https://doi.org/10.31891/2307-5732-2025-355-95
УДК 637.5.02

ФЕДОРИШИН БОГДАН
Національний Університет "Львівська Політехніка"
e-mail: bohdan.v.fedoryshyn@lpnu.ua

ПОРІВНЯЛЬНИЙ АНАЛІЗ ІНСТРУМЕНТІВ ДЛЯ ОРКЕСТРАЦІЇ СЕРВІСІВ В МІКРОСЕРВІСНІЙ АРХІТЕКТУРІ

В статті розглядається порівняння двох інструментів оркестрації - Docker swarm та Kubernetes, вибір цих технологій зумовлений їх широким застосуванням у практичних проєктах, що дозволяє отримати результати, які мають безпосередню практичну цінність для IT-індустрії. Проведено аналіз літератури на цю тему та виявлено, що в існуючій літературі мало приділяють уваги аналізу впливу архітектурних рішень на параметри доступності. Було проведено порівняльний аналіз, для імітації неполадок в системі було обрано три найпоширеніших типи збоїв - повна зупинка одного вузла, недоступність мережевого зв'язку між вузлами, раптовий пік трафіку. Кожний сценарій був відтворений 10 раз. В якості критеріїв оцінки доступності системи було обрано - середній час відновлення після збою та відсоток доступності сервісів. Результати дослідження показали, що Kubernetes має значно вищий рівень доступності порівняно з Docker Swarm у всіх протестованих сценаріях. Під час імітації збою вузла Kubernetes демонструє доступність на рівні 99.95%, тоді як Docker Swarm забезпечує лише 95.20%. При моделюванні мережевих затримок Kubernetes зберігає доступність на рівні 99.85%, що на 3.10% вище, ніж у Docker Swarm (96.75%). При моделюванні різких пікових навантажень (Load Spike) Kubernetes продемонстрував стійкість до змінного навантаження, забезпечивши 99.92% доступності, у Docker Swarm цей показник становив 95.10%. При повній зупинці вузла, Kubernetes відновлює роботу в середньому за 15 секунд, тоді як Docker Swarm потребує 42 секунди. Отримані результати мають безпосередню практичну цінність для розробників, архітекторів програмного забезпечення та спеціалістів DevOps, дослідження показує, що Kubernetes є кращим інструментом для забезпечення високої доступності сервісів в мікросервісній архітектурі.

Ключові слова: High availability, Docker, Kubernetes. microservices architecture.

FEDORYSHYN BOHDAN
National University "Lviv Polytechnic"

COMPARATIVE ANALYSIS OF SERVICE ORCHESTRATION TOOLS IN MICROSERVICES ARCHITECTURE

The article examines a comparison of two orchestration tools—Docker Swarm and Kubernetes. The choice of these technologies is driven by their widespread use in practical projects, allowing for results that have direct practical value for the IT industry. A literature review on this topic was conducted, revealing that existing research pays little attention to analyzing the impact of architectural decisions on availability parameters. A comparative analysis was performed, and three of the most common types of failures were selected for system failure simulation: the complete shutdown of a node, network communication loss between nodes, and a sudden traffic spike. Each scenario was reproduced ten times. The criteria for assessing system availability were chosen as the average recovery time after failure and the percentage of service availability. The research results showed that Kubernetes has a significantly higher level of availability compared to Docker Swarm in all tested scenarios. In the case of node failure simulation, Kubernetes demonstrated an availability level of 99.95%, whereas Docker Swarm provided only 95.20%. When modeling network delays, Kubernetes maintained an availability of 99.85%, which is 3.10% higher than Docker Swarm (96.75%). During the simulation of sudden peak loads (Load Spike), Kubernetes exhibited resilience to fluctuating workloads, ensuring 99.92% availability, while Docker Swarm achieved only 95.10%. In the event of a complete node shutdown, Kubernetes restored operation within an average of 15 seconds, whereas Docker Swarm required 42 seconds. The obtained results have direct practical value for developers, software architects, and DevOps specialists. The study demonstrates that Kubernetes is a superior tool for ensuring high service availability in a microservices architecture.

Keywords: High availability, Docker, Kubernetes. microservices architecture.

Стаття надійшла до редакції / Received 09.06.2025
Прийнята до друку / Accepted 26.06.2025

Вступ

У сучасних розподілених інформаційних системах критично важливим фактором є забезпечення високої доступності сервісів. В умовах динамічних навантажень, потенційних апаратних відмов та мінливих мережевих умов ефективне управління ресурсами та автоматичне відновлення після збоїв залишаються ключовими завданнями для будь-якої архітектури програмного забезпечення. Широке застосування технологій оркестрації контейнеризованих середовищ дозволяє автоматизувати балансування навантаження, горизонтальне масштабування та механізми самовідновлення, що значно покращує надійність програмних платформ. Традиційні монолітні архітектури характеризуються відносною простотою впровадження, проте мають суттєві обмеження у масштабованості та гнучкості під час адаптації до змін у навантаженні. Глобальні відмови, що виникають унаслідок виходу з ладу одного з компонентів, можуть спричиняти тривалі періоди простою всієї системи. Перехід до мікросервісної архітектури дозволяє розподілити обчислювальні ресурси між незалежними сервісами, підвищуючи загальну відмовостійкість і локалізуючи наслідки збоїв у межах окремих модулів. Забезпечення високої доступності в контейнеризованих середовищах потребує ефективного балансування навантаження, коректної маршрутизації трафіку та своєчасного масштабування сервісів відповідно до змін у вимогах до обчислювальних ресурсів. Швидкість та точність авто-масштабування

безпосередньо впливають на час відновлення після збоїв і загальну стабільність інфраструктури. Сучасні платформи оркестрації контейнерів реалізують механізми відмовостійкості з різним рівнем ефективності. Одним із найбільш поширених рішень є Kubernetes, що забезпечує розширену систему управління ресурсами, адаптивне масштабування та оптимізовану маршрутизацію запитів. Альтернативний підхід реалізований у Docker Swarm, що пропонує базові можливості оркестрації контейнерів та простішу конфігурацію, однак поступається у швидкості відновлення та адаптивності до змінного навантаження.

Визначення ефективності механізмів масштабування та відновлення після збоїв є важливим завданням при виборі платформи оркестрації. Порівняльний аналіз продуктивності, стійкості до збоїв і швидкості реакції на критичні події дозволяє зробити обґрунтовані висновки щодо доцільності використання кожного підходу залежно від вимог до надійності та безперервної роботи інформаційної системи.

Результати дослідження

В існуючих дослідженнях активно розглядається питання вибору та аналізу ефективності контейнерних технологій. Зокрема, проводиться якісний і кількісний аналіз різних контейнерних рушіїв, включно з оцінкою їх продуктивності, ресурсних витрат та функціональних можливостей [1]. Автори акцентують увагу на впливі вибору технологій контейнеризації на загальну доступність і продуктивність, однак не проводять аналіз впливу архітектурних рішень (монолітна або мікросервісна архітектура) на параметри високої доступності. У низці праць активно досліджується використання Kubernetes для оркестрації та управління контейнерами з метою оптимізації ресурсів і автоматичного масштабування. Наприклад, запропоновано емпіричні моделі прогнозування ресурсоспоживання систем, розгорнутих у Kubernetes, що дозволяють підвищити ефективність оркестрації [2]. Також досліджуються підходи до авто-масштабування на базі машинного навчання, які дозволяють підвищити ефективність управління ресурсами і швидкість реагування систем на зміни навантаження [3, 4]. Проте ці роботи не містять аналізу часу відновлення після збоїв, що є важливим критерієм для повноцінної оцінки високої доступності архітектур. Також розглядаються підходи, орієнтовані на розподілене розміщення та балансування навантаження між мікросервісами у гетерогенних та федеративних середовищах типу Fog-Cloud, що дозволяє значно скоротити час реакції системи на запити користувачів [5]. У подібних дослідженнях акцент робиться переважно на оптимальному розподілі ресурсів та навантаження між мікросервісами, проте вплив архітектурних рішень та наслідки збоїв не досліджуються в достатній мірі. Окремі дослідження присвячено порівняльному аналізу масштабованості фреймворків для обробки потокових даних, де продемонстровано, що мікросервісні архітектури забезпечують майже лінійну масштабованість за умови достатніх ресурсів [6]. Однак ці дослідження також не охоплюють питання порівняння різних типів архітектур (мікросервіси проти монолітів) з позиції їхньої стійкості до збоїв. Проведений аналіз наукових джерел показує, що на сьогодні відсутні комплексні дослідження, які б порівнювали монолітні та мікросервісні архітектури за параметрами доступності та часу відновлення після збоїв із застосуванням методів хаос-інженерії. Це обумовлює актуальність та новизну проведення такого дослідження. З метою порівняння ефективності монолітної та мікросервісної архітектур щодо забезпечення високої доступності (High Availability) і швидкості відновлення після збоїв за допомогою методів хаос-інженерії було проведене дослідження технологій Kubernetes та Docker Swarm, як представників відповідно мікросервісної та монолітної (або спрощеної мікросервісної) архітектур. Для проведення дослідження було створено два експериментальних середовища, які моделюють різні архітектурні підходи Docker Swarm (монолітна архітектура) та Kubernetes (мікросервісна архітектура). Вибір саме цих технологій зумовлений їх широким застосуванням у практичних проектах, що дозволяє отримати результати, які мають безпосередню практичну цінність для IT-індустрії. Kubernetes є найбільш поширеним оркестратором для складних, високонавантажених мікросервісних застосунків завдяки його широким можливостям автоматичного масштабування, самовідновлення та гнучкої конфігурації. Docker Swarm, у свою чергу, є простішим у налаштуванні та використанні рішенням, що часто застосовується для менших або простіших проектів, які не вимагають складного оркестрування та динамічного масштабування. Експериментальні середовища створювалися з використанням ідентичних апаратних ресурсів для забезпечення чистоти експерименту та об'єктивності отриманих результатів. Кожне середовище було розгорнуте на віртуальних машинах з такими параметрами: 8 віртуальних CPU, 16 ГБ оперативної пам'яті та 100 ГБ SSD. Методика хаос-інженерії була обрана через її ефективність у симуляції реалістичних сценаріїв збоїв і навантажень, які часто виникають у реальних умовах експлуатації інформаційних систем. Для реалізації сценаріїв збоїв були обрані спеціалізовані інструменти, що є стандартними і широко використовуваними в індустрії: LitmusChaos для Kubernetes і Chaos Toolkit для Docker Swarm. LitmusChaos дозволяє швидко й гнучко конфігурувати та проводити сценарії збою, інтегруючись із Kubernetes на високому рівні. Chaos Toolkit було обрано завдяки його гнучкості та простоті інтеграції в системи на базі Docker Swarm.

Було обрано три найбільш поширені та критичні для інформаційних систем типи збоїв:

- Повна зупинка одного вузла (Node Failure): такий сценарій дозволяє оцінити здатність системи автоматично перерозподіляти навантаження і відновлювати роботу у випадку втрати одного або декількох вузлів.
- Недоступність мережевого зв'язку між вузлами (Network Latency): цей тип збою симулює ситуацію, коли виникають проблеми в мережі, що є однією з найпоширеніших причин деградації роботи розподілених систем.

- Раптовий пік трафіку (Load Spike): моделює ситуації раптового збільшення навантаження на систему, що може спричинити деградацію продуктивності або відмову окремих сервісів.

Кожен сценарій збою відтворювався десять разів для забезпечення статистичної значущості результатів та зменшення впливу випадкових факторів.

Критерії оцінювання обиралися з огляду на їх важливість для кінцевих користувачів та бізнесу в цілому:

- Середній час відновлення після збою (Mean Time to Recovery, MTTR): це ключовий показник, який характеризує швидкість реагування і відновлення нормальної роботи системи після збою. MTTR вимірювався від моменту виникнення збою до повного відновлення нормальної роботи всіх компонентів системи.
- Відсоток доступності сервісів (Availability %): цей показник визначався як відношення часу, коли сервіс був доступний, до загального часу експерименту. Доступність обчислювалася автоматично за допомогою систем моніторингу, що фіксували факти простоїв та відновлень роботи системи.

Такий комплексний підхід дозволив отримати всебічну та об'єктивну оцінку ефективності архітектур у контексті їх високої доступності та стійкості до різних типів збоїв.

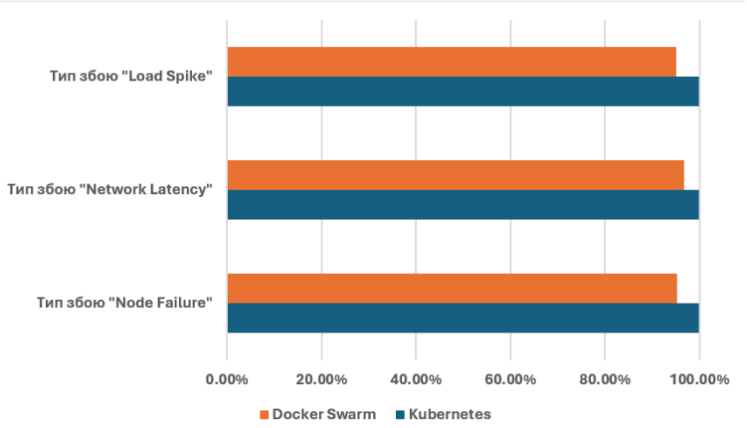

Рис. 1. Порівняння типів збою

Дослідження показало, що Kubernetes має значно вищий рівень доступності порівняно з Docker Swarm у всіх протестованих сценаріях. Під час імітації збою вузла Kubernetes демонструє доступність на рівні 99.95%, тоді як Docker Swarm забезпечує лише 95.20%. Це пояснюється тим, що Kubernetes швидко перерозподіляє навантаження на інші вузли та автоматично відновлює необхідні компоненти. Отримані показники доступності під час різних типів збоїв показані на рис. 1. При моделюванні мережевих затримок Kubernetes зберігає доступність на рівні 99.85%, що на 3.10% вище, ніж у Docker Swarm (96.75%). Такий результат досягається завдяки оптимізованій внутрішній маршрутизації Kubernetes, яка здатна швидко адаптуватися до змін у мережевому трафіку. При моделюванні різких пікових навантажень (Load Spike) Kubernetes продемонстрував найкращу стійкість до змінного навантаження, забезпечивши 99.92% доступності. У Docker Swarm цей показник становив лише 95.10%, що підтверджує меншу гнучкість цієї технології в умовах високих навантажень.

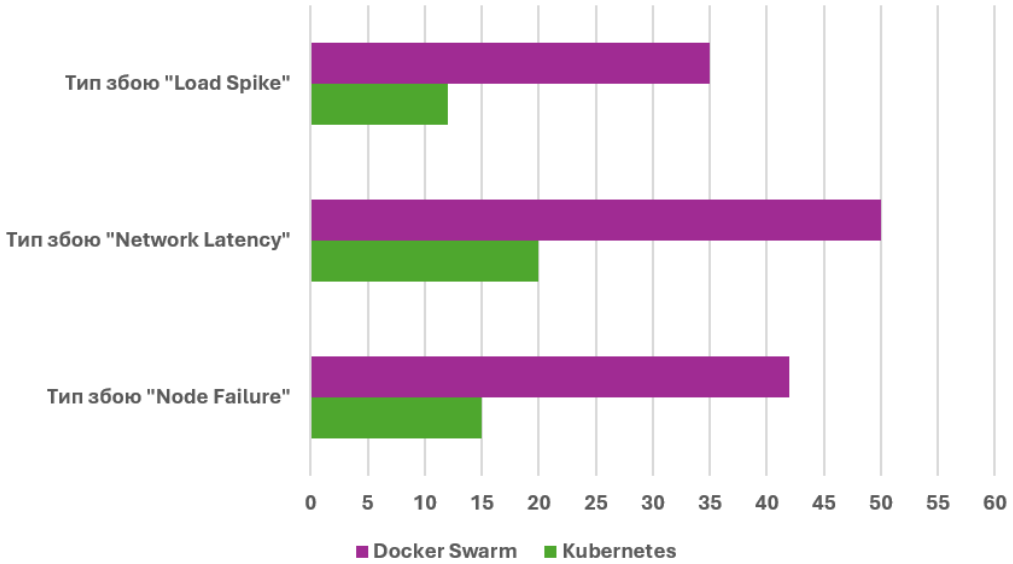

Рис. 2. Середній час відновлення збою (MTR)

Окрім показників доступності, дослідження включало аналіз середнього часу відновлення після збою (MTTR). Це один із ключових параметрів, який визначає швидкість реагування системи на критичні ситуації. Отримані показники зображено на рис. 2. Результати демонструють, що при повній зупинці вузла Kubernetes відновлює роботу в середньому за 15 секунд, тоді як Docker Swarm потребує 42 секунди. Така різниця пояснюється тим, що Kubernetes автоматично виявляє втрату вузла та розгортає нові репліки, тоді як Docker Swarm потребує ручного втручання або більш тривалого часу для переналаштування. При мережевих затримках Kubernetes демонструє MTTR у 20 секунд, тоді як Docker Swarm потребує 50 секунд, що майже у 2.5 рази довше. Це вказує на недостатню адаптивність Docker Swarm до проблем із мережею та менш ефективні алгоритми маршрутизації. Під час навантажувальних піків (Load Spike) Kubernetes справляється із ситуацією у 12 секунд, тоді як Docker Swarm відновлюється за 35 секунд. Це підтверджує, що Kubernetes краще масштабує сервіси під час змінного навантаження.

Аналіз отриманих результатів свідчить про значні переваги Kubernetes у забезпеченні високої доступності, швидкості відновлення після збоїв та ефективного масштабування під час пікових навантажень. Зокрема, дослідження продемонструвало, що рівень доступності Kubernetes суттєво перевищує показники Docker Swarm, що пояснюється розвиненими механізмами автоматичного балансування навантаження, самовідновлення та перерозподілу ресурсів. Це робить Kubernetes оптимальним вибором для розгортання критично важливих застосунків, які повинні функціонувати з мінімальними простоями та високими вимогами до безперервної роботи. Крім того, значне скорочення середнього часу відновлення після збою (MTTR) у Kubernetes у 2-3 рази порівняно з Docker Swarm підтверджує його здатність швидко реагувати на аварійні ситуації. Це забезпечується використанням розподіленої архітектури, яка автоматично відслідковує стан сервісів, перенаправляє запити у разі відмови окремих компонентів і динамічно перезапускає пошкоджені або недоступні екземпляри. Завдяки цьому Kubernetes дозволяє мінімізувати втрати продуктивності та зменшити ризики для бізнесу, які виникають через незаплановані простої. Окрім часу відновлення, важливим фактором є навантаження на системні ресурси під час роботи та в процесі відновлення після збоїв. Для аналізу було зібрано метрики використання процесорного часу (CPU Load) та оперативної пам'яті (RAM Utilization) в обох середовищах при штатній роботі та в період активного відновлення.

Як видно з рис. 3, в Kubernetes навантаження на ресурси в період аварійного відновлення залишається нижчим, ніж у Docker Swarm. Це свідчить про більш ефективне управління ресурсами та балансування процесів під час масштабування.

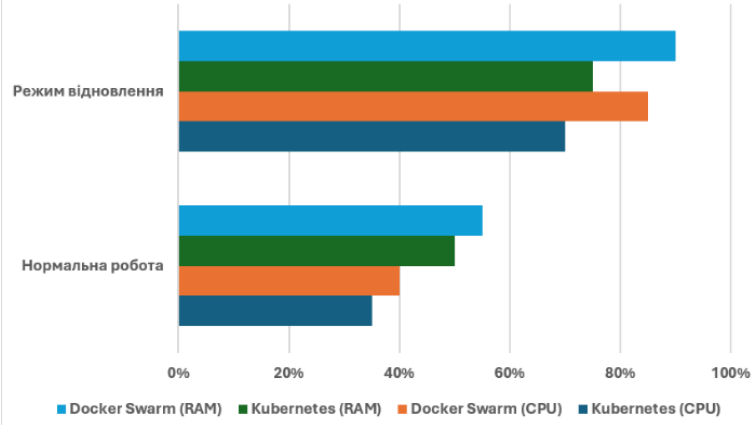

Рис. 3. Використання ресурсів (у відсотках від загальної потужності системи)

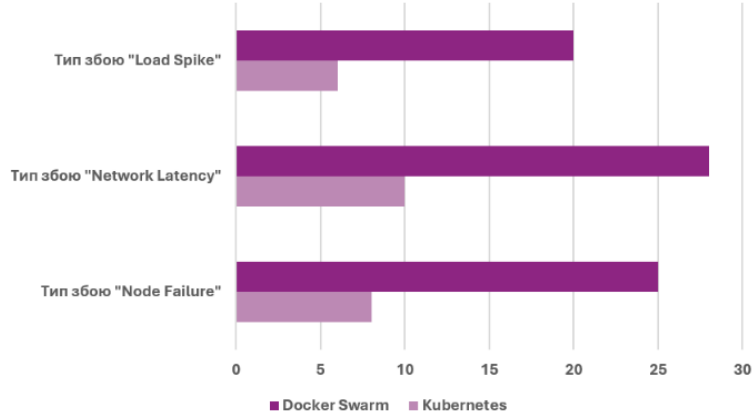

Рис. 4. Порівняння типів збою у Kubernetes та Docker Swarm.

Також у рамках проведеного дослідження було здійснено оцінку тривалості процесу автоматичного

розгортання додаткових екземплярів сервісів, що активуються після виявлення збою в системі. Аналіз проводився з метою визначення ефективності механізмів авто-масштабування кожної архітектури, а також порівняння швидкості реагування Kubernetes та Docker Swarm на аварійні ситуації. Отримані результати дають можливість оцінити, наскільки швидко кожна система здатна адаптуватися до змін у середовищі, забезпечуючи безперервність роботи сервісів і мінімізуючи час простою після критичних відмов. Результати проведеного аналізу швидкості автоматичного масштабування після кожного із розглянутих типів збоїв, отримані в процесі тестування, відображені на рис. 4.

Результати вимрювання свідчать про те, що Kubernetes у середньому демонструє значно вищу швидкість відновлення балансу обчислювальних потужностей та автоматичного розгортання нових реплік сервісів після виявлення збою, перевершуючи Docker Swarm у 2-4 рази. Така перевага обумовлена ефективністю вбудованих механізмів авто-масштабування, які забезпечують швидку адаптацію до змін у навантаженні та мінімізують вплив аварійних ситуацій на загальну продуктивність системи. Kubernetes має значно вищу гнучкість і стійкість до збоїв, забезпечуючи стабільну роботу навіть у критичних умовах. Це робить Kubernetes оптимальним вибором для високонавантажених і критично важливих систем, які вимагають мінімального часу відновлення та безперервної доступності сервісів.

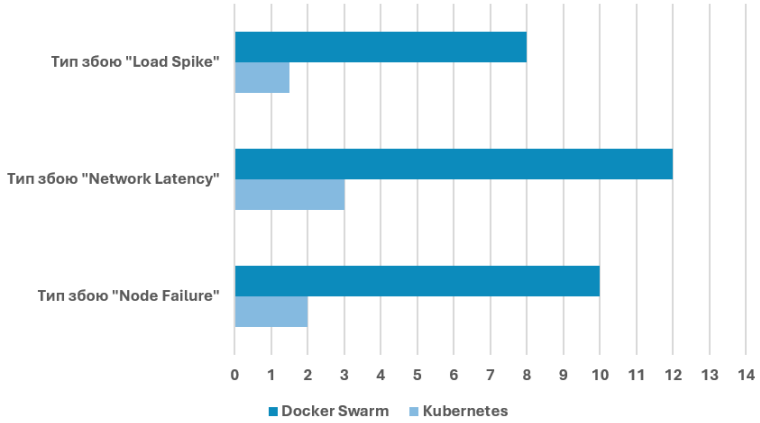

Рис. 5. Середній час перенаправлення трафіку після збою

На рис. 5 зображено результати аналізу маршрутизації трафіку після аварії. Kubernetes забезпечує значно швидше перенаправлення трафіку на активні вузли, що дозволяє мінімізувати вплив збоїв на кінцевих користувачів. Після втрати вузла або сервісу критично важливим фактором є швидкість перенаправлення запитів на здорові екземпляри сервісів. Ще одним важливим аспектом є адаптивність Kubernetes до змінного навантаження. Експериментальне моделювання пікових навантажень показало, що Kubernetes здатний динамічно масштабувати ресурси відповідно до поточного рівня навантаження, що дозволяє утримувати високу продуктивність навіть у випадках раптового збільшення кількості запитів. Це досягається завдяки вбудованим механізмам авто-масштабування та ефективного управління розподілом навантаження між сервісами. На відміну від цього, Docker Swarm демонструє нижчу ефективність у таких сценаріях через менш розвинені алгоритми балансування ресурсів, що призводить до більшого часу відгуку системи та можливих збоїв під навантаженням. Таким чином, результати дослідження підтверджують, що Kubernetes є значно більш надійним рішенням для побудови розподілених мікросервісних архітектур з високими вимогами до доступності та відмовостійкості. У той час як Docker Swarm може бути придатним для менш критичних або внутрішніх систем, його використання у високонавантажених середовищах або бізнес-критичних застосунках потребує додаткових механізмів захисту від збоїв та резервування ресурсів. Для забезпечення високої доступності та мінімального часу відновлення після збоїв доцільно використовувати Kubernetes, який забезпечує ефективне балансування навантаження, автоматичне масштабування та швидке реагування на несправності. Він є оптимальним вибором для розгортання критично важливих застосунків та сервісів із високими вимогами до безперервності роботи.

У випадках, коли простота розгортання має пріоритет над швидкістю відновлення після збоїв, Docker Swarm може бути ефективним рішенням. Його використання доцільне у невеликих проєктах або внутрішніх сервісах, де критичність відмови є відносно низькою, а адміністрування має бути максимально простим. Для середовищ, що характеризуються високим рівнем змінного навантаження, таких як IoT-системи або обробка великих даних (Big Data), Kubernetes є більш надійним вибором, оскільки дозволяє динамічно масштабувати ресурси та оперативно реагувати на зміну запитів, забезпечуючи стабільну роботу навіть при значних коливаннях навантаження. У разі використання Docker Swarm у критичних середовищах рекомендується передбачити додаткові механізми резервування, включаючи дублювання вузлів, використання зовнішніх балансувальників навантаження та налаштування ручного масштабування для зниження часу відновлення після збоїв. Результати дослідження показали, що Kubernetes є найкращим вибором для критичних і масштабованих систем, тоді як Docker Swarm може бути прийнятним рішенням для невеликих застосунків із менш жорсткими вимогами до доступності та відмовостійкості.

Висновки

Забезпечення високої доступності є ключовою вимогою для сучасних розподілених систем, особливо у контексті мікросервісних архітектур. Проведене дослідження дозволило не лише оцінити ефективність різних оркестраторів контейнеризованих середовищ, а й виявити конкретні особливості їхнього функціонування у критичних умовах. Аналіз механізмів відновлення після збоїв, авто-масштабування та балансування навантаження дав змогу визначити технологічні рішення, які забезпечують найкращі показники продуктивності та стійкості.

Результати тестувань демонструють, що ефективність механізмів авто-масштабування та самовідновлення суттєво впливає на стабільність роботи мікросервісних систем. Використання платформ з розвиненою інфраструктурою управління навантаженням дозволяє не лише зменшити тривалість простоїв, а й покращити прогнозованість поведінки системи при високих навантаженнях або критичних відмовах. Для підприємств, що орієнтуються на безперервну доступність сервісів, обрані технології оркестрації можуть стати визначальним фактором при розгортанні програмних рішень. Отримані результати мають безпосередню практичну цінність для розробників, архітекторів програмного забезпечення та спеціалістів DevOps, які проєктують високонавантажені системи. Для критично важливих застосунків, де навіть короткочасний збій може призвести до фінансових втрат або порушення бізнес-процесів, необхідно обирати рішення з максимальною автоматизацією процесів відновлення. У цьому контексті досліджені платформи продемонстрували суттєву різницю у швидкості реагування на збої, що є ключовим критерієм для вибору відповідного інструменту. Практичне застосування отриманих висновків можливе у сферах, де надійність інформаційних систем має вирішальне значення. Фінансові установи, які обробляють транзакції у реальному часі, можуть використовувати результати для вибору оптимального середовища для своїх мікросервісних платформ, забезпечуючи мінімальні затримки при збої одного з компонентів. У сфері телекомунікацій, де безперервна доступність є критичним параметром, використання надійної платформи оркестрації контейнерів дозволить підтримувати стабільний рівень якості обслуговування. Системи керування інтернетом речей, які часто працюють у змінних умовах навантаження, потребують швидкого масштабування та ефективного відновлення після можливих відмов. Отримані результати підтверджують, що вибір технології оркестрації суттєво впливає на реактивність таких платформ. У сфері електронної комерції, де продуктивність напряму впливає на користувацький досвід, впровадження правильного механізму управління ресурсами дозволить мінімізувати ризики деградації продуктивності під час пікових навантажень.

Окрім безпосереднього використання у бізнес-середовищах, отримані результати можуть бути корисними для академічної спільноти та дослідників у галузі програмної інженерії. Моделювання збоїв та оцінка часу відновлення є важливими елементами при тестуванні та проєктуванні надійних систем. Проведене дослідження може слугувати основою для подальших експериментів із використанням інших технологій контейнеризації та інструментів хаос-інженерії, що дозволить поглибити розуміння механізмів високої доступності у розподілених середовищах. Таким чином, результати дослідження демонструють, що вибір оркестратора контейнеризованого середовища є визначальним фактором для забезпечення безперервної роботи мікросервісних архітектур. Використання платформ із розширеними механізмами автоматичного масштабування та самовідновлення дозволяє суттєво покращити показники доступності, що є критично важливим для широкого спектра застосунків у реальних умовах експлуатації.

Література

1. Baresi, L., Quattrocchi, G., Rasi, N. A qualitative and quantitative analysis of container engines // Journal of Systems and Software. – 2024. – Vol. 210. – Art. no. 111965. – Available from: https://doi.org/10.1016/j.jss.2024.111965.

2. Turin, G., Borgarelli, A., Tapia Tarifa, S. L. Predicting resource consumption of Kubernetes container systems using resource models // Journal of Systems and Software. – 2023. – Vol. 203. – Art. no. 111750. – Sep. – Available from: https://doi.org/10.1016/j.jss.2023.111750.

3. Santos, J., Reppas, E., Wauters, T., Volckaert, B., De Turck, F. Gwydion: Efficient auto-scaling for complex containerized applications in Kubernetes through Reinforcement Learning // Journal of Network and Computer Applications. – 2025. – Vol. 234. – Feb. – Available from: https://doi.org/10.1016/j.jnca.2024.104909.

4. Ahmad, H., Treude, C., Wauters, M., Volckaert, B., De Turck, F. Towards resource-efficient reactive and proactive auto-scaling for microservice architectures // Journal of Systems and Software. – 2025. – Vol. 225. – Art. no. 112390. – Available from: https://doi.org/10.1016/j.jss.2025.112390.

5. Pallewatta, S., Kostakos, V., Buyya, R. MicroFog: A framework for scalable placement of microservices-based IoT applications in federated Fog environments // Journal of Systems and Software. – 2024. – Vol. 209. – Art. no. 111910. – Available from: https://doi.org/10.1016/j.jss.2023.111910.

6. Henning, S., Hasselbring, W. Benchmarking scalability of stream processing frameworks deployed as microservices in the cloud // Journal of Systems and Software. – 2024. – Vol. 200. – Art. no. 111879. – Feb. – Available from: https://doi.org/10.1016/j.jss.2023.111879.