КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (24), 2024
ISSN 2663 - 4023
DOI 10.28925/2663-4023.2024.24.341350
УДК 004.03

Скляренко Олена Вікторівна
Приватний вищий навчальний заклад «Європейський університет»
ORCID ID: 0000-0001-6555-1223
olena.skliarenko@e-u.edu.ua

Савченко Ярослав Олександрович
Приватний вищий навчальний заклад «Європейський університет»
ORCID ID: 0009-0008-0381-0224
yaroslav.savchenko@e-u.edu.ua

Литвиненко Леонід Олександрович
Приватний вищий навчальний заклад «Європейський університет»
ORCID ID: 0000-0002-0828-383X
leonid.lytvynenko@e-u.edu.ua

Сушинський Орест Євгенович
Приватний вищий навчальний заклад «Європейський університет»
ORCID ID: 0000-0002-2661-6458
orest.sushynskyi@e-u.edu.ua

АРХІТЕКТУРНІ ПІДХОДИ ДО РОЗРОБКИ МАСШТАБОВАНИХ ВЕБ-ЗАСТОСУНКІВ

Анотація. У цій статті досліджуються сучасні методи та технології для створення масштабованих вебзастосунків. Потреба у таких системах постійно зростає через збільшення обсягів даних і кількості користувачів, що вимагає високої продуктивності та надійності. Ця стаття присвячена вивченню сучасних способів масштабування вебзастосунків, що стає однією з найактуальніших проблем сучасного програмування, обумовленою стрімким зростанням обсягів даних та кількості користувачів. Масштабованість визначає здатність системи ефективно обробляти зростаюче навантаження за рахунок додавання ресурсів (процесорів, пам'яті, серверів) без шкоди для продуктивності. Недотримання принципів масштабованості може призвести до уповільнення роботи, збоїв та втрати користувачів, роблячи розробку масштабованих вебзастосунків пріоритетним завданням для програмних інженерів. У статті розглядаються ключові проблеми, з якими стикаються розробники, включаючи зростання навантаження на сервери, забезпечення безперебійної роботи під час пікових навантажень, оптимізацію ресурсів та гарантування високої доступності даних. Авторами проаналізовано та порівняно різні архітектурні підходи, такі як мікросервісна архітектура, використання хмарних сервісів, кешування, балансування навантаження та асинхронні черги повідомлень. Автори наводять приклади успішного застосування цих підходів у таких компаніях, як Netflix, Spotify, Facebook, Amazon та LinkedIn, та пропонують практичні рекомендації щодо вибору оптимальної архітектури для різних типів проєктів, враховуючи їх особливості та вимоги до продуктивності. Авторами підкреслюється важливість ретельного планування архітектури на початкових етапах розробки вебдодатків для забезпечення їх масштабованості та ефективності. Перспективи подальших досліджень включають розробку нових методів та інструментів для підвищення ефективності існуючих архітектурних підходів, інтеграцію різних методів для створення гнучких та масштабованих рішень, а також підвищення уваги до питань безпеки та управління ресурсами.

Ключові слова: масштабованість; вебзастосунки; мікросервісна архітектура; хмарні сервіси; надійність системи.

ВСТУП

Постановка проблеми. Сучасні вебзастосунки повинні бути здатні обробляти великі обсяги даних, забезпечуючи високий рівень доступності та швидкості роботи. Основні проблеми, з якими стикаються розробники під час створення масштабованих вебзастосунків, включають зростання навантаження на сервери зі збільшенням кількості користувачів, забезпечення безперебійної роботи під час пікових навантажень, оптимізацію використання ресурсів для зниження витрат, а також гарантування високої доступності та швидкості доступу до даних. Розв'язання цих проблем є важливим як з наукової точки зору, так і для практичного впровадження ефективних рішень у розробці вебзастосунків.

Аналіз досліджень та публікацій. Питання масштабованості вебзастосунків активно досліджуються багатьма науковцями та інженерами. Існує декілька архітектурних підходів, які дозволяють ефективно вирішувати ці проблеми. Серед них особливу увагу приділяють мікросервісній архітектурі, використанню хмарних сервісів, кешуванню, балансуванню навантаження та асинхронним чергам повідомлень. Дослідження у цій галузі зосереджені на підвищенні продуктивності, надійності та гнучкості вебзастосунків. Аналіз літератури показує, що ці підходи є ключовими для забезпечення масштабованості та ефективності сучасних вебсистем, що робить їх важливими для подальших досліджень та практичного застосування.

Мета статті. Мета цього дослідження полягає в порівнянні різних архітектурних підходів до розробки масштабованих вебзастосунків. Основними завданнями є аналіз переваг і недоліків кожного підходу, визначення умов їх ефективного використання, а також надання практичних рекомендацій щодо вибору оптимальної архітектури для конкретних проєктів. Автори прагнуть надати всебічний огляд сучасних методів та інструментів, що сприяють створенню високоефективних і масштабованих вебсистем.

ВИКЛАД ОСНОВНОГО МАТЕРІАЛУ

Архітектурні підходи до розробки масштабованих вебзастосунків є ключовими елементами, що визначають їх продуктивність, надійність та гнучкість. У сучасному середовищі, де кількість користувачів і обсяги даних постійно зростають, вибір відповідної архітектури стає критично важливим. Цей розділ дослідження присвячений детальному аналізу основних архітектурних підходів, таких як мікросервісна архітектура, використання хмарних сервісів, кешування, балансування навантаження та асинхронні черги повідомлень. Кожен з цих підходів буде розглянутий з точки зору його переваг, недоліків та умов ефективного застосування, що дозволить сформулювати рекомендації щодо вибору оптимальної архітектури для різних типів вебзастосунків.

Мікросервісна архітектура

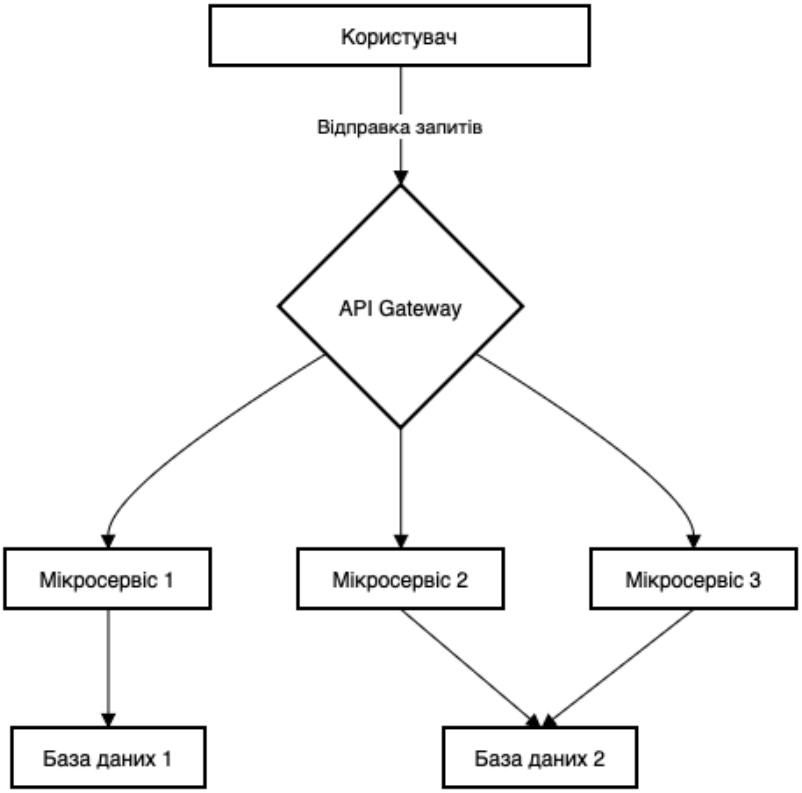

Рис. 1. Мікросервісна архітектура

Мікросервісна архітектура розділяє застосунок на незалежні, дрібні сервіси, які взаємодіють через API. Кожен сервіс відповідає за певну функціональність застосунку та може бути розгорнутий і масштабований незалежно від інших.

Переваги мікросервісної архітектури включають її гнучкість у розробці та розгортанні, можливість масштабування окремих сервісів та полегшене тестування і відлагодження. Мікросервіси дозволяють командам працювати над різними частинами застосунку незалежно, що пришвидшує розробку та впровадження нових функцій. Впровадження мікросервісів дозволяє значно знизити час на розгортання нових функцій. У разі зростання навантаження на конкретний сервіс його можна масштабувати незалежно від інших частин системи, що дозволяє ефективніше використовувати ресурси та суттєво знизити витрати у порівнянні з монолітними архітектурами. Крім того, мікросервіси можуть бути тестовані ізольовано, що знижує ймовірність помилок і полегшує відлагодження, зменшуючи кількість критичних помилок.

Проте, є й недоліки мікросервісної архітектури. Зростання кількості мікросервісів ускладнює управління взаємодією між ними, що вимагає впровадження додаткових інструментів для моніторингу та управління, що може збільшити витрати. Для ефективного управління мікросервісами також потрібні системи оркестрації, такі як Kubernetes, що додає додаткові вимоги до інфраструктури, вимагаючи додаткових ресурсів на управління.

Яскравим прикладом використання мікросервісної архітектури є Netflix, який застосовує цей підхід для забезпечення високої масштабованості та надійності своїх сервісів. Завдяки мікросервісній архітектурі компанія може швидко впроваджувати нові функції та масштабувати окремі сервіси відповідно до потреб. Використання мікросервісів дозволило Netflix значно збільшити кількість релізів програмного забезпечення.

Використання хмарних сервісів

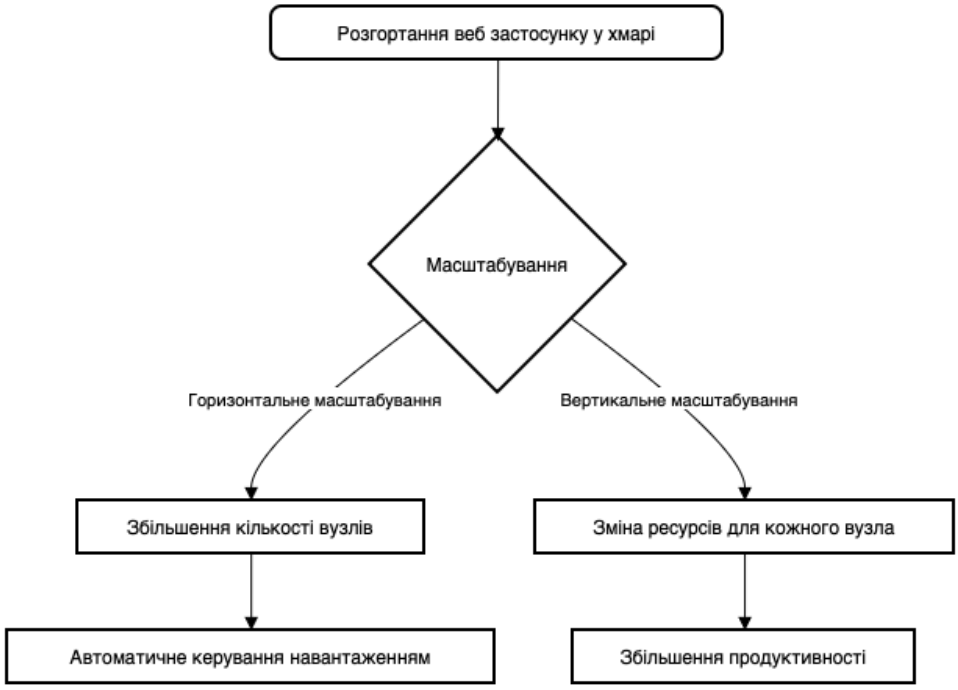

Рис. 2. Використання хмарних сервісів

Використання хмарних сервісів є важливим архітектурним підходом для розробки масштабованих вебзастосунків. Хмарні платформи, такі як AWS, Google Cloud та Azure, надають інструменти для автоматичного масштабування та управління ресурсами, дозволяючи швидко збільшувати або зменшувати обсяги ресурсів у залежності від поточного навантаження.

Переваги хмарних сервісів включають автоматичне масштабування, економію на інфраструктурі та високу доступність. Хмарні сервіси автоматично додають або зменшують ресурси залежно від навантаження, забезпечуючи безперебійну роботу застосунків. Автоматичне масштабування може сильно знизити витрати на інфраструктуру. Використання хмарних сервісів дозволяє уникнути витрат на придбання та обслуговування фізичної інфраструктури, що також може знизити капітальні витрати. Крім того, хмарні провайдери забезпечують високу доступність та надійність своїх сервісів завдяки географічному розподіленню дата-центрів. Використання хмарних сервісів може підвищити доступність систем до 99.99%.

Однак, є і недоліки. Використання хмарних сервісів означає залежність від конкретного постачальника, що може бути ризиком у випадку змін умов обслуговування або технічних проблем. Наприклад, під час збою AWS у 2020 році багато компаній

зазнали значних втрат. Також переміщення даних у хмару вимагає особливої уваги до питань безпеки та конфіденційності, і багато розробників стикаються з проблемами безпеки при переході до хмарних сервісів.

Яскравим прикладом використання хмарних сервісів є Spotify, який використовує хмарні сервіси для обробки великих обсягів даних та забезпечення безперебійного потокового передавання музики. Хмарні платформи дозволяють компанії швидко масштабувати ресурси відповідно до зростаючих потреб користувачів. Завдяки хмарним сервісам Spotify змогла збільшити кількість активних користувачів з 90 мільйонів у 2015 році, до більш ніж 500 мільйонів у 2023.

Архітектура з використанням кешування

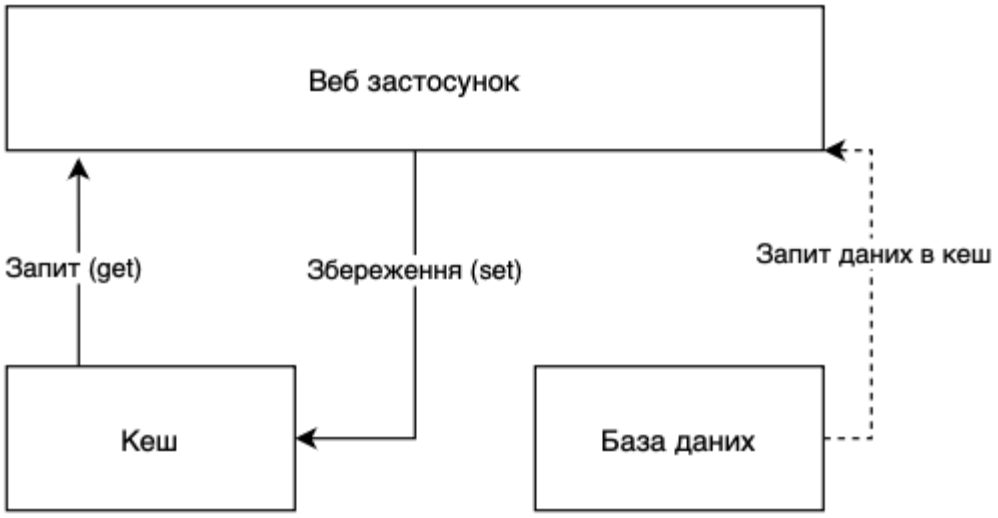

Рис. 3. Архітектура з використанням кешування

Архітектура з використанням кешування дозволяє зберігати часто використовувані дані в кешах, таких як Redis або Memcached, щоб зменшити навантаження на основні бази даних. Це значно знижує час відповіді та підвищує продуктивність системи.

Переваги кешування включають зниження часу відповіді та зменшення навантаження на бази даних. Зберігання часто запитуваних даних у кеші дозволяє значно знизити час відповіді, покращуючи користувацький досвідю Кешування також зменшує кількість запитів до основної бази даних, що підвищує її продуктивність і знижує ризик збоїв.

Проте, є і недоліки кешування. Управління консистентністю кешу може бути складним завданням, особливо у випадку частої зміни даних. Управління консистентністю кешу може суттєво збільшити складність системи. Впровадження кешування також може вимагати значних зусиль і додаткових налаштувань. Інтеграція Redis з існуючими системами може зайняти досить значний період часу та ресурсів, в залежності від поточної архітектури, розміру та інших параметрів системи.

Прикладом використання кешування є Facebook, який використовує кешування для прискорення доступу до даних користувачів, зберігаючи копії часто запитуваних даних у кешах для зниження навантаження на основні бази даних. Використання Memcached дозволило компанії значно знизити час відповіді на запити користувачів.

Архітектура з балансуванням навантаження

Рис. 4. Архітектура з балансуванням навантаження

Архітектура з балансуванням навантаження передбачає використання балансувальників навантаження, таких як NGINX чи HAProxy, для розподілу трафіку між кількома серверами, забезпечуючи рівномірне завантаження та підвищуючи надійність системи.

Переваги балансування навантаження включають покращення продуктивності та високу надійність. Розподіл трафіку між кількома серверами дозволяє збільшити продуктивність системи та забезпечити швидкий відгук на запити користувачів. Використання балансувальників навантаження може значно підвищити продуктивність системи. Крім того, балансувальники навантаження допомагають уникнути єдиної точки відмови, забезпечуючи безперебійну роботу системи навіть у випадку відмови одного з серверів. Також, використання балансувальників навантаження дозволяє знизити кількість простоїв.

Однак, є і недоліки. Налаштування балансувальників навантаження може бути складним завданням, що вимагає високого рівня технічних знань. Незважаючи на підвищену надійність, балансувальники навантаження самі по собі можуть стати точкою відмови, якщо не забезпечено їхнє резервування. Відмова балансувальника може призвести до простою всієї системи.

Прикладом використання балансувальників навантаження є Amazon, який використовує їх для забезпечення високої доступності своїх сервісів, розподіляючи трафік між численними дата-центрами по всьому світу. Використання балансувальників дозволило компанії забезпечити доступність систем 99.99%.

Використання асинхронних черг повідомлень

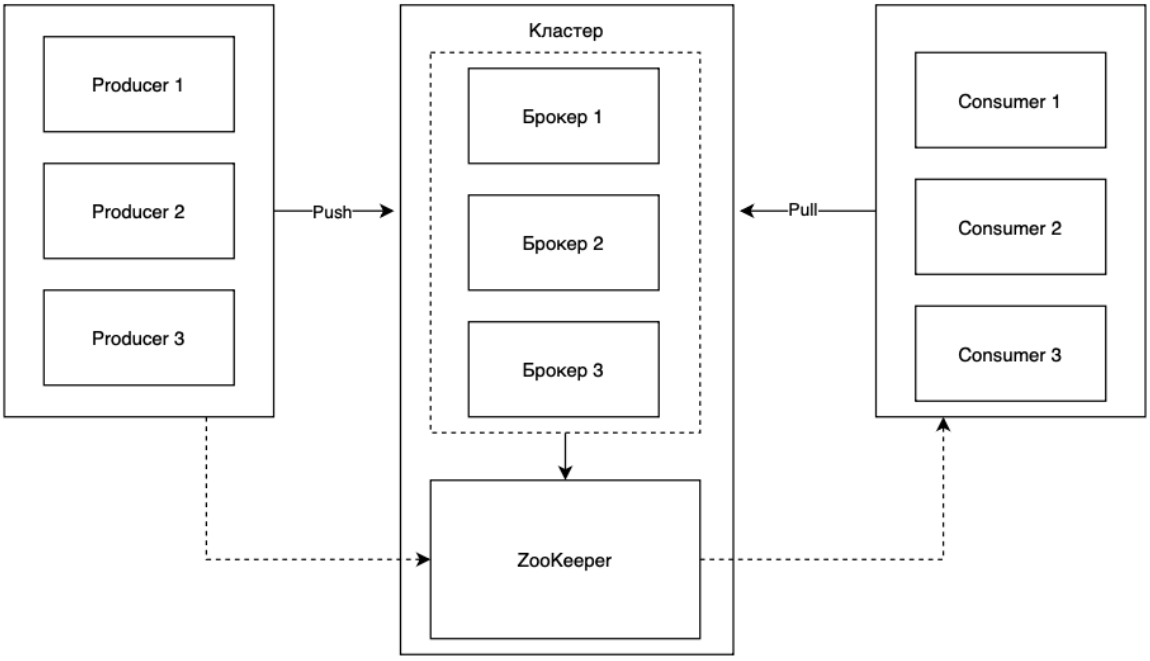

Рис. 5. Використання асинхронних черг повідомлень

Використання асинхронних черг повідомлень є важливим архітектурним підходом для забезпечення масштабованості вебзастосунків. Системи черг, такі як RabbitMQ або Apache Kafka, дозволяють асинхронну обробку завдань, що розвантажує основні сервіси та дозволяє обробляти великі обсяги даних.

Асинхронна обробка завдань дозволяє основним сервісам продовжувати роботу без затримок, передаючи обробку великих обсягів даних у фоновий режим. Черги повідомлень дозволяють знизити навантаження на основні сервіси, а також обробляти значні обсяги даних паралельно, що підвищує загальну продуктивність системи.

Проте, є і недоліки. Впровадження асинхронних черг повідомлень може вимагати значних зусиль і додаткових налаштувань. Крім того, асинхронна обробка може призводити до затримок у виконанні завдань, що може бути неприйнятним для деяких критичних систем.

Прикладом використання асинхронних черг повідомлень є LinkedIn, який використовує Apache Kafka для обробки великих обсягів даних у режимі реального часу, забезпечуючи безперебійну роботу своїх сервісів і швидкий відгук на дії користувачів.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Масштабованість є критичним аспектом розробки сучасних вебзастосунків. Використання мікросервісної архітектури, хмарних сервісів, кешування, балансування навантаження та асинхронних черг повідомлень дозволяє ефективно вирішувати проблеми зростання навантаження та забезпечувати високу продуктивність і надійність систем.

Кожен з цих підходів має свої переваги та недоліки, і вибір конкретного підходу залежить від специфіки проєкту, вимог до продуктивності, бюджетних обмежень та інших факторів. Важливо ретельно планувати архітектуру на ранніх етапах розробки, щоб уникнути складнощів у майбутньому та забезпечити можливість подальшого масштабування системи.

Перспективи подальших досліджень у цьому напрямі включають розвиток нових методів і інструментів для підвищення ефективності існуючих архітектурних підходів. Також варто досліджувати інтеграцію різних підходів для створення ще більш гнучких і масштабованих рішень, що здатні адаптуватися до швидких змін у вимогах та умовах експлуатації. Окрема увага має бути приділена безпеці та управлінню ресурсами, щоб забезпечити надійність і стабільність систем навіть при високих навантаженнях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly Media.
2. Siriwarden,a P., Indrasiri, K. (2018). *Microservices for the enterprise: designing, developing, and deploying.* Apress.
3. Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems.* O'Reilly Media.
4. Davis, C. (2019). *Cloud Native Patterns: Designing change-tolerant software.* Manning.
5. Petoff, J., Murphy, N., Beyer, B., & Jones, C. (2016). *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media.
6. Abbott, M. L., Fisher, M. T. (2015). *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.* Addison-Wesley Professional.
7. Atchison, L. (2020). *Architecting for Scale: How to Maintain High Availability and Manage Risk in the Cloud.* O'Reilly Media.
8. Ejsmont, A. (2015). *Web Scalability for Startup Engineers.* McGraw Hill.
9. Abbott, M. L., Fisher, M. T. (2011). *Scalability Rules: 50 Principles for Scaling Web Sites.* Addison-Wesley Professional.
10. Netflix. (n.d.). *Microservices.* https://netflixtechblog.com/tagged/microservices
11. Google. (n.d.). *Microservices architecture on Google Cloud.* https://cloud.google.com/blog/topics/developers-practitioners/microservices-architecture-google-cloud
12. *Spotify on Google Cloud.* (n.d.). https://cloud.google.com/customers/spotify
13. Facebook. (n.d.). *Scaling Memcache at Facebook.* https://research.facebook.com/publications/scaling-memcache-at-facebook/
14. Huang, Y., et al. (2007). Achieving Flexible Cache Consistency for Pervasive Internet Access. *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07).* https://doi.org/10.1109/PERCOM.2007.6
15. Google. (n.d.). *Integrating Redis with Cloud Run.* https://cloud.google.com/memorystore/docs/redis/connect-redis-instance-cloud-run
16. Microsoft. (n.d.). *Load Balancer documentation.* https://learn.microsoft.com/en-us/azure/load-balancer/
17. Netflix. (n.d.). *Chaos Monkey.* https://netflix.github.io/chaosmonkey/
18. Google. (n.d.). *Cloud Load Balancing overview.* https://cloud.google.com/load-balancing/docs/load-balancing-overview
19. *Amazon Elastic Load Balancing.* (n.d.). https://aws.amazon.com/elasticloadbalancing/
20. *Brief History of Scaling Uber.* (n.d.). https://www.linkedin.com/pulse/brief-history-scaling-uber-josh-clemm-dfqgc
21. *Streaming logging pipeline of Home timeline prediction system.* (n.d.). https://blog.x.com/engineering/en_us/topics/infrastructure/2020/streaming-logging-pipeline-of-home-timeline-prediction-system
22. Lytvynenko, L., Skliarenko, O., & Kolodinska, Y. (2020). Logic of building interfaces in the software. *Bulletin of the National Technical University «KhPI» Series: New Solutions in Modern Technologies, 1(3),* 54–59. https://doi.org/10.20998/2413-4295.2020.03.07

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (24), 2024
ISSN 2663 - 4023

Olena Skliarenko
Private Higher Educational Establishment "European University"
ORCID ID: 0000-0001-6555-1223
olena.skliarenko@e-u.edu.ua

Yaroslav Savchenko
Private Higher Educational Establishment "European University"
ORCID ID: 0009-0008-0381-0224
yaroslav.savchenko@e-u.edu.ua

Leonid Lytvynenk
Private Higher Educational Establishment "European University"
ORCID ID: 0000-0002-0828-383X
leonid.lytvynenko@e-u.edu.ua

Orest Sushynskyi
Private Higher Educational Establishment "European University"
ORCID ID: 0000-0002-2661-6458
orest.sushynskyi@e-u.edu.ua

ARCHITECTURAL APPROACHES TO THE DEVELOPMENT OF SCALABLE WEB APPLICATIONS

Abstract. This article explores modern methods and technologies for creating scalable web applications. The need for such systems is constantly growing due to the increase in data volumes and the number of users, which requires high performance and reliability. This article is devoted to the study of modern methods of scaling web applications, which is becoming one of the most pressing problems of modern programming due to the rapid growth of data and the number of users. Scalability determines the ability of a system to efficiently handle an increasing load by adding resources (processors, memory, servers) without compromising performance. Failure to comply with the principles of scalability can lead to slowdowns, failures, and loss of users, making the development of scalable web applications a priority for software engineers. This article discusses the key challenges faced by developers, including increasing server load, ensuring uninterrupted operation during peak loads, optimizing resources, and ensuring high data availability. The authors analyze and compare various architectural approaches, such as microservice architecture, use of cloud services, caching, load balancing, and asynchronous message queues. The authors provide examples of successful application of these approaches in companies such as Netflix, Spotify, Facebook, Amazon, and LinkedIn, and offer practical recommendations for choosing the optimal architecture for different types of projects, taking into account their features and performance requirements. The authors emphasize the importance of careful architecture planning at the initial stages of web application development to ensure scalability and efficiency. Prospects for further research include the development of new methods and tools to improve the efficiency of existing architectural approaches, the integration of different methods to create flexible and scalable solutions, and increased attention to security and resource management.

Keywords: scalability; web applications; microservice architecture; cloud services; system reliability.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Kleppmann, M. (2017). *Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems.* O'Reilly Media.
2. Siriwarden,a P., Indrasiri, K. (2018). *Microservices for the enterprise: designing, developing, and deploying.* Apress.
3. Newman, S. (2015). *Building Microservices: Designing Fine-Grained Systems.* O'Reilly Media.
4. Davis, C. (2019). *Cloud Native Patterns: Designing change-tolerant software.* Manning.

5. Petoff, J., Murphy, N., Beyer, B., & Jones, C. (2016). *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media.
6. Abbott, M. L., Fisher, M. T. (2015). *The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.* Addison-Wesley Professional.
7. Atchison, L. (2020). *Architecting for Scale: How to Maintain High Availability and Manage Risk in the Cloud.* O'Reilly Media.
8. Ejsmont, A. (2015). *Web Scalability for Startup Engineers.* McGraw Hill.
9. Abbott, M. L., Fisher, M. T. (2011). *Scalability Rules: 50 Principles for Scaling Web Sites.* Addison-Wesley Professional.
10. Netflix. (n.d.). *Microservices.* https://netflixtechblog.com/tagged/microservices
11. Google. (n.d.). *Microservices architecture on Google Cloud.* https://cloud.google.com/blog/topics/developers-practitioners/microservices-architecture-google-cloud
12. *Spotify on Google Cloud.* (n.d.). https://cloud.google.com/customers/spotify
13. Facebook. (n.d.). *Scaling Memcache at Facebook.* https://research.facebook.com/publications/scaling-memcache-at-facebook/
14. Huang, Y., et al. (2007). Achieving Flexible Cache Consistency for Pervasive Internet Access. *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom'07)*. https://doi.org/10.1109/PERCOM.2007.6
15. Google. (n.d.). *Integrating Redis with Cloud Run.* https://cloud.google.com/memorystore/docs/redis/connect-redis-instance-cloud-run
16. Microsoft. (n.d.). *Load Balancer documentation.* https://learn.microsoft.com/en-us/azure/load-balancer/
17. Netflix. (n.d.). *Chaos Monkey.* https://netflix.github.io/chaosmonkey/
18. Google. (n.d.). *Cloud Load Balancing overview.* https://cloud.google.com/load-balancing/docs/load-balancing-overview
19. *Amazon Elastic Load Balancing.* (n.d.). https://aws.amazon.com/elasticloadbalancing/
20. *Brief History of Scaling Uber.* (n.d.). https://www.linkedin.com/pulse/brief-history-scaling-uber-josh-clemm-dfqgc
21. *Streaming logging pipeline of Home timeline prediction system.* (n.d.). https://blog.x.com/engineering/en_us/topics/infrastructure/2020/streaming-logging-pipeline-of-home-timeline-prediction-system
22. Lytvynenko, L., Skliarenko, O., & Kolodinska, Y. (2020). Logic of building interfaces in the software. *Bulletin of the National Technical University «KhPI» Series: New Solutions in Modern Technologies, 1(3),* 54–59. https://doi.org/10.20998/2413-4295.2020.03.07