DOI 10.34185/1562-9945-5-162-2026-10
УДК 004.4:004.738.5

В.В. Спірінцев, О.В. Спірінцева, Є.С. Дубіна

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ РЕАЛІЗАЦІЇ МІКРОСЕРВІСНОЇ СИСТЕМИ МОНІТОРИНГУ КОРИСТУВАЧІВ ТА РЕСУРСІВ ХМАРНОЇ ПЛАТФОРМИ AWS

Анотація. Розроблено прототип мікросервісної системи моніторингу користувачів і ресурсів хмарної платформи AWS засобами мов програмування C# і Go. Виконано експериментальне тестування продуктивності мікросервісів на екземплярах AWS EC2, проведено аналіз використання CPU, оперативної пам'яті та часу відповіді сервісів, а також розроблено методику порівняльної оцінки ефективності реалізацій, що визначає закономірності зміни продуктивності під різними рівнями навантаження. Отримані результати визначають доцільність застосування зазначених мов програмування (C# і Go) в системах електронної комерції з хмарною архітектурою, що розширює уявлення про вибір технологій для мікросервісних систем моніторингу.
Ключові слова: мікросервіси, моніторинг, AWS, EC2, Go, C#, CPU, ефективність, ресурси.

Постановка проблеми. Розвиток електронної комерції вимагає працювати з великими обсягами даних, значною кількістю користувачів та динамічними змінами навантаження. В таких умовах важливо підтримувати стабільну роботу компонентів цифрової платформи, зокрема контролювати функціонування ресурсів. Збільшення масштабів застосунків та використання хмарних середовищ призвело до поширення мікросервісної архітектури [1], яка розподіляє логіку системи між незалежними компонентами. Це забезпечує масштабованість і можливість оновлення окремих сервісів без зупинки всієї системи. Аналіз властивостей і якості реалізації мікросервісної системи дає можливість визначити та забезпечити стабільність функціонування платформи. Актуальність дослідження обумовлена необхідністю підвищення ефективності моніторингу мікросервісних систем у хмарних середовищах електронної комерції, зокрема на платформі AWS, через порівняння реалізацій на різних мовах програмування та визначення доцільності їх використання у високонавантажених системах, зокрема забезпечення стабільності та продуктивності компонентів системи, оптимізації використання обчислювальних і мережевих ресурсів, підвищенні швидкодії обробки запитів користувачів та ефективності управління ресурсами в умовах динамічного навантаження.

Аналіз останніх досліджень і публікацій. Офіційні публікації AWS [1] надають комплексне технічне обґрунтування спостережуваності у мікросервісних системах. Документація AWS описує, як сервіс Amazon CloudWatch забезпечує збирання, обробку й

аналіз метрик, журналів та подій для компонентів системи з метою повного контролю продуктивності й стану AWS-ресурсів та додатків. До ключових елементів належать: *метрики використання CPU, пам'яті, затримок, логи, централізовані аналізи*, а також налаштовані *алерти* для автоматичного реагування на зміни стану системи. Такий підхід дозволяє виявляти аномалії й оперативно оптимізувати роботу сервісів у хмарному середовищі AWS. В роботі [2] було запропоновано модельно-орієнтований підхід для безперервного інжинірингу продуктивності у мікросервісних системах шляхом зв'язку між архітектурою системи та даними моніторингу. В роботі [3] було здійснено систематичний аналіз інструментів DevOps і моніторингу мікросервісів. В роботі [4] було оцінено *вплив хмарної мікросервісної архітектури на продуктивність додатків,* розглянуто метрики продуктивності (час відгуку, пропускну здатність, масштабованість і надійність) для порівняння cloud-native архітектур із монолітними. В роботі [5] розглянуто *уніфікований моніторинг мікросервісів із застосуванням Prometheus і Grafana*, як масштабованого рішення для збору метрик, конфігурації алармів і візуалізації даних продуктивності, що доповнює моніторинг AWS CloudWatch та підкреслює *важливість розширеного моніторингу для складних розподілених систем. В роботі [6] здійснено огляд методів та оцінено сучасні техніки для оперативного виявлення аномалій і аналізу причин збоїв у сервіс-орієнтованих та мікросервісних хмарних додатках, що є ключовою частиною моніторингу та забезпечення надійності. Але можливості подальшого вдосконалення та* розвитку даного напрямку досліджень ще не вичерпані та потребують розгляду.

Мета дослідження полягає у підвищення ефективності мікросервісного моніторингу користувачів і ресурсів хмарної платформи AWS шляхом порівняння його реалізацій засобами мов програмування C# і Go та аналізу отриманих результатів для визначення доцільності застосування кожної мови у високонавантажених системах електронної комерції.

Основна частина. Для прототипу мікросервісної системи було розгорнуто шість екземплярів Amazon EC2 (рисунок 1) на операційній системі Linux Ubuntu. П'ять вузлів містять по два мікросервіси кожен, окрім сервісу автентифікації, який розгорнуто окремо. Один екземпляр виділено під базу даних MySQL. Усі екземпляри мають тип екземпляру t3.micro, що має два логічні процесорні ядра, та 1 ГБ оперативної пам'яті. Всі вузли розташовані в одній VPC з виділеною підмережею для тестового прототипу, забезпечуючи мережеву ізоляцію та взаємодію між сервісами. Для забезпечення мережевої доступності та взаємодії мікросервісів налаштовані правила груп доступ, що дозволяють вхідний трафік на такі порти: TCP 80 використовується для HTTP-запитів до інтерфейсів сервісів; TCP 443 застосовується для HTTPS-з'єднань, та забезпечує захищену передачу даних між користувачами та сервісами, включаючи авторизацію, обробку платежів, управління сесіями та передачу метрик; TCP 22 призначений для підключення до екземплярів через SSH. Використовується для керування вузлами, оновлення системи, розгортання сервісів та діагностики роботи прототипу; TCP 3306 відкритий

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

для підключення до бази даних MySQL. Вузли з мікросервісами можуть виконувати запити до бази даних для створення та отримання інформації.

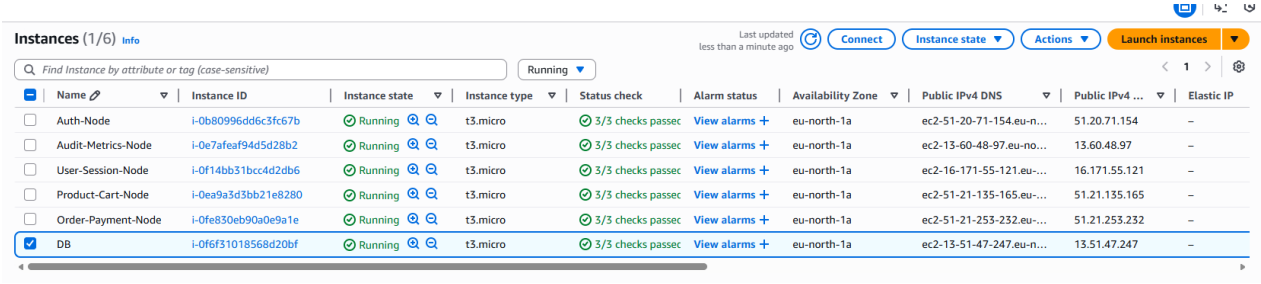

Рисунок 1 – Представлення розгорнутих екземплярів в консолі AWS

Набір вхідних точок охоплює повний життєвий цикл взаємодії користувача з системою та обробки бізнес-процесів. Через HTTP-запити реалізовано перевірку прав доступу, керування сесіями та контроль активності. Окремі групи вхідних точок забезпечують отримання, зміну та керування даними. Відповіді на запити повертаються в форматі JSON. Таким чином, структура відображає модульний підхід до побудови системи, де кожна група запитів реалізує окрему частину логіки, а взаємодія між ними здійснюється через стандартизовані HTTP-інтерфейси.

Для роботи з системою та її модулями розроблено інтерфейс, який забезпечує взаємодію користувача з мікросервісами та статистичними даними (рисунок 2).

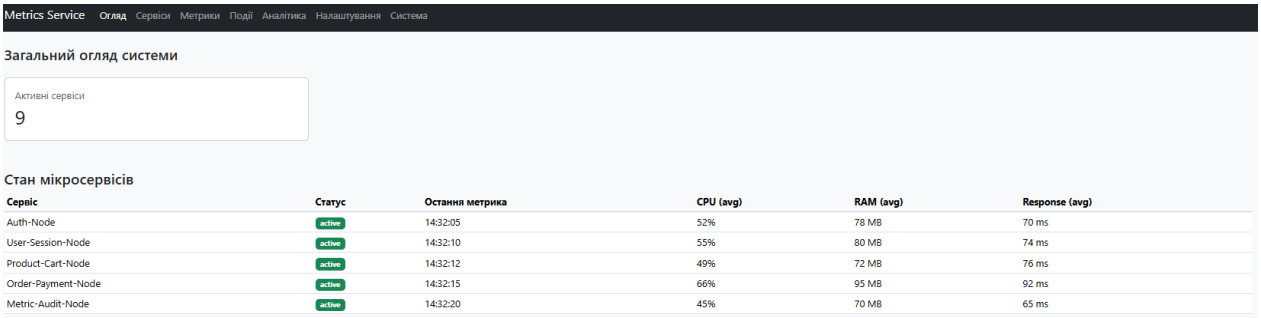

Рисунок 2 – Інформаційна дошка сервісу метрик

Блок загального огляду системи відображає середній час відповіді, завантаження CPU і пам’яті конкретних мікросервісів. Зокрема, реалізовано пошук і фільтри для відбору мікросервісів та їх метрик за часом. Деталізовані метрики представляють інформацію про роботу окремих мікросервісів у системі та їх ресурсоспоживання за певний період. Для кожного сервісу фіксуються такі параметри: тип операції, використання CPU, обсяг пам’яті, час відповіді та ідентифікатор запиту. Метрики збираються окремо для реалізацій на Go і C#, що дозволяє порівнювати продуктивність і споживання ресурсів між мовами. Графічне відображення метрик формується на основі агрегованих значень, для відстеження трендів завантаження CPU, використання RAM і часу відповіді сервісів у реальному часі. Таблиці подій показують часову послідовність дій, ресурсне навантаження та унікальні ідентфікатори для кожної операції.

На рисунку 3 наведено порівняння того, як мікросервіси на основні поточних даних за обраним часом споживають ресурси системи.

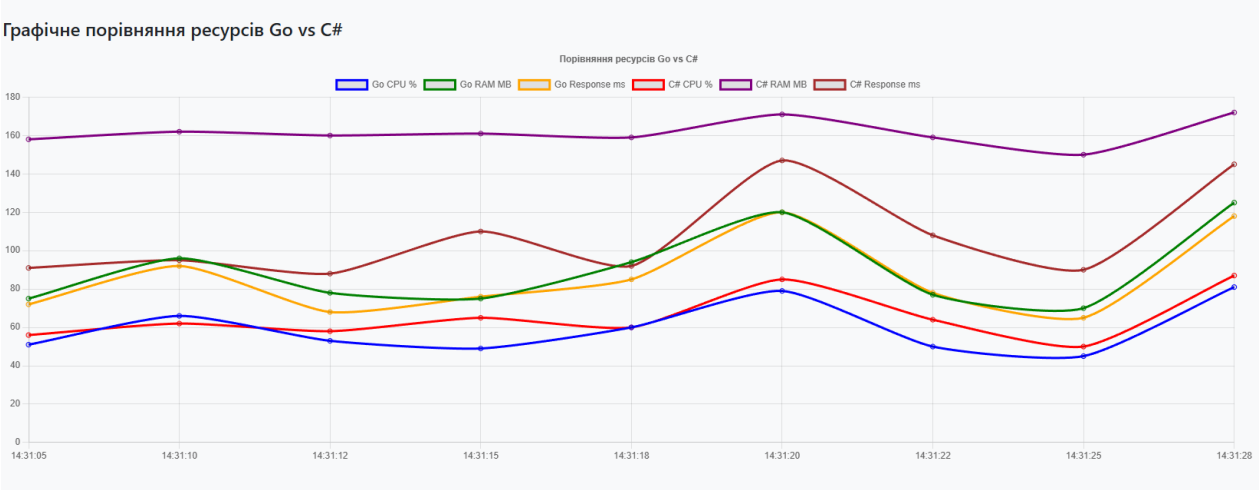

Рисунок 3 – Графічне порівняння ресурсів C# і Go за часом

Окремо розроблено інтерфейс для моніторингу ефективності подій мікросервісів. Графік відображає ефективність роботи сервісу автентифікації користувачів за обраний час (рисунок 4), та демонструє, як змінюються ключові показники ресурсоємності сервісу під час виконання операцій. Додатково на сторінці наведено числові показники: загальна кількість зареєстрованих користувачів, кількість активних користувачів та кількість активних сесій. Також надаються дані, що відображають хронологію подій які відбувалися у сервісі автентифікації та управління сесіями протягом часу.

Дані сервісу користувачів відображають список зареєстрованих користувачів із зазначенням їхнього статусу, дати реєстрації та часу останнього входу.

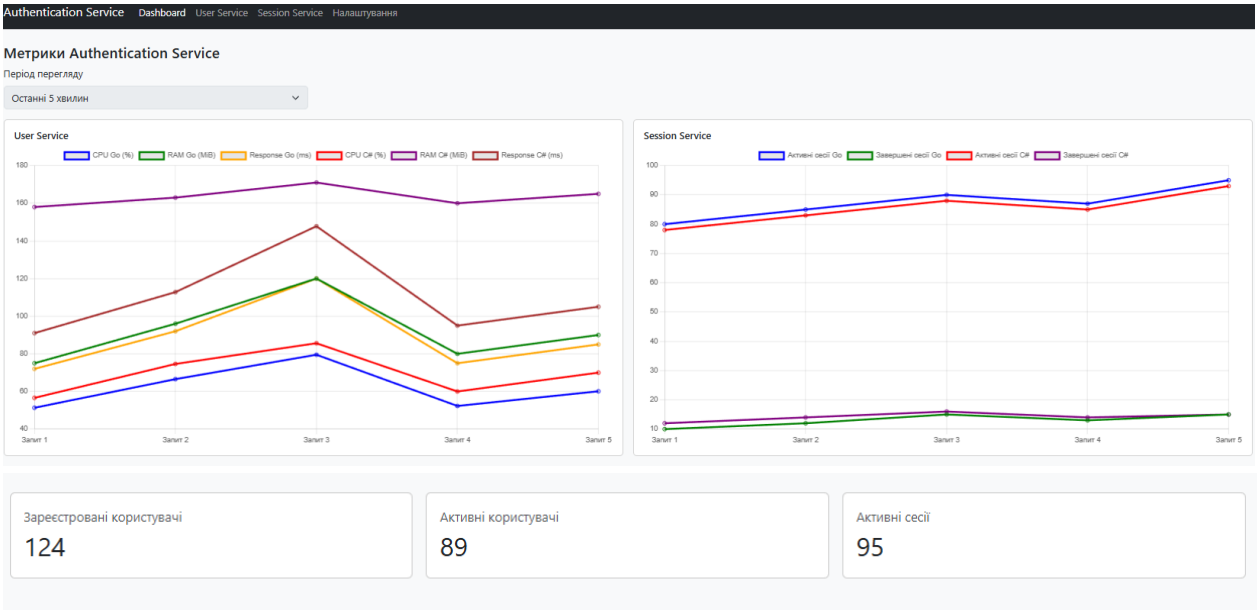

Рисунок 4 – Метрики сервісу автентифікації

Для порівняння ефективності роботи мікросервісів був розроблений скрипт на мові PHP, який моделює поведінку користувача під час взаємодії з сервісами та збирає статистику відповідей. Скрипт здійснює серію HTTP-запитів до сервісу метрик, повто-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

рюючи операції у пакетах визначеного розміру, що дозволяє імітувати одночасну активність кількох користувачів. Використання пакетної обробки запитів забезпечує реалістичне навантаження на сервіс, адже запити виконуються паралельно за допомогою механізму curl_multi, а результати кожного запиту зберігаються для подальшого аналізу. Для запобігання надмірного перевантаження між пакетами передбачена пауза, що імітує затримку між діями. Скрипт фіксує відповіді сервісу або помилки запитів, що дозволяє оцінити стабільність і швидкодію мікросервісу. Експеримент навантаження проводився шляхом послідовного збільшення кількості одночасних запитів до сервісу з метою оцінки його продуктивності та використання ресурсів (табл. 1). Графічне представлення графіку використання ресурсів під час запитів в системі зображено на рисунок 5. Графік демонструє зміну використання ресурсів мікросервісу метрик протягом обраного інтервалу 23 секунди для реалізацій на Go та C#. Для Go показники CPU коливаються від 45 % до 81 %, використання оперативної пам’яті – від 70 до 125 MiB, а час відповіді варіюється від 65 до 120 мс. Для C# CPU змінюється від 50 % до 87 %, оперативна пам’ять – від 150 до 172 MiB, а час відповіді – від 88 до 147 мс. Дані демонструють більш високе споживання ресурсів і повільніший час відповіді порівняно з Go, що вказує на більший вплив одночасних запитів на продуктивність сервісу.

Таблиця 1

Витрати ресурсів мікросервісу метрик під навантаженням

Кількість паралельних запитів	100	200	300
Реалізація з використанням GO, середні значення			
CPU, %	51,34	66,56	79,52
RAM Usage, МБ	75	96	126
Response Time, мс	72	92	118
Реалізація з використанням C#, середні значення			
CPU, %	56,6	74,63	85,63
RAM Usage, МБ	158	163	171
Response Time, мс	91	122,8	147,8

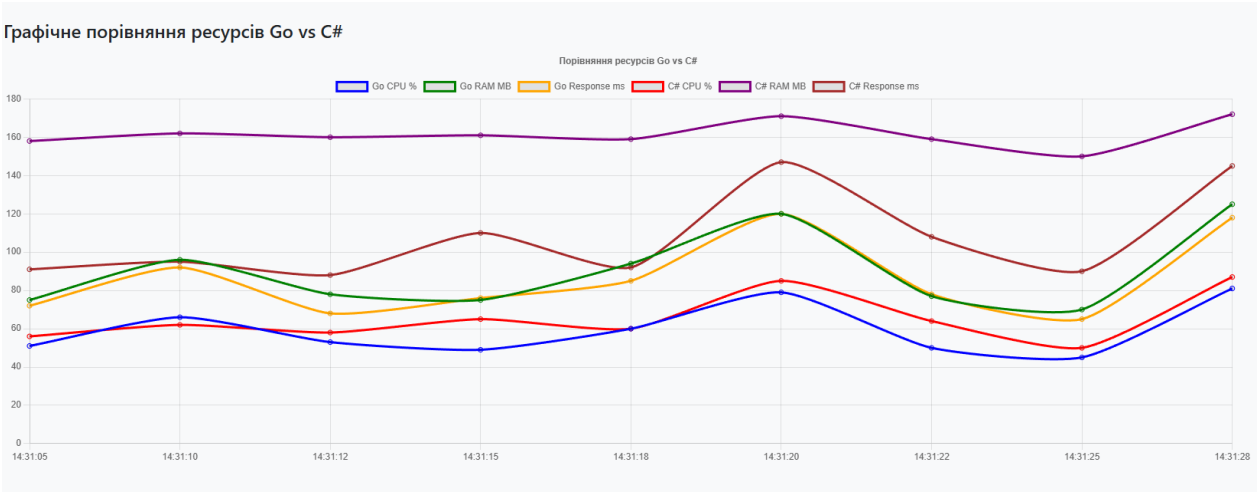

Рисунок 5 – Графік порівняння використання ресурсів для C# та Go

Для визначення поведінки застосунку під тривалим навантаженням було виконано тестування з помірним навантаженням протягом 6 хвилин (рисунок 6). Отримані метрики демонструють поведінку навантаження на процесор для обох реалізацій. Аналіз графіка CPU Utilization свідчить, що за однакового помірного навантаження значення використання процесора для реалізації на C# є вищими, ніж для реалізації на Go. Це відображено у лівій частині графіка, де зафіксовано тестування C#, у порівнянні з правою частиною, що відповідає навантаженню сервісу на Go. За метриками мережевого трафіку та кількості пакетів не спостерігається суттєвих відмінностей, які могли б пояснити зростання процесорного навантаження. За відсутності ознак обмежень з боку інфраструктури, отримані результати вказують, що підвищене значення CPU Utilization у реалізації на C# пов'язане переважно з особливостями виконання коду та внутрішньою моделлю обробки запитів.

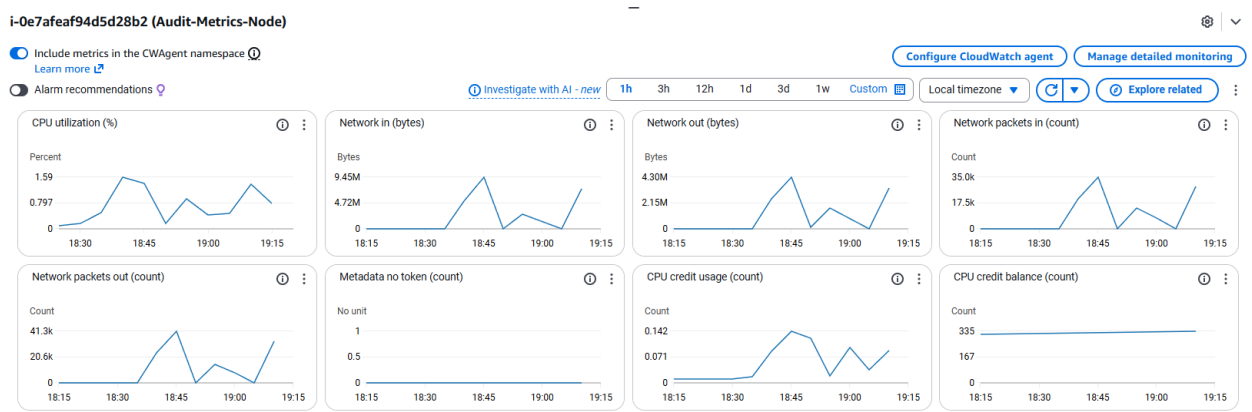

Рисунок 6 – Моніторинг ресурсів екземпляру EC2 мікросервіса метрик

Висновки. Реалізовано та досліджено мікросервісну систему моніторингу користувачів і ресурсів хмарної платформи AWS, а також проведено порівняльний аналіз двох реалізацій сервісу метрик на мовах C# та Go. Для відтворення навантаження, наближеного до реальної поведінки користувачів, розроблено скрипт PHP, який забезпечує паралельне виконання HTTP-запитів і збір статистики відповідей сервісу. У межах експерименту виконано серію навантажувальних тестів із поетапним збільшенням кі-

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

лькості одночасних запитів. Отримані результати дозволили проаналізувати використання процесорних ресурсів, оперативної пам'яті та час відповіді сервісу за різних рівнів навантаження. Порівняння середніх значень показало стабільну різницю між реалізаціями, що підтверджується як табличними даними, так і графічними результатами моніторингу в середовищі AWS EC2. Додатковий аналіз метрик тривалого помірного навантаження засвідчив, що за відсутності суттєвих відмінностей у мережевому трафіку та кількості пакетів підвищене використання CPU в реалізації на C# зумовлене особливостями виконання коду та моделі обробки запитів. Таким чином, проведене дослідження підтвердило, що реалізація мікросервісної системи моніторингу на Go за однакових умов характеризується меншим використанням CPU та оперативної пам'яті і меншим часом відповіді порівняно з реалізацією на C#, що свідчить про доцільність її використання при побудові мікросервісних систем моніторингу в хмарних середовищах.

ЛІТЕРАТУРА

1. Implementing Microservices on AWS. *Monitoring Microservices on AWS,* AWS Whitepaper, 2023. [Online]. Available:
https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/monitoring.html
2. A Model-driven Approach for Continuous Performance Engineering in Microservice-based Systems. URL: https://arxiv.org/pdf/2302.09999
3. Monitoring tools for DevOps and microservices: A systematic grey literature review. URL: https://doi.org/10.1016/j.jss.2023.111906
4. L. Desina, "Evaluating the Impact of Cloud-Native Microservices Architecture on Application Performance," *arXiv preprint arXiv:2305.15438,* 2023.
5. Jani, "Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions," *ResearchGate,* 2024. DOI:10.51219/JAIMLD/yash-jani/206
6. J. Soldani and A. Brogi, "Anomaly Detection and Root Cause Analysis in Cloud and Microservice-Based Applications: A Survey," DOI:10.48550/arXiv.2105.12378

REFERENCES

1. Implementing Microservices on AWS. Monitoring Microservices on AWS, AWS Whitepaper, 2023. [Online]. Available:
https://docs.aws.amazon.com/whitepapers/latest/microservices-on-aws/monitoring.html
2. A Model-driven Approach for Continuous Performance Engineering in Microservice-based Systems. URL: https://arxiv.org/pdf/2302.09999
3. Monitoring tools for DevOps and microservices: A systematic grey literature review. URL: https://doi.org/10.1016/j.jss.2023.111906
4. L. Desina, "Evaluating the Impact of Cloud-Native Microservices Architecture on Application Performance," arXiv preprint arXiv:2305.15438, 2023.
5. Jani, "Unified Monitoring for Microservices: Implementing Prometheus and Grafana for Scalable Solutions," ResearchGate, 2024. DOI:10.51219/JAIMLD/yash-jani/206
6. J. Soldani and A. Brogi, "Anomaly Detection and Root Cause Analysis in Cloud and Microservice-Based Applications: A Survey," DOI:10.48550/arXiv.2105.12378

Received 05.01.2026.
Accepted 09.01.2026.

Research on the efficiency of implementing a microservice system for monitoring users and resources on the AWS cloud platform

The relevance of this study is driven by the need to improve the effectiveness of monitoring microservice-based systems in cloud environments for e-commerce, particularly on the AWS platform. This is achieved through a comparative analysis of implementations developed in different programming languages and an assessment of their suitability for use in high-load systems. Special attention is given to ensuring the stability and performance of system components, optimizing the utilization of computing and network resources, increasing the speed of user request processing, and improving the efficiency of resource management under dynamically changing load conditions. The paper considers key aspects of studying the effectiveness of the microservice system for monitoring users and resources of the AWS cloud platform, implemented using C# and Go. The results of performance and resource utilization between implementations in the C# and Go programming languages were analyzed to determine the feasibility of using each language in e-commerce systems with a cloud architecture. To reproduce a load close to real user behavior, a PHP script was developed. It provides parallel execution of HTTP requests and collection of service response statistics. A series of load tests were performed with a gradual increase in the number of simultaneous requests. The results obtained allow to analyze the use of processor resources, RAM, and service response time at different load levels. Comparison of average values showed a stable difference between implementations, which is confirmed by both tabular data and graphical monitoring results in the AWS EC2 environment. Additional analysis of long-term moderate load metrics showed that in the absence of significant differences in network traffic and the number of packets, the increased CPU utilization in the C# implementation is due to the peculiarities of code execution and the request processing model. Thus, the conducted study confirmed that the implementation of the metrics microservice on Go demonstrates lower computational resource consumption and shorter response time under the same load conditions, which justifies the feasibility of its use when building microservice monitoring systems in cloud environments.

Спірінцев В'ячеслав Васильович – к.т.н., доцент, доцент кафедри програмного забезпечення комп'ютерних систем НТУ «Дніпровська політехніка».
ORCID: https://orcid.org/0000-0002-0908-1180
Спірінцева Ольга Володимирівна – к.т.н., доцент, доцент кафедри електронних обчислювальних машин Дніпровського національного університету імені Олеся Гончара.
ORCID: https://orcid.org/0000-0002-5050-5985
Дубіна Єгор Сергійович – магістр, НТУ "Дніпровська політехніка".
ORCID: https://orcid.org/0009-0004-5693-1696

Spirintsev Viacheslav Vasyliovych – candidate of technical sciences, ass.professor, associate professor of computer system's software department of the Dnipro University of Technology.
ORCID: https://orcid.org/0000-0002-0908-1180
Spirintseva Olga Volodymyrivna – candidate of technical sciences, ass. professor, associate professor of Computer Systems Department of the Oles Honchar Dnipro National University.
ORCID: https://orcid.org/0000-0002-5050-5985
Dubina Yegor Serhiyovych – master, Dnipro University of Technology.
ORCID: https://orcid.org/0009-0004-5693-1696

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)