УДК 004.75
DOI https://doi.org/10.32782/tnv-tech.2024.6.3

ТЕОРЕТИЧНІ ОСНОВИ ВІДСТЕЖЕННЯ ПОДІЙ У РОЗПОДІЛЕНИХ СИСТЕМАХ

***Вакалюк Т. А.** – доктор педагогічних наук, професор,*
завідувач кафедри інженерії програмного забезпечення
Державного університету «Житомирська політехніка»
ORCID ID: 0000-0001-6825-4697
Scopus-Author ID: 57211133927
Researcher ID: C-3650-2016

***Фант М. О.** – кандидат філологічних наук,*
доцент кафедри інженерії програмного забезпечення
Державного університету «Житомирська політехніка»
ORCID ID: 0000-0002-4994-8009

***Єфремов Ю. М.** – кандидат технічних наук, доцент,*
доцент кафедри інженерії програмного забезпечення
Державного університету «Житомирська політехніка»
ORCID ID: 0000-0002-1249-5560
Scopus-Author ID: 57213819156

***Фаррахов О. В.** – кандидат технічних наук,*
в.о. вченого секретаря Центру інформаційно-аналітичного
та технічного забезпечення моніторингу об'єктів атомної енергетики
Національної академії наук України
ORCID ID: 0000-0003-4988-126X
Scopus-Author ID: 57817543200

***Жиляєв Є. В.** – здобувач*
Державного університету «Житомирська політехніка»
ORCID ID: 0009-0000-4396-7826

У статті досліджено теоретичні засади та ключові аспекти відстеження подій у розподілених системах, що набуває особливої актуальності в контексті стрімкого розвитку мікросервісної архітектури. Проаналізовано основні характеристики та особливості мікросервісної архітектури, включаючи її переваги та недоліки порівняно з монолітною архітектурою. Визначено, що мікросервісний підхід, попри свої переваги у вигляді гнучкості розробки та масштабування, створює додаткові виклики в контексті моніторингу та відстеження подій через розподілену природу системи.

Розкрито ключові концепції систем відстеження подій, зокрема поняття «спен» (span) як базової одиниці відстеження, «трейс» (trace) для відслідковування повного шляху запиту, «багаж» (baggage) для передачі метаданих між сервісами, «теги» (tags) для забезпечення контексту та «семплінг» (sampling) для оптимізації навантаження на систему. Детально проаналізовано різні підходи до семплінгу, включаючи методи «наперед» (head-based) та «опісля» (tail-based), їх особливості та випадки застосування.

Досліджено архітектуру сучасних систем відстеження подій, яка складається з п'яти основних компонентів: проміжного програмного забезпечення, механізму поширення контексту, системи зберігання трейсів, колектора даних та інтерфейсу аналізу. Особливу увагу приділено механізмам поширення контексту та їх ролі у забезпеченні ефективного відстеження подій у розподілених системах.

Визначено, що ефективна система відстеження подій повинна забезпечувати мінімальний вплив на продуктивність основних бізнес-процесів, що досягається через асинхронну обробку даних моніторингу та оптимізовані механізми їх збору і зберігання. Результати дослідження мають практичне значення для розробників та архітекторів програмного забезпечення, які працюють з мікросервісними архітектурами та потребують ефективних інструментів для моніторингу та діагностики розподілених систем.

Ключові слова: *розподілені системи, мікросервісна архітектура, відстеження подій, спен, трейс, багаж, семплінг, моніторинг, асинхронна обробка, поширення контексту.*

Vakaliuk T. A., Fant M. O., Iefremov Yu. M., Farrakhov O. V., Zhyliaiev Ye. V. Theoretical foundations of event tracking in distributed systems

The article explores the theoretical foundations and key aspects of event tracking in distributed systems, which is becoming particularly relevant in the context of the rapid development of microservice architecture. It analyzes the main characteristics and features of microservice architecture, including its advantages and disadvantages compared to monolithic architecture. It has been determined that the microservice approach, despite its advantages in terms of development flexibility and scalability, creates additional challenges in the context of monitoring and event tracking due to the system's distributed nature.

The key concepts of event tracking systems are revealed, including the concept of "span" as a basic unit of tracking, "trace" for tracking the complete request path, "baggage" for transmitting metadata between services, "tags" for providing context, and "sampling" for optimizing system load. Different approaches to sampling are analyzed in detail, including "head-based" and "tail-based" methods, as well as their features and application cases.

The architecture of modern event tracking systems, which consists of five main components: middleware, context propagation mechanism, trace storage system, data collector, and analysis interface, is studied. Particular attention is paid to context propagation mechanisms and their role in ensuring effective event tracking in distributed systems.

It has been determined that an effective event-tracking system should ensure minimal impact on the performance of core business processes, which is achieved through asynchronous processing of monitoring data and optimized mechanisms for their collection and storage. The research results have practical significance for software developers and architects working with microservice architectures, which require practical tools for monitoring and diagnosing distributed systems.

Key words: *distributed systems, microservice architecture, event tracking, span, trace, baggage, sampling, monitoring, asynchronous processing, context propagation.*

Актуальність. Масовий перехід на мікросервісну архітектуру не сповільнює своїх темпів, та судячи зі всього, мікросервіси увійшли в тренди розробки програмного забезпечення надовго. Однак, така архітектура має свої унікальні проблеми. Запит кожного користувача, який надходить до одного мікросервісу, майже завжди перенаправляється від одного сервісу до іншого, кожен з яких може бути написаний за допомогою різних фреймворків та програмних мов. Це ставить нову задачу перед командою розробників – як зрозуміти і відстежити подорож кожного такого запиту. Ця задача складається із багатьох факторів, таких як неоднорідна природа мікросервісів, сама природа розподілених комп’ютерних систем, в яких запиту доводиться подорожувати через мережу від сервісу до сервісу без будь-яких гарантій щодо стану мережі майбутніх сервісів, та навіть самого поняття аномальної поведінки для конкретного середовища. Усе це і вимагає дослідження та оптимізації доступних методів та інструментів, які можуть допомогти у вирішені вищезгаданої задачі.

Огляд останніх досліджень і публікацій. До проблеми дослідження звертались у своїх публікаціях такі науковці, як Своєр С. (Swoyer S.) [1], Фовлер М. (Fowler M.) [2], Дж. Льюіс (J. Lewis) [2, 3], Харенко А. (Kharenko A.) [4], Шкурко Ю. (Schkuro Y.) [6, 7], Гейнріх Р. (R. Heinrich) [8], Де Камарго (de Camargo) [9], Салвадорі І. (I. Salvadori) [9], Мелло Р. (R. d. S. Mello) [9], Сікьюера Ф. (F. Siqueira) [9], Кнокхе Х. (H. Knoche) [10], та ще ряд авторів [11].

Метою дослідження є аналіз підходів до відстеження подій у розподілених системах.

Виклад основного матеріалу. Звичайні підходи для відстеження процесів дозволили легко спостерігати за простими та статичними програми з монолітною архітектурою і процес був відносно простим. У сучасному корпоративному середовищі це зовсім інша історія. Відбувається зростання сервіс-орієнтованих архітектур (SOA). Це ускладнює розуміння того, як відбуваються конкретні транзакції в різних підрозділах програми. Більше того, швидке впровадження хмарних мікросервісів зі складними середовищами зробило майже неможливим збереження актуальності звичайних рішенньі відстеження подій. Відповідю на нові виклики є системи відстеження подій у розподілених комп'ютерних системах.

Викладене вище дозволяє зрозуміти важливість розподіленого відстеження подій в корпоративній сфері, але при цьому важливо також і зрозуміти, що сама ідея того як саме розподілене відстеження має реалізуватись і які критерії є важлими при виборі рішення, знаходяться в активному розвитку, а тому головним завданням цієї роботи є визначення таких критеріїв, та аналіз існуючих інструментів, які можуть задовольнити ті чи інші критерії.

Розробка мікросервісної архітектури – це підхід до розробки програми яка являє собою набір невеликих сервісів, кожен із яких працює у своєму власному системному процесі та взаємодіє з іншими сервісами через спеціальні механізми, наприклад через програмні інтерфейси які використовують HTTP протокол. Ці сервіси побудовані з орієнтацію саме на бізнес потреби (а не на технічні потреби) та можуть незалежно один від іншого розгортатися на сервері повністю автоматизованими процессами. Існує мінімум централізованого управління такими сервісами, і тому вони можуть бути написані різними мовами програмування та використовуватися різні технології зберігання даних [2]. Кожен мікросервіс – це невелика програма, яка має власну архітектуру, складається як із самої бізнес-логіки разом так і з різних методів зв'язку з іншими сервісами. Мікросервіси часто використовують програмний інтерфейс типу REST, але відкриті і до багатьох інших методів інтеграції. Мікросервісна архітектура має ряд переваг у порівнянні з монолітною архітектурою. Розділивши програму на набір базових служб, специфічних для поставлених бізнес завдань, складність кожної з яких зведена до мінімуму, завдяки чому легше зрозуміти систему, розробляти нові можливості, а потім і розгорнути на сервері незалежно від інших сервісів. При мікросервісній архітектури також можна вибрати будь-яку підходящу мову програмування та відповідні технології для кожного окремого мікросервісу.

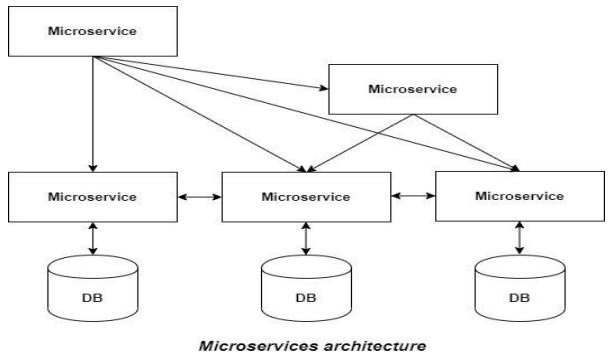

Рис. 1. Схема мікросервісу, який має п'ять незалежних сервісів з різними базами даних і з різним форматом зв'язку між сервісами

Мікросервісна архітектура суттєво впливає на зв'язок між сервісом і базою даних. В архітектурі мікросервісів замість спільного використання однієї схеми бази даних, як це відбувається при монолітній архітектурі, а кожен сервіс має свою власна схема бази даних. Деякі стверджують, що цей підхід суперечить ідеї єдиної моделі даних для всієї бізнес логіки. Крім того, це часто призводить до дублювання деяких даних. Однак, якщо для кожного сервісу є окрема схема бази даних, це забезпечує слабкий зв’язок(decoupling) між мікросервісами. Більше того, кожен сервіс може використовувати той тип бази даних, який найкраще підходить для його потреб [4].

Однак, як і монолітна архітектура, мікросервісна архітектура має недоліки. Мікросервісна архітектура створює розподілені комп’ютерні системи. При взаємодії з розподіленими системами, щоб вони функціонували належним чином, розробнику необхідно налагодити взаємодію між процесами а також продумати що робити з частковими збоями при запитах, коли місце призначення запиту може працювати занадто повільно або бути повністю недоступним. Це додає складності архітектурі такого типу. Ще один момент полягає в тому, що мікросервіс використовує базу даних, яка є лише логічним підрозділом загальної бази даних (з точки зору бізнес логіки), а отже для оновлення або видалення деяких об’єктів може бути необхідно оновити декілька баз дних, які належать різних сервісам і все це у рамках однієї єдиної транзакції. Тестування всієї програми також вимагає набагато більше роботи, ніж при роботі з програмою побудованою за монолітною архітектурою.

Хоча офіційного визначення мікросервісу немає, багато розробники домовляються про загальні характеристики мікросервісної архітектури архітектури. Наприклад, Джеймс Льюїс і Марін Фаулер визначили у своєму блозі наступні загальні характеристики мікросервісів, що представлені у табл. 1 [2].

Розглянемо основні терміни щодо систем відстеження подій.

Спен (span) – це найменша логічна одиниця розподіленої системи. Це основний блок з яких і формується розподілена система. Кожен мікросервіс, куди проходить запит, призначає такий спен, що визначає окрумий процесс та його тривалість. Різні системи з мікросервісною архітектурою мають різні дані що містяться у такому спені. Згідно з специфікацією OpenTracing API [5], вони мають містити назву операції, початок і завершення часу виконання, контекст спену, в якому дані передаються від процесу до процесу та пари типу «ключ-значення» що називаються тегами(tags) і логами(logs). Теги включають визначені користувачем назви проміжків, щоб можно було їх віднайти, фільтрувати та отримувати дані щодо відслідковування цих процесів. Логи корисні для запису даних що відносяться до конкретного діапазону, та іншої інформації необхідної для дебаггінгу [5].

Трейс (trace) охоплює весь шлях запиту у всіх сервісах, які він проходить. Він складається з усіх спенів що стосувалися цього запиту. Трейс створюється після завершення запиту. Коли запит надходить до мікросервісу, створюється унікальний глобальний ідентифікатор для відслідковування, і він не змінюється при переході до наступноїго мікросервісу. Спен охоплює кожну операцію в межах одного мікросервісу. Зазвичай мікросервіс має декілька таких спенів, оскільки під час виконання запиту він може створити один або кілька спенів.

Багаж (baggage) – це механізм поширення контексту, який використовується в OpenTracing API для захоплення, передачі та отримання метаданих. Це набір пар типу «ключ-значення», які визначаються та використовуються розподіленою

системою відслідковування подій. Поширення контексту є механізмом, який можна використовувати для цілей повністю не пов'язані з відслідковуванням подій, але при цьому має широке використання саме у цій сфері. У OpenTracing API це і називається «багаж», термін який був визначений професором Родріго Фонсека з Університету Брауна [6], оскільки дозволяє додавати довільні дані до запиту, які автоматично передаються фреймворком. Багаж передається разом із запитами що є необхідними для бізнес логіки і може бути прочитаний у будь-який момент обробки запиту.

Теги (tags) – це пари типу «ключ-значення», які пов'язані із метричними даними що використовуються для надання контексту, та надання можливісті розрізняти та групувати такі метричні дані під час їх аналізу.

Семплінг (sample) – в контексті відслідковування подій в розподілених комп'ютерних системах, семплінг є процессом генерування лише обраної підмножини трейсів, для того щоб знизити навантаження на швидкість дії та зберігання даних у системі.

Таблиця 1

Загальні характеристики мікросервісів [2]

Характеристика	Коментар
Розділ на компоненти за допомогою мікросервісів	Компонентізація функціональності в складних сервісах досягається за допомогою сервісів або мікросервісів, які є незалежними процесами, що зв'язуються один з одним через мережу. Мікросервіси призначені для забезпечення чітко вивірених інтерфейсів, малих розмірів самих сервісів, які автономно розробляються та розгортаються на сервері незалежно один від іншого.
«Розумні» (smart) сервіси та «нерозумні» (dumb) канали зв'язку	Зв'язок між службами використовує протоколи які не прив'язані до конкретних технологій, такі як HTTP і REST, на відміну від «розумних» механізмів, типу Enterprise Service Bus (ESB)
Бізнес-орієнтований дизайн	Сервіси будуються з врахуванням бізнес-потреб («сервіс профілю користувача» або «сервіс каталогу») а не технологій. Процес розробки розглядає сервіси як продукти, що постійно розвиваються, а не проекти, які в якійсь момент вважаються завершеними.
Децентралізоване управління	Дозволяє використовувати різним мікросервісам різні технологічні стеки
Децентралізоване управління даними	Проявляється в тому що рішення з точки зору концептуальних моделей даних і з точки зору технології зберігання даних приймаються відносно кожного сервісу окремо і незалежко один від іншого.
Автоматизація інфраструктури	Послуги «будуються», ідуть в реліз та розгортаються на сервері за допомогою автоматизованих процесів з використанням автоматизованого тестування, безперервної інтеграції та безперервного розгортання на сервері.
Дизайн розрахований на збій	Очікується, що сервіси завжди будуть зтикатися зі збоями інших сервісів і будуть в такому випадку або повторювати свої запити, або вміло «згортати» свій функціонал на час збою.
Еволюційний дизайн	Окремі компоненти мікросервісної архітектури розвиваються окремо один від одного, а тому не вимагають оновлення інших сервісів, які залежать від них.

Всього існує два види семплінгу, а саме семлінг «наперед» (head-based) та «опісля» (tail-based). В першому випадку, рішення про семплінг відбувається до створення необхідних даних для відслідковування події. А тому, при підході «наперед», навантаження на систему відбувається тільки у випадку якщо таке рішення було позитивним. І навпаки, при підході «опісля», генерація необхідних даних відбувається відбувається до рішення про семплінг. Сгенеровані дані надсилаются до бекенду системи відслідковування подій, а тому, дозволяє семплінгу зберігати лише найбільш інформативні дані і надавати більш повну картину подорожі окремого процессу в системі.

Використання мікросервісної архітектури для розробки як малих, так і великих програм лише зростає останні роки. Мікросервіс має ряд переваг перед монолітною архітектурою; однак це не обійшлося без додаткового ускладнення. В якості конкретного прикладу, це особливо відчувається, коли намагаєшься відстежити конкретний запит, проаналізувати його і вказати конкретне місце де що сталося і коли саме запит переходить від одного сервісу до іншого.

Відстеження подій в мікросервісній архітектурі – це саме про змогу відслідити увесь шлях, який пройшов запит починаючи від программного інтерфейсу або мобільного додатка до кожного окремого мікросервісу, звідти до бази даних і назад. Відстеження подій в розподілених системах може використовуватися для моніторингу швидкості програми. Або використовується розробниками для оптимізації їх коду, що стає все складніше, адже сучасна розробка програм все більше переходить у хмарне середовище. В даний час існує багато комерційних продуктів в цій сфері і рішень з відкритим кодом. Багато рішень для відстеження подій у розподілених системах засновані на, або ж надихалися дослідженням від Google щодо їх системи Dapper.

Розподілені системи в основному складаються з п'яти компонентів. Першим є проміжне програмне забезпечення(middleware), що має підключити систему відслідковування подій до мікросервісу. Дані збираються в точках відстеження клієнтського додатку. Запит може бути оброблений у двух точках: точці де запит надходить до сервісу і точці перед передачею його в інший сервіс. Зібрані дані в цих точках спільно і називаються единим слідом, або трейсом (trace).

Другий серед цих п'яті компонентів, з яких складається розподілена система, є поширення контексту (context propagation). Було запропоновано кілька рішень для відслідковування слідів одного запиту який проходить через різні сервіси. Поширення контексту – це коли точки відстеження отримують глобальний ідентифікатор для даних, які вони створюють, який є унікальним для кожної відстежуємої події, а передача такого глобального ідентифікатора має виконуватись разом з потоком виконання. Потім, згрупувавши такі трейси за їх ідентифікатором, увесь шлях запиту можно реконструювати.

Третім є зберігання таких трейсів. А саме спосіб передачі таких даних до способу зберігання. Інтерфейс системи відстеження подій реалізовано за допомогою конкретної бібліотеки, яка передає зібрані дані до бекенд-частини системи відстеження, як правило, використовуючи технологію відправлення таких даних в пакетах, щоб зменшити кількість окремих запитів. Звіти завжди створюються асинхронно у фоні, щоб не перешкоджати запитам які є критично важливими для бізнес-логіки програми [7]. Коли запит надходить до мікросервісу або відсилається з нього, метадані та інша інформація, пов’язані з операцією також додаються до даних необхідних для такого відслідковування. Бібліотека яка реалізує систему відслідковування збирає ці дані та передає їх до колектора (collector).

Колектор зберігає ці дані у базі даних, і потім ці дані аналізуються шляхом відстеження конкретних процесів через інтерфейс.

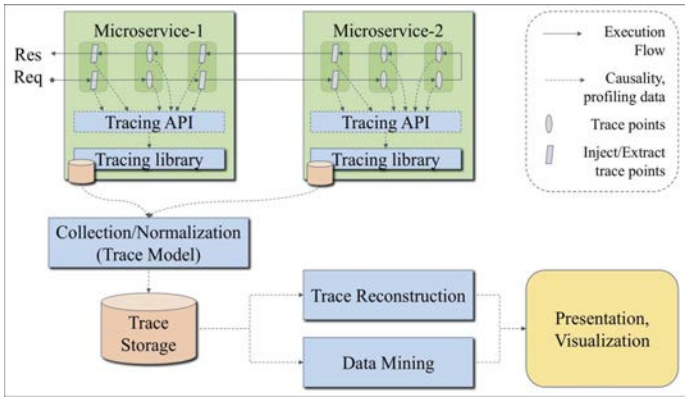

Рис. 2. Анатомія відслідковування подій в мікросервісній архітектурі

Висновки. У результаті проведеного дослідження встановлено, що відстеження подій у розподілених системах є критично важливим компонентом при роботі з мікросервісною архітектурою. Визначено основні поняття: спен (span) як найменша логічна одиниця, трейс (trace) для відслідковування повного шляху запиту, багаж (baggage) для передачі метаданих, теги (tags) для забезпечення контексту та семплінг (sampling) для оптимізації навантаження.

Проаналізовано, що сучасні розподілені системи відстеження подій складаються з п'яти основних компонентів: проміжного програмного забезпечення для підключення системи до мікросервісів, механізму поширення контексту, системи зберігання трейсів, колектора для збору даних та інтерфейсу для аналізу зібраної інформації. Встановлено, що такі системи використовують асинхронний підхід для збору та обробки даних, щоб мінімізувати вплив на основну роботу сервісів.

Дослідження показало, що при впровадженні систем відстеження подій особливу увагу слід приділяти механізмам поширення контексту та вибору оптимальної стратегії семплінгу, оскільки ці фактори безпосередньо впливають на продуктивність та ефективність всієї системи моніторингу. Визначено, що правильно налаштована система відстеження подій дозволяє ефективно діагностувати проблеми, оптимізувати продуктивність та забезпечувати надійність роботи розподілених систем на базі мікросервісної архітектури.

Перспективами подальших досліджень вбачаємо дослідження в напряму відстеження аномалій у розподілених системах.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Swoyer S., Loukides M. Microservices Adoption in 2020. URL: https://www.oreilly.com/radar/microservices-adoption-in-2020 (дата звернення: 20.12.2024).

2. Fowler M., Lewis J. Microservices a definition of this new architectural term. 2014. URL: https://www.martinfowler.com/articles/microservices.html (дата звернення: 20.12.2024).

3. Lewis J. Micro services – Java, the Unix Way. 2012. URL: http://2012.33degree.org/talk/show/67 (дата звернення: 20.12.2024).

4. Kharenko A. Monolithic vs. Microservices Architecture. 2015. URL: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59 (дата звернення: 20.12.2024).

5. OpenTracing API docs. What is a Span? 2022. URL: https://opentracing.io/docs/overview/spans (дата звернення: 20.12.2024).

6. Schkuro Y. Mastering Distributed Tracing: Analyzing performance in microservices and complex systems. Birmingham : Packt Publishing, 2019. 372 p. https://www.oreilly.com/library/view/mastering-distributed-tracing/9781788628464/

7. Shkuro Y. Embracing context propagation. 2019. URL: https://medium.com/jaegertracing/embracing-context-propagation-7100b9b6029a (дата звернення: 20.12.2024).

8. Robert Heinrich, André van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger. 2017. Performance Engineering for Microservices: Research Challenges and Directions. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion (ICPE '17 Companion). Association for Computing Machinery, New York, NY, USA, 223–226. https://doi.org/10.1145/3053600.3053653

9. André de Camargo, Ivan Salvadori, Ronaldo dos Santos Mello, and Frank Siqueira. 2016. An architecture to automate performance tests on microservices. In Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services (iiWAS '16). Association for Computing Machinery, New York, NY, USA, 422–429. https://doi.org/10.1145/3011141.3011179

10. Holger Knoche. 2016. Sustaining Runtime Performance while Incrementally Modernizing Transactional Monolithic Software towards Microservices. In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE '16). Association for Computing Machinery, New York, NY, USA, 121–124. https://doi.org/10.1145/2851553.2892039

11. T. Salah, M. Jamal Zemerly, Chan Yeob Yeun, M. Al-Qutayri and Y. Al-Hammadi, "The evolution of distributed systems towards microservices architecture," 2016 11th International Conference for Internet Technology and Secured Transactions (ICITST), Barcelona, Spain, 2016, pp. 318–325, doi: 10.1109/ICITST.2016.7856721.

REFERENCES:

1. Swoyer, S., & Loukides, M. (2020). Microservices Adoption in 2020. Retrieved from https://www.oreilly.com/radar/microservices-adoption-in-2020

2. Fowler, M., & Lewis, J. (2014). Microservices a definition of this new architectural term. Retrieved from https://www.martinfowler.com/articles/microservices.html

3. Lewis, J. (2012). Micro services – Java, the Unix Way. Retrieved from http://2012.33degree.org/talk/show/67

4. Kharenko, A. (2015). Monolithic vs. Microservices Architecture. Retrieved from https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

5. OpenTracing API docs. (2022). What is a Span? Retrieved from https://opentracing.io/docs/overview/spans

6. Schkuro Y. (2019). Mastering Distributed Tracing: Analyzing performance in microservices and complex systems. Birmingham : Packt Publishing,. 372 p. https://www.oreilly.com/library/view/mastering-distributed-tracing/9781788628464/

7. Shkuro Y. (2019). Embracing context propagation. URL: https://medium.com/jaegertracing/embracing-context-propagation-7100b9b6029a.

8. Robert Heinrich, André van Hoorn, Holger Knoche, Fei Li, Lucy Ellen Lwakatare, Claus Pahl, Stefan Schulte, and Johannes Wettinger (2017). Performance Engineering for Microservices: Research Challenges and Directions. In Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion (ICPE '17 Companion). Association for Computing Machinery, New York, NY, USA, 223–226. https://doi.org/10.1145/3053600.3053653

9. André de Camargo, Ivan Salvadori, Ronaldo dos Santos Mello, and Frank Siqueira (2016). An architecture to automate performance tests on microservices. In Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services (iiWAS '16). Association for Computing Machinery, New York, NY, USA, 422–429. https://doi.org/10.1145/3011141.3011179

10. Holger Knoche (2016). Sustaining Runtime Performance while Incrementally Modernizing Transactional Monolithic Software towards Microservices. In Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering (ICPE '16). Association for Computing Machinery, New York, NY, USA, 121–124. https://doi.org/10.1145/2851553.2892039

11. T. Salah, M. Jamal Zemerly, Chan Yeob Yeun, M. Al-Qutayri and Y. Al-Hammadi (2016). "The evolution of distributed systems towards microservices architecture," 11th International Conference for Internet Technology and Secured Transactions (ICITST), Barcelona, Spain, pp. 318-325, doi: 10.1109/ICITST.2016.7856721.