЛІТЕРАТУРА

1. What are microservices? URL: https://microservices.io/ (дата звернення: 22.09.2023).

2. Asynchronous Communication: Definition and How to Use It. URL: https://www.getguru.com/reference/synchronous-vs-asynchronous-communication (дата звернення: 22.09.2023).

3. Consistency model. URL: https://en.wikipedia.org/wiki/Consistency_model (дата звернення: 24.09.2023).

4. What are message brokers? URL: https://www.ibm.com/topics/message-brokers. (дата звернення: 24.09.2023).

5. What is Message Queueing? URL: https://www.cloudamqp.com/blog/what-is-message-queuing.html (дата звернення: 23.09.2023).

6. What do you mean by «Event-Driven»? URL: https://martinfowler.com/articles/201701-event-driven.html (дата звернення: 11.09.2023).

7. Building a Robust Microservice Architecture: Understanding Communication Patterns. URL: https://identio.fi/en/blog/building-a-robust-microservice-architecture-understanding-communication-patterns/ (дата звернення: 10.09.2023).

8. Spring Boot with Kafka for Event-Driven Microservices. URL: https://medium.com/@AlexanderObregon/spring-boot-with-kafka-for-event-driven-microservices-5f9c0e51256e. (дата звернення: 11.09.2023).

9. Spring Boot Official Documentation. URL: https://spring.io/projects/spring-boot (дата звернення: 21.09.2023).

10. Apache Kafka Official Website. URL: https://kafka.apache.org/ (дата звернення: 26.09.2023).

11. Richardson C. Microservices Patterns: With examples in Java. URL: https://www.manning.com/books/microservices-patterns (дата звернення: 12.09.2023).

12. RabbitMQ with Java, Spring and Docker, asynchronous communication between microservices. URL: https://levelup.gitconnected.com/rabbitmq-with-java-and-spring-asynchronous-communication-between-microservices-c087595c500b. (дата звернення: 09.09.2023).

13. API. URL: https://en.wikipedia.org/wiki/API (дата звернення: 19.09.2023).

14. Publisher-Subscriber pattern. URL: https://www.enjoyalgorithms.com/blog/publisher-subscriber-pattern (дата звернення: 22.09.2023).

15. RabbitMQ Official Website. URL: https://www.rabbitmq.com/ (дата звернення: 25.09.2023).

16. Event-Driven Systems: A Deep Dive into Pub/Sub Architecture. URL: https://levelup.gitconnected.com/event-driven-systems-a-deep-dive-into-pubsub-architecture-39e416be913c (дата звернення: 12.09.2023).

17. What is REST? URL: https://www.codecademy.com/article/what-is-rest (дата звернення: 14.09.2023).

18. What is a Webhook? URL: https://www.markheath.net/post/basic-introduction-webhooks. (дата звернення: 09.09.2023).

19. WebSockets Security: Main Attacks and Risks. URL: https://www.vaadata.com/blog/websockets-security-attacks-risks. (дата звернення: 09.09.2023).

20. TCP/IP. URL: https://en.wikipedia.org/wiki/TCP/IP (дата звернення: 19.09.2023).

21. TCP handshake. URL: https://developer.mozilla.org/en-US/docs/Glossary/TCP_handshake (дата звернення: 19.09.2023).

22. Spring Integration Official Documentation. URL: https://spring.io/projects/spring-integration (дата звернення: 17.09.2023).

23. An Introduction to Annotations and Annotation Processing in Java. URL: https://reflectoring.io/java-annotation-processing/ (дата звернення: 27.09.2023).

REFERENCES

1. What are microservices?, available at: https://microservices.io/ (Accessed 22 September 2023).

2. Asynchronous Communication: Definition and How to Use It, available at: https://www.getguru.com/reference/synchronous-vs-asynchronous-communication (Accessed 22 September 2023).

3. Consistency model, available at: https://en.wikipedia.org/wiki/Consistency_model (Accessed 24 September 2023).

4. What are message brokers? available at: https://www.ibm.com/topics/message-brokers (Accessed 24 September 2023).

5. What is Message Queueing? available at: https://www.cloudamqp.com/blog/what-is-message-queuing.html (Accessed 23 September 2023).

6. What do you mean by "Event-Driven"? available at: https://martinfowler.com/articles/201701-event-driven.html (Accessed 11 September 2023).

7. Building a Robust Microservice Architecture: Understanding Communication Patterns, available at: https://identio.fi/en/blog/building-a-robust-microservice-architecture-understanding-communication-patterns/ (Accessed 10 September 2023).

8. Spring Boot with Kafka for Event-Driven Microservices, available at: https://medium.com/@AlexanderObregon/spring-boot-with-kafka-for-event-driven-microservices-5f9c0e51256e (Accessed 11 September 2023).

9. Spring Boot Official Documentation, available at: https://spring.io/projects/spring-boot (Accessed 21 September 2023).

10. Apache Kafka Official Website, available at: https://kafka.apache.org/ (Accessed 26 September 2023).

11. Richardson C. Microservices Patterns: With examples in Java, available at: https://www.manning.com/books/microservices-patterns (Accessed 12 September 2023).

12. RabbitMQ with Java, Spring and Docker, asynchronous communication between microservices, available at: https://levelup.gitconnected.com/rabbitmq-with-java-and-spring-asynchronous-communication-between-microservices-c087595c500b (Accessed 09 September 2023).

13. API, available at: https://en.wikipedia.org/wiki/API (Accessed 19 September 2023).

14. Publisher-Subscriber pattern, available at: https://www.enjoyalgorithms.com/blog/publisher-subscriber-pattern (Accessed 22 September 2023).

15. RabbitMQ Official Website, available at: https://www.rabbitmq.com/ (Accessed 25 September 2023).

16. Event-Driven Systems: A Deep Dive into Pub/Sub Architecture, available at: https://levelup.gitconnected.com/event-driven-systems-a-deep-dive-into-pubsub-architecture-39e416be913c (Accessed 12 September 2023).

17. What is REST? available at: https://www.codecademy.com/article/what-is-rest (Accessed 14 September 2023).

18. What is a Webhook? available at: https://www.markheath.net/post/basic-introduction-webhooks (Accessed 09 September 2023).

19. WebSockets Security: Main Attacks and Risks, available at: https://www.vaadata.com/blog/websockets-security-attacks-risks (Accessed 09 September 2023).

20. TCP/IP, available at: https://en.wikipedia.org/wiki/TCP/IP (Accessed 19 September 2023).

21. TCP handshake, available at: https://developer.mozilla.org/en-US/docs/Glossary/TCP_handshake (Accessed 19 September 2023).

22. Spring Integration Official Documentation, available at: https://spring.io/projects/spring-integration (Accessed 17 September 2023).

23. An Introduction to Annotations and Annotation Processing in Java, available at: https://reflectoring.io/java-annotation-processing/ (Accessed 17 September 2023).

СТАТТЯ НАДІЙШЛА ДО РЕДАКЦІЇ 25.09.2023