УДК 004.78
DOI https://doi.org/10.32782/2663-5941/2024.6.2/12

Козуб В.В.
Державний університет «Київський авіаційний інститут»

ПРАКТИКА ВИКОРИСТАННЯ НОВИХ ТЕХНОЛОГІЙ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ ПОБУДОВИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У статті здійснено аналіз новітніх технологій мікросервісної архітектури, яка вже давно є ключовою концепцією в розробці гнучких, масштабованих та надійних програмних систем. Мікросервісний підхід дозволяє розбивати додатки на незалежні компоненти, що взаємодіють через чітко визначені інтерфейси, що забезпечує модульність і масштабованість. Методи дослідження включають огляд літератури, аналіз існуючих технологій і підходів у сфері мікросервісів, а також огляд прикладів застосування новітніх інструментів для вирішення проблем, що виникають при інтеграції, масштабуванні та безпеці. Зокрема, акцентовано на нових інструментах автоматизації процесів розгортання та тестування, технологіях для забезпечення спостережуваності, методах оптимізації продуктивності, а також інтеграції штучного інтелекту для прогнозування навантажень і автоматизованого масштабування. Основні результати дослідження включають структуризацію основних напрямів, в яких відбувається вдосконалення мікросервісної архітектури, зокрема: автоматизація пайплайнів для тестування й розгортання, використання технологій для кешування та прискорення доступу до даних, інтеграція OpenTelemetry і Prometheus для моніторингу, а також інтеграція AI для аналізу і прогнозування навантажень. Також виявлено значення гібридних архітектур, які дозволяють поступово інтегрувати нові технології без порушення роботи існуючих систем. Наукова новизна роботи полягає в комплексному аналізі новітніх технологій та їхнього впливу на ефективність мікросервісної архітектури, а також у виявленні перспективних напрямків для подальших досліджень, спрямованих на подолання існуючих проблем в інтеграції, безпеці та масштабуванні. Важливим внеском є також вивчення впливу штучного інтелекту та автоматизації на покращення ефективності управління мікросервісами. У висновках зазначено, що для подальшого вдосконалення мікросервісної архітектури необхідно активніше інтегрувати новітні інструменти автоматизації, машинного навчання і штучного інтелекту, що дозволяють забезпечити ефективне управління ресурсами, зниження затримок та поліпшення безпеки. Подальші дослідження повинні орієнтуватися на розв'язання питань інтеграції старих і нових систем, а також на вдосконалення процесів оркестрації та спостережуваності.

Ключові слова: *архітектура мікросервісів, різноманітніття розгортання, балансування навантаження, стійкість системи, оптимізація ресурсів.*

Постановка проблеми. Мікросервісна архітектура вже тривалий час є ключовою концепцією у розробці програмного забезпечення, що дозволяє будувати гнучкі, масштабовані та стійкі системи. Мікросервісна архітектура – це підхід до побудови програмного забезпечення, у якому додаток розбивається на невеликі незалежні компоненти (мікросервіси). Кожен мікросервіс виконує певну функцію та може працювати автономно, використовуючи чітко визначені інтерфейси для взаємодії з іншими компонентами. Цей підхід відрізняється від монолітної архітектури, де вся функціональність інтегрована в єдину кодову базу.

При використанні мікросервісної архітектури забезпечується модульність, незалежність розгортання та спрощення обслуговування ПЗ, що робить її незамінною для сучасних хмарних платформ, мобільних додатків, фінтех-рішень та промислових IoT-систем [1, с. 16638; 2, с. 196]. Але разом з перевагами мікросервісам притаманні численні недоліки, вирішення яких є викликом для фахівців при побудові нових систем:

1) складність інтеграції, яка обумовлена необхідністю використання складної інфраструктури для управління взаємодією між десятками (сотнями) мікросервісів (API Gateway, Service Mesh) і вимогами до забезпечення безперервного з'єднання та обробки запитів у системах [3];

2) питання продуктивності і затримок, які є основним і постійним недоліком у роботі мікросервісів [4];

3) потреба у динамічному масштабуванні для підтримки продуктивності без перевитрат ресурсів;

4) питання моніторингу і безпеки є особливо складною задачею через необхідність аналізу великої кількості метрик, логів і трасування запитів.

Хоча сучасні дослідження демонструють значний прогрес у вдосконаленні архітектури мікросервісів, акцентуючи увагу на інтеграції нових інструментів, підходів і технологій, але носять розрізнену структуру і вирішують задачі окремих проблематик. Тому дане дослідження не тільки не втратило актуальності в останні роки, а й зросло, через те, що компанії активно переходять до мікросервісної архітектури, і змушені самотужки шукати рішення для мінімізації негативного ефекту від недоліків використання мікросервісної архутектури. Особливу роль відіграють концепції оркестрації, спостережуваності (observability), безпеки, автоматизації процесів, а також використання штучного інтелекту для підвищення ефективності управління мікросервісами.

Аналіз останніх досліджень і публікацій. Мікросервісна архітектура залишається фундаментальним підходом у побудові розподілених систем, і аналіз сучасних досліджень підкреслює кілька основних напрямків її розвитку. Зокрема, в центрі уваги перебувають автоматизація, оптимізація продуктивності, забезпечення спостережуваності, інтеграція штучного інтелекту, а також гібридні архітектури, що поєднують моноліти з мікросервісами.

Автоматизація є ключовою для стабільності та ефективності мікросервісів, і ця тема докладно розглядається у дослідженнях Браветі [5, с. 352], Фу [6, с. 416], Мустяла [7, с. 61], Муґерая і Девадкар [8, с. 2]. Ці роботи аналізують алгоритми динамічного розгортання та управління ресурсами, підкреслюючи роль Kubernetes у реалізації автоматизованого масштабування залежно від навантаження. Зокрема, в цих дослідженнях пропонують оптимальні стратегії автоматизації для зменшення витрат на інфраструктуру, тоді як деякі оцінюють продуктивність схем управління контейнерами на різних хмарних платформах, демонструючи, як адаптивні алгоритми підвищують ефективність навіть у середовищах із піковими навантаженнями. Мустяла фокусується на оптимізації розподілу завдань у Kubernetes, досліджуючи, як поєднання динамічного планування завдань і алгоритмів машинного навчання дозволяє досягати балансу між вартістю та продуктивністю.

Продуктивність і масштабованість розглядаються як основні виклики в сучасних мікросервісах. Гонзалес [9] і Ганг [10, с. 3] зосереджуються на модульному дизайні систем, що забезпечує стійкість до збоїв і підтримуваність навіть у великих масштабах. Один з них аналізує стратегічні підходи до побудови модульних архітектур, які дозволяють підвищити продуктивність завдяки оптимізації внутрішньої взаємодії сервісів. У свою чергу другий пропонує механізм забезпечення стійкості мікросервісів через різноманітність конфігурацій, що дозволяє зменшити вплив відмов та покращити безпеку.

Спостережуваність є важливим напрямком, і ця тема висвітлюється в роботах Себастьян [11, с. 3], Бельхірі [12, с. 27], Усман [13, с. 86906]. Себастьян [11, с. 4] підкреслює, як інструменти спостережуваності, такі як OpenTelemetry і Jaeger, дозволяють покращити кібербезпеку та операційну стійкість критичних систем. Водночас інші роботи акцентують увагу на важливості трасування та логування для виявлення вузьких місць у мікросервісах, пропонуючи адаптивні підходи до моніторингу залежно від поточного навантаження.

Інтеграція штучного інтелекту для підвищення ефективності мікросервісів є ще однією важливою тенденцією, яку розглядають у дослідженнях [12, с. 28; 13, с. 86908]. Ці дослідження демонструють використання AI для прогнозування навантажень і автоматизації управління ресурсами, що дозволяє адаптувати системи до змін у режимі реального часу. Наприклад, вивчаючи алгоритми динамічного розподілу завдань у хмарних середовищах, підкреслюють, що використання AI підвищує як продуктивність, так і стабільність систем.

Гібридні архітектури, що поєднують моноліти з мікросервісами, є ефективним способом поступової модернізації старих систем, і це розглядається в роботах [14, с. 72; 15], а саме побудови API Gateway для інтеграції старих і нових компонентів, тоді як Hang et al. фокусуються на розгортанні стійких архітектур, що поєднують стабільність монолітів із гнучкістю мікросервісів. Sebastião підкреслює роль таких підходів у забезпеченні стійкості критичних систем, особливо в контексті фінансових і банківських додатків. Таким чином, аналіз досліджень виявляє загальні тенденції розвитку мікросервісних технологій: автоматизація, адаптація до змін у навантаженні через AI, інтеграція нових підходів до моніторингу та поступовий перехід від монолітів до гібридних архітектур. Ці тенденції дозволяють забезпечувати високу продуктивність, стійкість і гнучкість сучасних систем.

Постановка завдання. Метою статті є дослідження сучасних технологій та підходів, які

дозволяють покращити продуктивність, масштабованість і надійність мікросервісних систем, а також оцінка нових рішень, що застосовуються на практиці.

Виклад основного матеріалу. Традиційно програмні додатки будувалися як монолітні системи, де всі компоненти інтегровані в єдину кодову базу. Цей підхід забезпечував простоту розробки та розгортання, але з часом виникали проблеми з масштабованістю та підтримкою. З розвитком технологій та зростанням вимог ринку архітектурні підходи еволюціонували, що призвело до появи мікросервісної архітектури.

Виділялись наступні ключові аспекти мікросервісної архітектури:

1) масштабованість: мікросервісна архітектура дозволяє масштабувати окремі сервіси незалежно один від одного, що забезпечує ефективне використання ресурсів та підвищує продуктивність системи;

2) децентралізація: кожен мікросервіс розробляється та розгортається окремо, що дозволяє командам працювати автономно та впроваджувати зміни без впливу на інші частини системи [16];

3) незалежність сервісів: мікросервіси можуть бути реалізовані з використанням різних технологій та мов програмування, що надає гнучкість у виборі оптимальних рішень для кожного конкретного завдання [17].

Типовими викликами мікросервісної архітектури є складність тестування, взаємодія сервісів і забезпечення продуктивності. Тестування мікросервісів значно ускладнюється порівняно з монолітними додатками через необхідність моделювати взаємодії між різними сервісами, що вимагає комплексного підходу та спеціальних інструментів. Взаємодія сервісів також є значним викликом, оскільки забезпечення ефективної та надійної комунікації між численними мікросервісами потребує ретельного планування і використання сучасних технологій для інтеграції. Ще однією складністю є забезпечення високої продуктивності, оскільки розподілена природа мікросервісів може призводити до затримок у передачі даних, що потребує оптимізації архітектури та застосування рішень для мінімізації цих затримок. Усі ці аспекти потребують глибокого аналізу і правильного підходу для досягнення стабільної та ефективної роботи системи.

Сучасні технології та підходи до розгортання мікросервісів зосереджуються на автоматизації, інтеграції штучного інтелекту та застосуванні алгоритмів оптимізації. Це дозволяє вирішувати складні задачі масштабування, продуктивності, та ефективності використання ресурсів.

Автоматизація тестування є одним із ключових аспектів забезпечення якості в мікросервісній архітектурі. Розподілена природа мікросервісів вимагає ефективних підходів до тестування, які включають TDD, BDD, DDD, інтеграцію з CI/CD-процесами, а також використання симуляцій для моделювання взаємодії між сервісами. TDD (Test-Driven Development – розробка через тестування) – це підхід, при якому тести пишуться перед реалізацією функціоналу. В мікросервісах це дозволяє гарантувати, що кожен сервіс працює незалежно й відповідає очікуванням.

Переваги TDD включають покращення якості коду, забезпечення стабільності та швидке виявлення помилок. Завдяки написанню тестів до реалізації функціоналу код стає більш структурованим і легко підтримуваним, що спрощує його подальшу модифікацію. Крім того, тести гарантують стабільність системи, оскільки нові зміни перевіряються на сумісність із існуючим функціоналом, мінімізуючи ризик виникнення помилок. Ще однією важливою перевагою TDD є можливість виявляти помилки на ранніх етапах розробки, що значно знижує витрати на їх виправлення і прискорює процес розробки.

BDD (Behavior-Driven Development – розробка через поведінку) – це підхід, що орієнтується на опис поведінки системи у вигляді тестів, написаних зрозумілою для бізнесу мовою. BDD забезпечує прозорість, адже тести зрозумілі всім членам команди, включно з бізнес-аналітиками. Методологія фокусується на бізнес-логіці, гарантує відповідність реалізації очікуваній поведінці системи та полегшує автоматизацію регресійного тестування, що дозволяє перевіряти вплив змін на існуючі сценарії.

Domain-Driven Design (DDD) — це підхід до розробки програмного забезпечення, орієнтований на бізнес-домен. Основна ідея DDD полягає у глибокому розумінні предметної області (домену) і тісній взаємодії між технічними спеціалістами та експертами предметної області.

DDD забезпечує високу відповідність бізнес-вимогам, оскільки розробка базується на реальних потребах, що мінімізує ризик створення нерелевантного функціоналу. Уніфікована мова сприяє прозорості та покращує комунікацію між розробниками й бізнесом. Розподіл системи на незалежні контексти полегшує їх підтримку й розвиток, а також забезпечує масштабованість, дозволяючи окремим контекстам розвиватися незалежно.

На рис. 1 показано перетин трьох підходів до розробки програмного забезпечення: ATDD (Acceptance Test-Driven Development – тестування, орієнтоване на узгодження критеріїв прийняття), BDD і DDD.

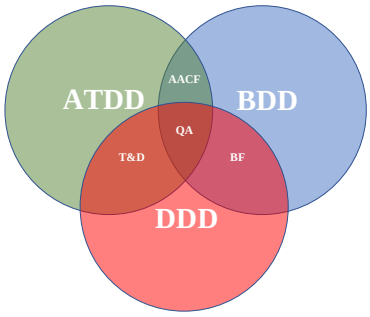

Рис. 1. Перетин трьох підходів до розробки програмного забезпечення

Внесені позначки:

– T&D – Testing & Design (тестування і дизайн);

– AACF – Automated Acceptance Criteria First (спочатку автоматизовані критерії прийняття);

– BF – Business Focus (фокус на бізнесі);

– QA – Quality Assurance (Забезпечення якості).

Рис. 1 показує, як ATDD, BDD і DDD доповнюють один одного, а їх об'єднаний підхід дозволяє створювати програмне забезпечення, орієнтоване на бізнес, протестоване і розроблене з урахуванням реальних потреб користувачів і домену.

Зони перетину:

– ATDD + BDD = AACF – об'єднує фокус на тестуванні з формулюванням зрозумілих бізнес-критеріїв;

– BDD + DDD = BF – узгоджує очікування бізнесу з технічним дизайном через опис поведінки, орієнтованої на домен;

– ATDD + DDD = T&D – забезпечує створення програмного забезпечення, яке відповідає тестам і точно відображає домен;

– ATDD + BDD + DDD = QA – зона, де всі три підходи об'єднуються, забезпечуючи високу якість розробки програмного забезпечення.

В той же час використання симуляцій дозволяє моделювати взаємодії між мікросервісами без потреби у запуску всіх залежностей. Це важливо для тестування розподілених систем.

Використання симуляцій забезпечує швидкість і зручність, дозволяючи заміняти запуск усіх сервісів «стабами» або «моками». Це допомагає виявляти проблеми у складних сценаріях, які важко відтворити в реальному середовищі, таких як відмови сервісів чи затримки в мережі. Крім того, симуляції економлять ресурси, оскільки споживають їх менше, ніж повноцінне середовище.

Серед основних інструментів для симуляцій: WireMock, який дозволяє створювати HTTP-запити та відповіді для тестування API; Mockito, що використовується для мокінгу залежностей у юніт-тестах; та Testcontainers, який забезпечує запуск тестів у контейнерах, імітуючи реальні умови.

Масштабованість і продуктивність є критичними аспектами мікросервісної архітектури, особливо в умовах зростаючих навантажень. Сучасні технології пропонують ефективні інструменти та підходи для підтримки цих аспектів. Горизонтальне масштабування (horizontal scaling) передбачає збільшення кількості екземплярів сервісу (подів) для обробки більшого навантаження. Kubernetes є провідним інструментом для оркестрації контейнерів і забезпечує автоматизацію цього процесу [18; 19, с. 2108; 20; 21; 22, с. 0185].

Сучасні підходи до масштабування та продуктивності мікросервісів зосереджуються на автоматизації та оптимізації процесів, зокрема через горизонтальне масштабування за допомогою Kubernetes, що автоматизує розширення ресурсів і забезпечує стабільність навіть при високих навантаженнях, використання gRPC для низьколатентної комунікації, ідеально підходить для систем з високими вимогами до швидкодії, а також кешування, яке знижує навантаження на базу даних і прискорює доступ до даних.

Автоматичне масштабування в Kubernetes використовує Horizontal Pod Autoscaler (HPA) для автоматичного збільшення або зменшення кількості подів залежно від метрик, таких як використання CPU чи пам'яті [22, с. 0186]. Наприклад, якщо трафік на API зростає, HPA запускає додаткові поди для обробки запитів. Рівномірний розподіл навантаження забезпечується через Kubernetes Load Balancer, який розподіляє запити між подами, гарантуванням рівномірного використання ресурсів. Масштабування кластера здійснюється за допомогою Cluster Autoscaler, який додає або видаляє вузли в кластері залежно від потреби у ресурсах. Серед його переваг: гнучкість у розподілі ресурсів, ефективне використання інфраструктури, автоматичне реагування на зміни у навантаженні. Обмеження включають потребу в ретельному налаштуванні HPA та Cluster Autoscaler для уникнення перевитрат ресурсів.

Завдяки використанню HTTP/2 і Protocol Buffers (Protobuf), gRPC забезпечує низькі затримки та компактну передачу даних [23, с. 122]. Переваги gRPC включають низьку латентність завдяки використанню HTTP/2, що дозволяє мультиплек-

сувати запити через одне з'єднання, зменшуючи затримки, високу продуктивність завдяки компактній серіалізації Protobuf, що знижує обсяг переданих даних, а також підтримку двостороннього стрімінгу, що дозволяє передавати дані в реальному часі між клієнтом і сервером.

Переваги gRPC включають високу пропускну здатність, швидку взаємодію між сервісами та підтримку багатьох мов програмування. Однак є й обмеження: складність налаштування у порівнянні з REST та менша підтримка у браузерах, оскільки для цього потрібен gRPC-Web.

Кешування для підвищення швидкості доступу до даних є одним із ключових методів оптимізації продуктивності. Такі інструменти, як Redis і Memcached, дозволяють зберігати часто використовувані дані у пам’яті для швидкого доступу [24, с. 64877].

Кешування забезпечує швидкий доступ до даних, оскільки читання з пам'яті значно швидше, ніж з дискових баз даних, зменшує навантаження на базу даних, що дозволяє уникнути вузьких місць, а також надає гнучкість, дозволяючи кешувати будь-які дані, від API-відповідей до результатів складних обчислень. Однак є й обмеження: потреба в управлінні TTL для уникнення застарілих даних та обмежений обсяг пам'яті для великих даних.

Спостережуваність є критично важливим аспектом сучасних мікросервісних систем. Здатність моніторити, аналізувати та реагувати на метрики, логи й трасування запитів є ключем до підтримки продуктивності, надійності та стійкості мікросервісів. Нові підходи до спостережуваності зосереджені на інтеграції даних, автоматизації процесів та зниженні впливу моніторингу на систему [25, с. 1981].

На рис. 2 представлено основні складові спостережуваності (Observability) у розподілених системах, що ілюструє концепцію, де спостережуваність формується з кількох типів даних, які забезпечують повну картину стану системи [26, с. 3197].

Три основні стовпи спостережуваності:

1. Logs (Логи): записи подій, які генеруються під час виконання системи. Логи допомагають виявляти непередбачувану поведінку та помилки у компонентах мікросервісів. Вони є найпоширенішим джерелом інформації для аналізу помилок і несправностей;

2. Metrics (метрики): числові значення, які описують стан або продуктивність системи (наприклад, використання CPU, час відповіді, обсяг пам'яті). Метрики дозволяють відстежувати зміни у реальному часі;

3. Traces (трасування): інформація про шлях виконання запиту через різні сервіси в системі. Трасування забезпечує прозорість взаємодії між мікросервісами, що допомагає виявляти вузькі місця.

Додаткові метрики включають досвід користувачів, що надає інформацію про взаємодію з системою та дозволяє оцінювати якість обслуговування, події, які фіксують важливі моменти часу, що сигналізують про зміни стану системи або її поведінки, залежності додатків, що описують, як компоненти взаємодіють між собою, і є критичними для аналізу відмов і оптимізації, а також логи розгортання, які містять дані про зміни конфігурації та процеси розгортання, що можуть спричинити збої або вплинути на продуктивність системи.

Для забезпечення спостережуваності в розподіленій системі важливо збирати дані не тільки з окремих додатків, але й із усієї IT-інфраструктури [27, с. 385]. Поєднання логів, метрик, трасувань та інших типів даних дозволяє операторам швидко визначати, де, що і коли сталося, а також передбачати потенційні проблеми.

Поєднання основних та додаткових метрик дозволяє створити повну картину про роботу

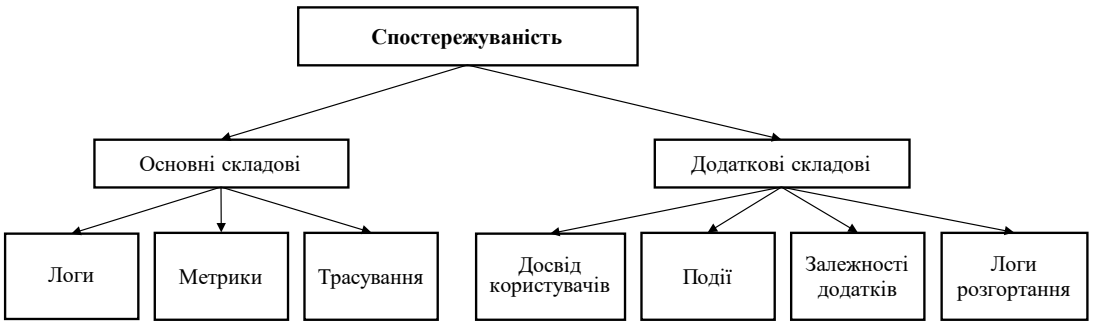

Рис. 2. Основні складові спостережуваності за системами на базі мікросервісної архітектури

системи, підвищити ефективність виявлення проблем і швидкість реагування. Це критично важливо для сучасних розподілених систем, особливо для мікросервісної архітектури. Adaptive Observability – це підхід до динамічного налаштування моніторингу залежно від поточного навантаження та стану системи [28]. Основні принципи полягають у збиранні лише необхідних даних у моменти високого навантаження для зниження впливу моніторингу на систему, а також в автоматичній адаптації рівня деталізації даних до поточного стану системи.

Переваги включають зниження витрат на зберігання та обробку даних, підвищення ефективності системи моніторингу та швидке реагування на зміни у навантаженні. Нові підходи до спостережуваності дозволяють уніфікувати процеси збору даних за допомогою OpenTelemetry, забезпечити прозорість взаємодії між сервісами через Jaeger, підвищити ефективність моніторингу ресурсів за допомогою Prometheus, збирати низькорівневу інформацію для глибокого аналізу через eBPF та динамічно адаптуватися до змін у системі з використанням Adaptive Observability.

Ці інструменти та методи є критично важливими для побудови стійких і продуктивних мікросервісних систем у реальних умовах. Зведена харктеристика інструментів наведена в табл. 1.

Поєднання описаних інструментів з підходами eBPF та Adaptive Observability дозволяє розробникам будувати ефективні системи спостережуваності. Ці методи знижують витрати, забезпечують високу продуктивність та швидке реагування на зміни, що робить їх критично важливими для сучасних мікросервісних архітектур.

Основними напрямами інтеграції штучного інтелекту в роботу з мікросервісами є прогнозування ресурсів за допомогою Predictive Analytics, оптимізація масштабування через Reinforcement Learning, використання Anomaly Detection для автоматичного коригування конфігурацій.

Прогнозна аналітика (Predictive Analytics) базується на використанні машинного навчання для аналізу історичних даних і прогнозування майбутніх потреб у ресурсах. Алгоритми аналізують метрики, такі як трафік, використання CPU/пам'яті або пікові навантаження. На основі отриманих прогнозів система може заздалегідь виділяти або звільняти ресурси, забезпечуючи безперебійність роботи. Так, наприклад, Netflix використовує Predictive Analytics для оптимізації розподілу серверів у різних регіонах, що дозволяє уникати перевантажень у пікові години [29].

Оптимізація масштабування через Reinforcement Learning (метод навчання системи через нагородження за успішні дії) використовує RL-агенти, які навчаються оптимізувати масштабування мікросервісів залежно від змін у навантаженні (наприклад, система може визначити, коли краще запускати нові екземпляри сервісу, щоб зменшити затримки або знизити вартість. Наприклад, Amazon Web Services застосовує RL для адаптивного масштабування своїх сервісів, що підвищує ефективність використання ресурсів [30].

Використання Anomaly Detection (виявлення аномалій) для автоматичного коригування конфігурацій дозволяє системі ідентифікувати та реагувати на відхилення від нормальної поведінки. Алгоритми машинного навчання аналізують метрики, наприклад, раптове збільшення затримок або споживання пам'яті. У разі виявлення аномалії система може автоматично перезапустити сервіс, масштабувати його або повідомити адміністратора. Так Google Kubernetes Engine (GKE) інтегрує аномалії для динамічної адаптації сервісів [31].

Використання Kubernetes з AI-driven orchestration інтегрує AI для автоматизації роз-

Таблиця 1

Особливості популярних інструментів спостережуваності

Інструмент	Основна функція	Особливості	Переваги	Обмеження
OpenTelemetry	Моніторинг, трасування та логування	Підтримує стандарти для збору телеметрії. Інтеграція з багатьма платформами.	Уніфікований підхід до спостережуваності. Гнучкість інтеграції з Prometheus, Jaeger, Grafana тощо.	Вимагає ручного налаштування інтеграцій.
Jaeger	Розподілене трасування	Відстежує запити між мікросервісами. Побудова залежностей між викликами.	Зручна візуалізація трасування. Швидке виявлення вузьких місць у системі.	Фокус лише на трасуванні.
Prometheus	Збір і зберігання метрик	Підтримує PromQL для аналізу даних. Інтеграція з Grafana для візуалізації.	Легка інтеграція. Ідеально підходить для моніторингу Kubernetes.	Обмеження у зборі логів та трасування.

поділу ресурсів і перенаправлення трафіку. Kubernetes автоматично адаптує кількість подів, перенаправляє трафік та розподіляє ресурси залежно від метрик. Інтеграція AI дозволяє передбачати зміни у навантаженні та заздалегідь оптимізувати конфігурацію кластера.

Інтеграція штучного інтелекту та алгоритмів оптимізації відкриває нові можливості для управління мікросервісами. Завдяки AI-driven orchestration, Predictive Analytics та автоматизації процесів CI/CD компанії можуть досягати високої продуктивності, ефективного масштабування та стабільності системи. Ці підходи стають стандартом для організацій, які прагнуть розробляти інноваційні та надійні хмарні сервіси.

Гібридна архітектура, яка поєднує моноліт і мікросервіси, стає популярним рішенням у ситуаціях, коли повний перехід до мікросервісної архітектури неможливий або недоцільний [32]. Гібридний підхід до архітектури дозволяє поєднувати переваги монолітів і мікросервісів, що стає у нагоді за таких кейсів. Поетапна міграція передбачає використання мікросервісів для нових функцій, залишаючи основний функціонал у стабільному моноліті.

Багато організацій починають із моноліту, але з часом, через зростання вимог до масштабованості, функціональності та швидкості оновлення, виникає потреба у переході до мікросервісів. Повна перебудова архітектури може бути дороговартісною, часозатратною і ризикованою через можливі збої. Інтеграція застарілих систем здійснюється через API Gateway, який об'єднує старі системи з сучасними мікросервісами.

Деякі системи, наприклад, основні транзакційні процеси у банках, дуже стабільні і не потребують частих змін, тому доцільно залишити цей функціонал у моноліті, а мікросервіси використовувати для нових функцій, таких як мобільні додатки чи аналітичні сервіси. Реалізація нових функцій вимагає гнучкості, і гібридні архітектури дозволяють поступово впроваджувати інновації без значних ризиків для існуючих систем. Старі системи (legacy) часто не підтримують прямий перехід до мікросервісів через архітектурні обмеження, тому одним із рішень є інтеграція через API Gateway, який виступає посередником між монолітом і мікросервісами. Банківські системи є яскравим прикладом використання гібридної архітектури через складність і критичність їхніх основних функцій.

Монолітні компоненти включають обробку транзакцій, управління рахунками та процеси авторизації та автентифікації. Мікросервіси, у свою чергу, охоплюють аналітику та звітність, мобільні додатки та системи управління лояльністю (cashback, бонуси).

Проте гібридна архітектура має низку обмежень. По-перше, складність інтеграції, оскільки забезпечення коректної взаємодії між моноліт ом і мікросервісами вимагає додаткових зусиль. По-друге, ускладнення моніторингу, оскільки необхідно контролювати як моноліт, так і мікросервіси, що збільшує навантаження на команди DevOps. Останнє обмеження — можливість технічного боргу, адже якщо перехід до мікросервісів затягується, моноліт може стати «пляшковим горлом» в роботі системи

Гібридна архітектура є ефективним рішенням для організацій, які прагнуть поступово інтегрувати мікросервіси у свої системи. Завдяки поєднанню моноліту для стабільних функцій та мікросервісів для інноваційних компонентів, цей підхід дозволяє оптимізувати процеси, мінімізуючи ризики. Банківські системи та компанії, що працюють із застарілими системами, активно використовують цей підхід для збереження стабільності та швидкого впровадження нових технологій.

Висновки. Мікросервісна архітектура є одним із ключових напрямків сучасної розробки програмного забезпечення завдяки її здатності забезпечувати гнучкість, масштабованість і надійність систем. Однак разом із перевагами вона має низку викликів, таких як складність інтеграції, необхідність у динамічному масштабуванні, продуктивність, забезпечення спостережуваності та безпека. Проведене дослідження дозволяє виявити сучасні технології, які допомагають вирішувати ці проблеми.

Серед основних напрямів удосконалення роботи з мікросервісами слід відзначити автоматизацію використання CI/CD-пайплайнів, динамічне управління ресурсами через Kubernetes, використання автоматизованих рішень для розгортання, що дозволяє знижувати складність розробки та пришвидшувати адаптацію систем до змін. Це особливо важливо в умовах високих навантажень, коли потрібна швидка реакція на зростання трафіку.

За останні роки реалізовано низку рішень в напрямі оптимізації продуктивності в мікросервісах, яка значною мірою залежить від застосування горизонтального масштабування, gRPC для швидкої комунікації та кешування за допомогою Redis або Memcached. Ці інструменти

не тільки підвищують швидкодію системи, але й дозволяють уникати перевитрат ресурсів.

Можливість спостережуваності також стала важливою умовою ефективної роботи розподілених систем. Такі інструменти, як OpenTelemetry, Prometheus і Jaeger, дозволяють виявляти проблеми на ранніх етапах, аналізувати метрики та трасування, а також забезпечувати прозорість у роботі мікросервісів. У цьому контексті адаптивний моніторинг стає особливо актуальним, дозволяючи зменшити вплив моніторингу на систему під час пікових навантажень.

Практично всі покращення в роботі з мікросервісами використовують штучний інтелект. Так Predictive Analytics, Reinforcement Learning і Anomaly Detection допомагають прогнозувати майбутні навантаження, адаптувати ресурси до змін та виявляти аномалії у роботі систем. Ці технології вже демонструють свою ефективність у реальних кейсах, таких як Netflix та Amazon Web Services.

Проведене дослідження показало, що сучасні технології та підходи значно розширюють можливості мікросервісної архітектури, підвищуючи її ефективність, стабільність та адаптивність, а це є основним чинником у виборі мікросервісної архітектури для створення систем, що відповідають сучасним вимогам бізнесу.

Список літератури:

1. Garimilla M. Microservices architecture: revolutionizing modern software development. *International Journal of Innovative Research.* 2024. Vol. 13. P. 16637-16645.

2. Dragoni N, Giallorenzo S., Lluch L., Mazzara M., Montesi F., Mustafin R., Safina L. Microservices: Yesterday, Today, and Tomorrow. *In Present and Ulterior Software Engineering*. 2017. P. 195-216. URL: https://backend.orbit.dtu.dk/ws/portalfiles/portal/140638498/1606.04036.pdf .

3. Про мікросервісну архітектуру. URL: https://foxminded.ua/mikroservisna-arkhitektura (Дата звернення: 26.10.2024).

4. Scale Microservices Performance with Distributed Caching. URL: https://www.alachisoft.com/blogs/scale-microservices-performance-with-distributed-caching/ (Дата звернення: 26.10.2024).

5. Bravetti M., Giallorenzo S., Mauro J., Talevi I., Zavattaro G. Optimal and automated deployment for microservices, Fundamental Approaches to Software Engineering. 2019. Vol. 11424. P. 351-368. DOI: https://doi.org/10.1007/978-3-030-16722-6_21.

6. Fu Y., Gu S., Cheng L., Liu L. Performance evaluation of resource management schemes for cloud-native platforms with computing containers, *IEEE International Performance, Computing, and Communications Conference (IPCCC).* 2022. P. 414-415. DOI: https://doi.org/10.1109/ipccc55026.2022.9894300.

7. Mustyala A. Dynamic resource allocation in Kubernetes: Optimizing cost and performance, *EPH - International Journal of Science and Engineering*. 2021. Vol. 7. no. 3. P. 59-71. DOI: https://doi.org/10.53555/ephijse.v7i3.237.

8. Mugeraya S., Devadkar K. Dynamic task scheduling and resource allocation for microservices in cloud, *Journal of Physics: Conference Series.* 2022. Vol. 2325. P. 1-10. DOI: https://doi.org/10.1088/1742-6596/2325/1/012052

9. González S. Modular software design in distributed systems: Strategic approaches for building scalable, maintainable, and fault-tolerant architectures in modern microservice environments, *Eigenpub Review of Science and Technology*. 2023. Vol. 7. no. 1. DOI: https://doi.org/10.1007/s10916-020-1195-x.

10. Hang Y., Xiulei W., Changyou X., Bo X. A Microservice Resilience Deployment Mechanism Based on Diversity, *Security and Communication Networks*. 2022. P. 1-13. DOI: https://doi.org/10.1155/2022/7146716.

11. Sebastião F. P. The role of a microservice architecture on cybersecurity and operational resilience in critical systems, *Master's Thesis, University of Porto*. 2023. Vol. 2325. P. 1-10. DOI: https://doi.org/10.1088/1742-6596/2325/1/012052.

12. Belkhiri A., Bushehri A.S., Magalhaes F., Nicolescu G. Transparent Trace Annotation for Performance Debugging in Microservice-oriented Systems, *ICPE '23 Companion: Companion of the 2023 ACM/SPEC International Conference on Performance Engineering*. 2023. P. 25-32. DOI: https://doi.org/10.1145/3578245.3585030.

13. Usman M., Ferlin S., Brunstrom A., Taheri J., "A Survey on Observability of Distributed Edge & Container-Based Microservices," *in IEEE Access*. 2023. Vol. 10, P. 86904-86919, DOI: https://doi.org/10.1109/ACCESS.2022.3193102.

14. Mejía P. Best practices for microservice framework design, Advances in Intelligent Information Systems, *Journal of Intelligent Connectivity and Emerging Technologies*. 2022. Vol. 8. no. 4. P. 71-85. URL: https://questsquare.org/index.php/JOUNALICET/article/view/77/83.

15. Архітектура мікросервісів: Особливості, переваги, реальні приклади. URL: https://www.hostzealot.com.ua/blog/about-solutions/arxitektura-mikroservisiv-osoblivosti-perevagi-realni-prikladi (Дата звернення: 27.10.2024).

16. Порівняння ефективності мікросервісів та монолітних архітектур URL: https://peerdh.com/uk/blogs/programming-insights/comparing-the-efficiency-of-microservices-and-monolithic-architectures (Дата звернення: 28.10.2024).

17. Pedersen S.F. Enhancing microservices architecture with BDD, ATDD, and DDD in the IT industry (BAT3D). 2023. URL: https://medium.com/destinationaarhus-techblog/enhancing-microservices-architecture-with-bdd-atdd-and-ddd-in-the-financial-industry-7f97d5c3c8d0 (Дата звернення: 28.10.2023).

18. Vivek B.R. Performance impact of microservices architecture. The review of contemporary scientific and academic studies, *An International Multidisciplinary Online Journal*. 2023. Vol. 3. no. 6. DOI: https://doi.org/10.55454/rcsas.3.06.2023.010.

19. Oyeniran O., Adewusi A., Adeleke A., Akwawa L., Azubuko C. Microservices architecture in cloud-native applications: Design patterns and scalability, *Computer Science & IT Research Journal*. 2024. Vol. 5. no. 9, P. 2107-2124. DOI: https://doi.org/10.51594/csitrj.v5i9.1554.

20. Aksakalli I. K., Çelik T., Can A. B., Tekinerdoğan, B. Deployment andcommunication patterns in microservice architectures: A systematic literaturereview, Journal of Systems and Software. 2021. Vol. 180, P. 111014. DOI: https://doi.org/10.1016/j.jss.2021.111014

21. The Evolution of Data Centers: From Humble Beginnings to Technological Powerhouses. URL: https://www.linkedin.com/pulse/evolution-data-centers-from-humble-beginnings-akhil-sirasao-zrh6f/ (Дата звернення: 02.11.2024).

22. Shah J., Dubaria D. "Building modern clouds: using docker, kubernetes & google cloud platform," *IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, 2019, P. 0184-0189. DOI: https://doi.org/10.1109/CCWC.2019.8666479

23. Cinque M., Della Corte R., Pecchia A., "Advancing Monitoring in Microservices Systems," *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Berlin, Germany, 2019, P. 122-123, DOI: https://doi.org/10.1109/ISSREW.2019.00060.

24. Laghari A. A., He H., Khan A., Kumar N., Kharel R. Quality of experience framework for cloud computing (QoC), IEEE Access. 2018. Vol. 6, P. 64876–64890. DOI: https://doi.org/10.1109/ACCESS.2018.2865967

25. Sharma B., D. Nadig, "eBPF-Enhanced complete observability solution for cloud-native microservices," *ICC 2024 - IEEE International Conference on Communications*, Denver, CO, USA, 2024, P. 1980-1985. DOI: https://doi.org/10.1109/ICC51166.2024.10622329.

26. Monteiro D., Yu Y., Zisman A., Nuseibeh B. "Adaptive observability for forensic-ready microservice systems," *In IEEE Transactions on Services Computing*. 2023. Vol. 16, no. 5, P. 3196-3209. DOI: https://doi.org/10.1109/TSC.2023.3290474.

27. Koneni V. Microservices architectures using spring boot: embracing containerization and observability, *International Innovative Research Journal of Engineering and Technology*. 2024. Vol. 11. no. 3, P. 384-396.

28. Google Kubernetes Engine: Qwik Start. URL: https://www.cloudskillsboost.google/focuses/878?locale=uk&parent=catalog. (Дата звернення: 03.11.2024)

29. Gokul M. How Netflix Uses Artificial Intelligence. URL: https://www.argoid.ai/blog/netflix-ai (Дата звернення: 03.11.2024)

30. AWS Auto Scaling. URL: https://aws.amazon.com/autoscaling/ (Дата звернення: 03.11.2024)

31. T. Twain. Kubernetes & its Role in AI: Orchestrating End-to-End AI Pipelines. 2024. URL: https://amazic.com/kubernetes-its-role-in-ai-orchestrating-end-to-end-ai-pipelines/ (Дата звернення: 03.11.2024).

32. M. Puhl. Patterns for Microservices Architecture in Couchbase. 2021. URL: https://www.couchbase.com/blog/microservices-architecture-in-couchbase/ (Дата звернення: 21.07.2021).

Kozub V.V. PRACTICE OF USING NEW TECHNOLOGIES IN MICROSERVICE ARCHITECTURE OF SOFTWARE DEVELOPMENT

This article presents an analysis of the latest technologies in microservices architecture, which has long been a key concept in developing flexible, scalable, and reliable software systems. The microservices approach allows applications to be broken down into independent components that interact via clearly defined interfaces, ensuring modularity and scalability. The research methods include a literature review, analysis of existing technologies and approaches in the microservices domain, as well as examining practical examples of new tools used to address challenges in integration, scaling, and security. Specifically, the article focuses on new automation tools for deployment and testing processes, technologies for ensuring observability, performance optimization methods, and the integration of artificial intelligence for load forecasting and automated scaling. The main results of the study include a structured overview of key areas where microservices architecture is being enhanced, including: automation of pipelines for testing and deployment, use of technologies for caching and accelerating data access, integration of OpenTelemetry and Prometheus for monitoring, as well as AI integration for load analysis and forecasting. The research also highlights the significance of hybrid architectures that allow gradual integration of new technologies without disrupting the operation of legacy systems. The scientific novelty of the work lies in the comprehensive analysis of the latest technologies and their impact on the effectiveness of microservices architecture, as well as identifying promising directions for further research aimed at addressing existing challenges in integration, security, and scalability. An important contribution is the study of the impact of artificial intelligence and automation on improving microservices management efficiency. In the conclusions, it is noted that to further improve microservices architecture, newer tools for automation, machine learning, and artificial intelligence need to be more actively integrated, which will ensure efficient resource management, reduced latency, and enhanced security. Future research should focus on solving the integration challenges between old and new systems, as well as improving orchestration and observability processes.

Key words: *microservices architecture, deployment diversity, load balancing, system resilience, resource optimization.*