Electronics and information technologies, 2025, 29, 47–56
http://publications.lnu.edu.ua/collections/index.php/electronics/index

УДК 004.03, 004.9

ЕФЕКТИВНІСТЬ ОБРОБКИ ПАРАЛЕЛЬНИХ ЗАПИТІВ У СЕРЕДОВИЩІ AMAZON WEB SERVICES З ВИКОРИСТАННЯМ КОНТЕЙНЕРІВ

Олег Сігунов*, *Лідія Демків
Львівський національний університет імені Івана Франка,
вул. Драгоманова 50, 79005 м. Львів, Україна

Сігунов, О, Демків, Л. (2025). Ефективність обробки паралельних запитів у середовищі Amazon Web Services з використанням контейнерів. *Електроніка та інформаційні технології*, 29, 47-56. https://doi.org/10.30970/eli.29.5

АНОТАЦІЯ

Передумови. У статті досліджено ефективність роботи web-застосунку, розгорнутого у хмарних сервісах Amazon веб сервіс (AWS) із використанням контейнеризації на базі AWS сервісу еластичних контейнерів (ECS) та AWS еластичного сервісу Kubernetes (EKS). Особливу увагу приділено аналізу продуктивності системи в умовах високого навантаження, характерного для онлайн-сервісів, що забезпечують відображення спортивних змагань у реальному часі. Оцінено здатність застосунку обробляти велику кількість запитів від користувачів одночасно, що є критично важливим для забезпечення коректної роботи у режимі реального часу.

Методи та інструменти. Методологія дослідження базується на навантажувальному тестуванні із застосуванням сучасних інструментів для генерації запитів та збору метрик. Проведено серію експериментів з моделювання різних сценаріїв використання, включаючи пікові навантаження, стабільний робочий режим та ситуації з різкою зміною трафіку. Аналізувалися ключові параметри продуктивності: швидкість обробки запитів сервером, рівень стабільності системи при зростанні навантаження, а також частка помилково оброблених запитів.

Результати та обговорення. Результати експериментів продемонстрували, що AWS EKS забезпечує найкраще співвідношення продуктивності, стабільності та вартості для розгортання подібних рішень. Використання Kubernetes-кластерів дозволяє гнучко масштабувати інфраструктуру, адаптуючи її під змінні навантаження без значних втрат у швидкодії. AWS ECS, хоча й забезпечує достатню продуктивність, виявився менш гнучким у порівнянні з AWS EKS, особливо у випадках динамічних змін навантаження.

Висновки. Дослідження підтвердило доцільність використання AWS EKS як основної платформи для контейнеризації веб-застосунку з високим навантаженням. Отримані результати можуть бути корисними для розробників та архітекторів хмарних рішень, які працюють із системами реального часу, що вимагають високої продуктивності, гнучкості та надійності. Окрім того, наведені дані можуть слугувати основою для подальших досліджень у сфері оптимізації хмарних ресурсів та підвищення ефективності роботи веб-сервісів.

Ключові слова: хмарні сервіси, контейнери, AWS ECS, AWS EKS, Gatling, сценарії тестування.

ВСТУП

В сучасних цифрових хмарних середовищах багато проектів працюють у режимі реального часу. Забезпечення стабільної та надійної роботи проектів вимагає ефективного використання ресурсів, які пропонують хмарні сервіси для забезпечення масштабованості та стабільності роботи проектів, а також з метою уникнення непередбачуваних витрат на інфраструктуру та обслуговування. Особливо важливою є проблема забезпечення стабільної роботи проектів у хмарних сервісах при непостійному характері споживання ресурсів з боку користувачів. В хмарних сервісах обчислювальні ресурси та дані переносяться на віддалені сервери. Для забезпечення стабільної роботи виникає потреба у виборі оптимальних параметрів хмарних сервісів та створенні ефективних систем для зберігання, оброблення та аналізу даних. Оцінюючи продуктивність та стабільність, важливо враховувати різноманітні параметри, такі як обсяг доступної пам'яті, пропускна здатність мережі та локальні затримки, особливо у контексті контейнеризації. В [1,2] проведено дослідження оптимізації витрат на AWS через розгляд моделей ціноутворення, стратегій та практичних ідеї управління ресурсами. Показано, що ретельне планування та визначення пріоритетів завдань дозволяє ефективно мінімізувати витрати часу та ресурсів. У роботі [3] розглянуто технічні аспекти розгортання, керування та масштабування контейнеризованих програм і застосунків в інфраструктурі AWS за допомогою платформи контейнеризації Docker на віртуальних машинах Amazon EC2, а також у середовищах AWS ECS і AWS EKS. В сучасній розробці та розгортанні програмного забезпечення використання контейнеризації часто залежить від конкретних потреб і масштабів проекту. В [4,5] розглянуто особливості контейнеризації та автоматизованого управління контейнерами для оптимізації вартості розгортання мікросервісів. Оскільки контейнеризація стала широко використовуваним підходом, важливо дослідити її переваги для ефективного управління та масштабування інфраструктури, В статті визначається оптимальний сервіс для контейнеризації шляхом проведення навантажувального тестування web-застосунку, розгорнутого в хмарному сервісі. Розглядаючи різні хмарні сервіси від різних постачальників, необхідно визначити оптимальне середовище та розглянути можливості розміщення системи в різних хмарних середовищах.

МЕТОДИ ТА ІНСТРУМЕНТИ

Вибір середовища розгортання для тестового веб застосунку, розробленого мовою Java та призначеного для трансляції результатів змагань у реальному часі великій кількості користувачів, потребує врахування таких аспектів, як мережева доступність і зручність масштабування. Оптимальним рішенням є запуск сервісу в хмарному середовищі, що має ряд переваг перед використанням стаціонарного сервера. Для розгортання обрано хмарні сервіси Amazon Web Services, які в порівнянні з Microsoft Azure та Google Cloud Platform (GCP) мають свої плюси та мінуси. До переваг AWS належить наявність програмного інтерфейсу взаємодії для всіх сервісів, що дозволяє керувати хмарними ресурсами автоматизовано через код, знижуючи витрати на підтримку. Серед недоліків – дещо вищі витрати в деяких сценаріях використання та складність розгортання проекту, яка вимагає глибоких знань сервісів або підтримки спеціалізованої команди.

У контексті тестового веб застосунку важливим функціоналом є забезпечення стабільного доступу до даних користувачів та отримання результатів у реальному часі. Тому конфігурація системи має бути оптимізована для швидкого відправлення даних багатьом користувачам у короткі проміжки часу. Після вибору хмарного сервісу проведено тестування конфігурацій, розгорнутих в AWS EKS та AWS ECS, щоб обрати оптимальний сервіс для стабільної роботи веб застосунку.

Мова програмування Scala є новим поколінням віртуальної машини Java (JVM – Java Virtual Machine). Scala використовує сучасну та ефективну модель акторів (в даному контексті термін «актори» позначає універсальні примітиви паралельного виконання), що дозволяє розробникам визначати кожен об'єкт як окрему сутність з властивостями, поведінкою та станом. Це спрощує взаємодію між потоками, забезпечуючи більший контроль над даними та підвищуючи ефективність процесів.

Gatling – це відкрите програмне забезпечення для тестування роботи програми, написане мовою Scala. Воно дозволяє обробляти великий об'єм трафіку на одному комп'ютері. Під час тестування досліджується, як система справляється з різними навантаженнями. Багато програм добре працюють тільки при невеликому числі користувачів, але при стрімкому збільшенні кількості користувачів можуть виникати проблеми з роботою системи. Тестування дозволяє виявити проблемні місця та виправити їх для підвищення продуктивності.

Існують кілька типів тестування:

- тестування навантаженням (Load Testing): тестування системи при плановій кількості користувачів та/або трафіку;
- стрес-тестування (Stress Testing): тестування системи за постійного збільшення навантаження для виявлення точки відмови;
- тестування стабільності (Soak Testing): тестування зі стабільним навантаженням протягом тривалого часу.

Gatling має різні сценарії тестування, але для нашої задачі найкраще підходить стратегія constantConcurrentUsers яка підтримує певну кількість користувачів (симулює користувачів, які спостерігають за змаганнями).

Основні параметри оцінки продуктивності системи включають час відгуку транзакції, пропускну здатність, відсоток успішних запитів. Gatling надає інформативні звіти для полегшення аналізу результатів.

Amazon ECS дозволяє керувати контейнерами у хмарному середовищі AWS. Amazon EKS – це керована служба кластерів Kubernetes від AWS, яка дозволяє легко запускати, керувати і масштабувати контейнеризовані додатки в хмарному середовищі.

ECS та EKS підтримують два типи розгортання: використання віртуальних машин у хмарному середовищі (інстанси) та повністю автоматизований сервіс Fargate, який керує вибором, масштабуванням та управлінням інстансами. Для тестування web-застосунку використано варіант EC2 для того, щоб моделювання відбувалось за однакових ресурсів.

AWS Elastic Load Balancer (ELB) розподіляє вхідний трафік між різними компонентами системи, підвищуючи її надійність та масштабованість. Amazon Elastic Container Registry (ECR) дозволяє керувати реєстрацією контейнерів у хмарному середовищі AWS, забезпечуючи зберігання, управління та розповсюдження контейнерів. AWS Relational Database Service (RDS) надає можливість розгортання реляційних баз даних у хмарному середовищі AWS, включаючи вибір типу бази даних та її двигуна для оптимальної продуктивності та надійності.

Web-застосунок написано за допомогою Java програмного каркасу Spring, а проект для тестування використовує Scala. Під час розробки використано конструктор залежностей (Maven) та середовище розробки IntelliJ IDEA. Програма складається з двох незалежних між собою мікросервісів: один з них відповідає за реєстрацію та менеджмент користувачів, а інший за передачу даних в реальному часі. Маршрутизація вхідних запитів та спілкування між сервісами налагоджено в межах сервісів. Для забезпечення максимальної близькості до реальних умов, застосунок було розгорнуто як ізольоване середовище - докер контейнер, що містить застосунок разом із усіма необхідними залежностями (бібліотеками, конфігураціями тощо)

Використання контейнеризації має ряд переваг. По-перше, контейнеризація забезпечує консистентність середовища розробки, тестування і виробництва, що

дозволяє уникнути проблем, пов'язаних з різницею між конфігураціями. Крім того, контейнери дозволяють ефективно використовувати ресурси сервера, оскільки вони можуть бути миттєво розгорнуті та знищені за потреби. Друга важлива перевага полягає в легкості перенесення додатків між різними середовищами, що спрощує розгортання та масштабування додатків. Контейнеризація також підвищує безпеку, оскільки вона ізолює додатки один від одного і дозволяє керувати доступом до ресурсів. Використання контейнеризації сприяє швидкому розгортанню нових версій додатків і має позитивний вплив на розробку та впровадження нових функцій шляхом зменшення часу, потрібного для тестування та перевірки змін.

Зображена на рис.1 процедура розгортання застосунку для тестування у хмарному середовищі включає декілька кроків. Для застосунку створеного з використанням Spring Boot Application здійснюється побудова Docker-образу, який може бути завантажений до реєстру (наприклад, Docker Hub або AWS ECR). Після завантаження образу до реєстру для нього буде згенеровано публічне посилання, яке необхідно вказати під час налаштування конфігурацій кластерів, таких як Amazon ECS або Amazon EKS. Це дозволить кластерам коректно використовувати цей образ під час запуску контейнерів, оскільки один і той самий образ може мати декілька версій від кількох різних дистриб'юторів.

Архітектура системи для тестування, розгорнутої за допомогою ECS/EKS, зображена на схемі рис. 2. Основні компоненти включають Application Load Balancer (ALB), який розподіляє запити між контейнерами в кластері. Кластер складається з набору інстансів, на яких розгорнуті сервіси (Spring Boot application), створені з образів за допомогою сервісу Elastic Container Registry (ECR) для зберігання, управління та розповсюдження контейнерних образів Docker у хмарі AWS, використаних для конфігурації кластера. Також система включає базу даних завдяки використанню керованого сервісу реляційної бази даних від AWS, що зберігає дані, які використовуються застосунком. Ця архітектура забезпечує надійну та масштабовану інфраструктуру для тестування додатків, що дозволяє ефективно розподіляти навантаження та працювати з необхідними даними.

Необхідно порівняти продуктивність застосунку, який запущений в ECS або EKS при умові рівних ресурсів (однаковий тип інстансу, однаковий програмний код) та визначити оптимальну конфігурацію інстансів, яка буде одночасно витримувати велику кількість користувачів. В Amazon ECS та Amazon EKS можна використовувати

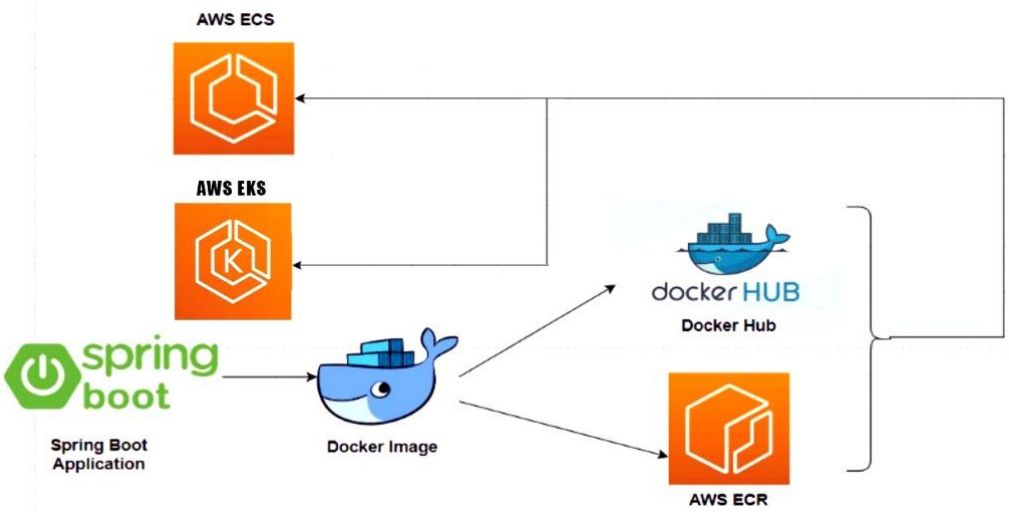


Рис. 1. Схема розгортання web-застосунку для тестування за допомогою ECS/EKS.
Fig. 1. Deployment scheme of the web application for testing using ECS/EKS.

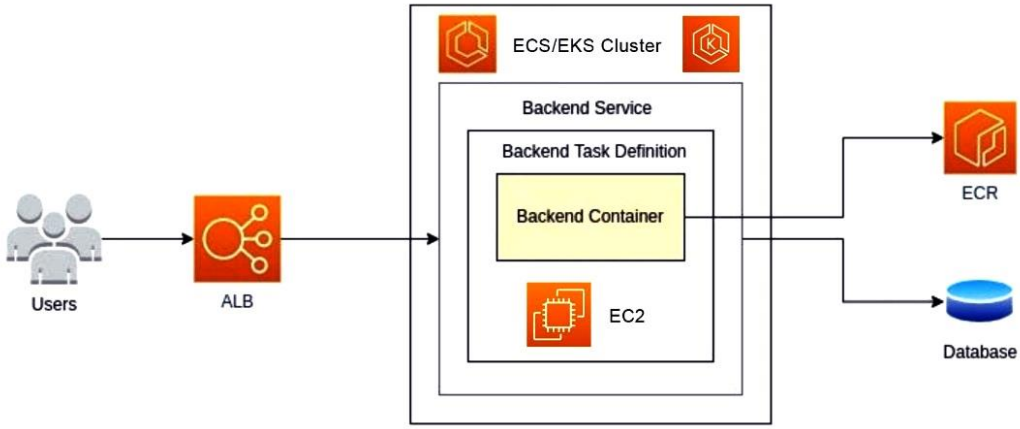


Рис. 2. Архітектура системи для тестування розгорнутої за допомогою ECS/EKS.
Fig. 2. System architecture for testing deployed using ECS/EKS.

EC2 сервери, які представлені багатьма типами з різними ключовими характеристиками. Сервери призначені для вирішення різноманітних задач у різному ціновому діапазоні. Вони дозволяють обрати найбільш оптимальні ресурси для конкретних потреб. Для дослідження використано інстанси t2.small, які належать до групи загального застосування. Інстанси забезпечують баланс між продуктивністю та вартістю, мають одноядерний віртуальний центральний процесор (1 vCPU) та 2 ГБ оперативної пам'яті, що робить їх ідеальним вибором для широкого спектру завдань, включаючи розробку, тестування від невеликих до середніх систем в продуктивних середовищах.

MySQL база даних web-застосунку розміщена у хмарному сервісі за допомогою AWS RDS. Для бази даних обрано сервер db.r5.large, який базується на 2vCPU 16 ОЗУ та 5 гбіт/с мережі, що для задач web-застосунку дуже надлишково, але це необхідно для того, щоб впевнитись, що тестування перевіряє саме сервери програми, а не базу даних.

РЕЗУЛЬТАТИ ТА ОБГОВОРЕННЯ

Для тестування розроблено сценарій, який є наближеним до реальної моделі роботи застосунку. Користувачі відправляють запит на сервер для отримання даних турнірних таблиць у реальному часі. Кількість одночасних користувачів варіювалась для оцінки впливу навантаження на систему. Тести проведені для 20, 30, 50 та 80 користувачів, які одночасно надсилають запит на сервер за принципом повторного надсилання запиту після отримання відповіді на попередній. Цей підхід дозволив перевірити не лише працездатність системи в умовах реального завантаження, але й з'ясувати, як вона поводиться за різних умов, що важливо для оцінки її ефективності та надійності. Тривалість кожного проведеного тестування дорівнювала 5 хв. Така тривалість дає змогу показати як поводить себе застосунок під постійним навантаженням, оскільки цього часу достатньо, щоб виявити вузькі місця системи. Такі тести є об'єктивними і для триваліших проміжків часу.

Зведені результати тестування для всіх рівнів навантаження представлено у таблицях 1 та 2 (для ECS та EKS, відповідно). У лівій колонці таблиці представлено параметри: мінімальний, максимальний та середній час відповідей сервера у мілісекундах, кількість успішних чи невдалих відповідей сервера.

З даних таблиць бачимо, що однаковим для EKS та ECS є те, що зі зростанням кількості користувачів росте також середній час відповіді, при чому росте майже лінійно. З результатів тестування для EKS випливає, що середній час обробки запиту

Таблиця 1. **Результати тестування для ECS**
Table 1. **Testing results for ECS**

	Кількість одночасних користувачів			
	20	30	50	80
Мінімальний час відповіді, мс	51	51	52	51
Максимальний час відповіді, мс	20033	20030	20030	20037
Середній час відповіді, мс	1260	1876	3470	4365
Кількість успішних запитів	4579	4580	4146	4655
Кількість не успішних запитів	186	237	185	872
Відсоток не успішних запитів	4%	5%	4%	16%

Таблиця 2. **Результати тестування для EKS**
Table 2. **Testing results for EKS.**

	Кількість одночасних користувачів			
	20	30	50	80
Мінімальний час відповіді, мс	51	51	51	51
Максимальний час відповіді, мс	3103	67455	81087	87388
Середній час відповіді, мс	79	84	187	225
Кількість успішних запитів	72574	74633	77066	90045
Кількість не успішних запитів	0	0	5	42
Відсоток не успішних запитів	0%	0%	0%	0%

значно менший (<100 ms), ніж для ECS. Для ECS час обробки запиту завжди більше 1 секунди, через що кількість опрацьованих запитів за однаковий час відрізняється дуже суттєво: більше 70000 тисяч для EKS та 5 тисяч для ECS.

Цікавим спостереженням є те, що деякі запити для EKS займали понад хвилину часу, однак все ще були успішно виконаними. В той же час для ECS не має запитів, які тривали більше 20 секунд, однак кількість не виконаних запитів значно більша. Ці результати пов'язані з тим, як система Kubernetes керує вхідними запитами. В Kubernetes запити можуть стояти в черзі довше, однак зрештою вони будуть опрацьовані. На всіх рівнях тестування в EKS менше 1% похибок для всіх рівнів навантаження, на відміну від ECS, де серед розглянутих на максимальному рівні запитів - 16% не є успішними.

Gatling генерує звіти в графічній формі. Нижче наведені звіти з тестування ECS та EKS при однакових навантаженнях у 80 користувачів на секунду.

З діаграм на рис.3 видно, що в обох випадках найбільша частина запитів оброблялася швидше за 800 мс. Однак, для ECS суттєва частка запитів оброблялася в діапазонах 800-1200 мс та понад 1200 мс. Також велика кількість запитів є неуспішною. В той час, як для EKS, дуже мала частка запитів оброблялася в часовому діапазоні 800-1200 мс, а кількість невдалих запитів є несуттєвою (менше 0,5%).

З графіків на рис.4 видно, що система на ECS більшу частину часу підтримувала стабільну кількість користувачів, близько 150. Проте, оскільки це є фактичним максимумом для цієї системи, час відповіді на запити змінювався. На графіку видно

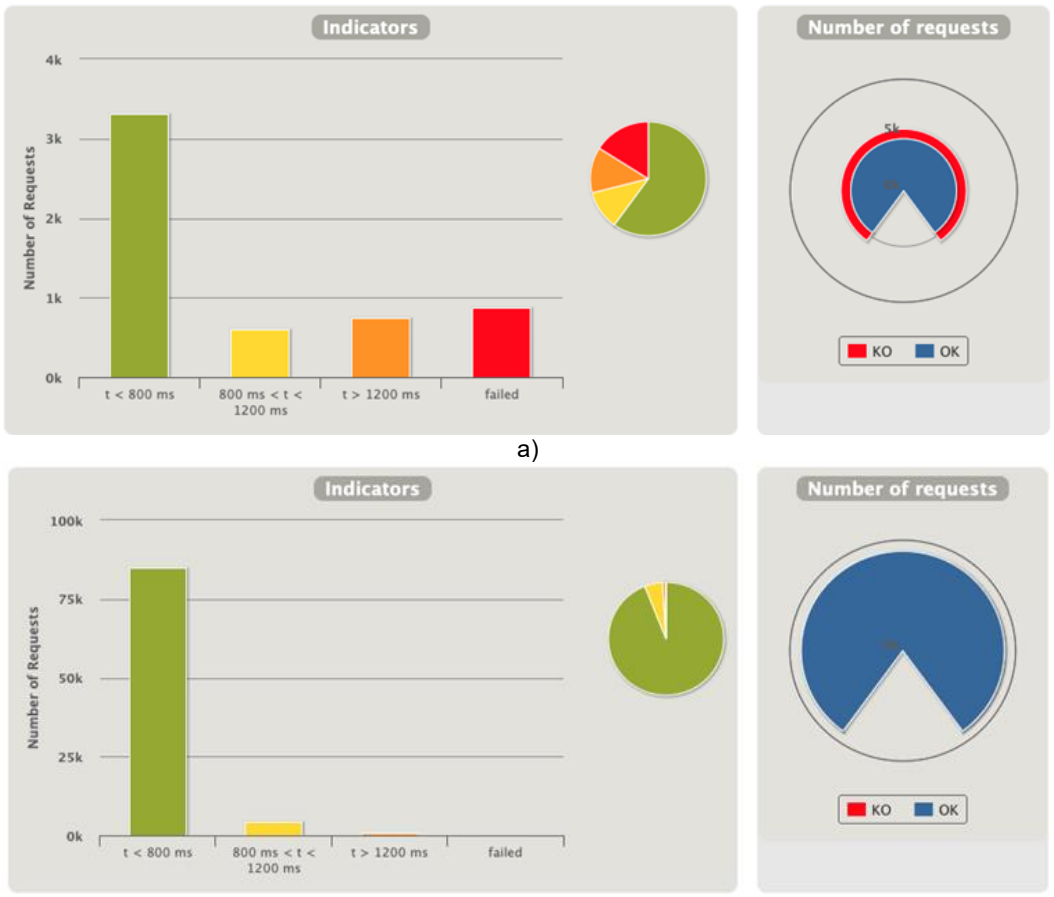


Рис. 3. Діаграми розподілу часу відповіді на запит для а) ECS та б) EKS.
Fig. 3. Response time distribution diagrams for (a) ECS and (б) EKS.

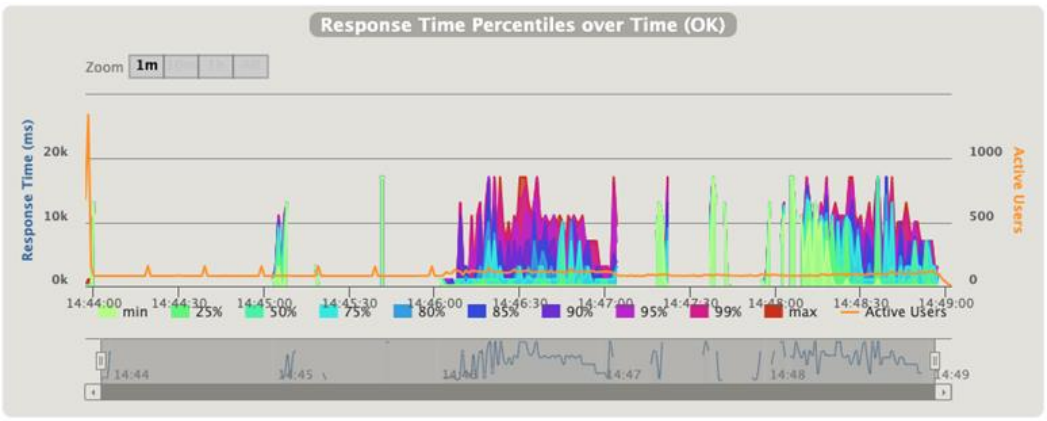
кілька стрибків, які показують періоди, коли запити оброблялися близько 20 секунд і відповідно потрапляли у 90-й процентиль.

У той же час система на EKS приймала значно більшу кількість запитів і, відповідно, більшу кількість користувачів з деякими піками понад 1000 користувачів. Попри це, вона забезпечувала дуже стабільний час відповіді, який потрапляв у межі 50-го процентиля. Це свідчить про високу ефективність і продуктивність системи на EKS у порівнянні з ECS.

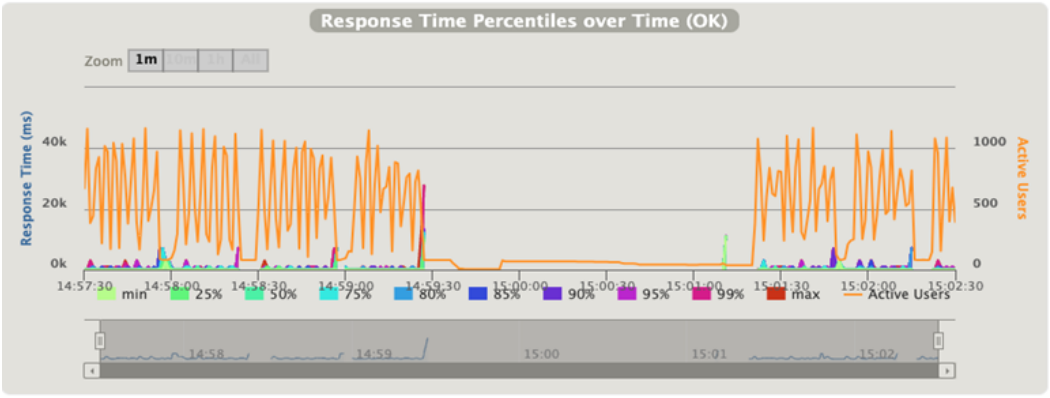
Детальний аналіз показує, що ECS, при навантаженні близько 150 користувачів, досягала свого максимального потенціалу, що призводило до значних затримок у відповіді на запити. Це особливо помітно під час пікових навантажень, коли час відповіді перевищував 20 секунд.

На відміну від ECS, система на EKS демонструвала високу стійкість і ефективність при значно більшому навантаженні. Навіть при пікових навантаженнях коли в системі було понад 1000 користувачів, EKS продовжувала забезпечувати стабільний і швидкий час відповіді. Це підкреслює її здатність обробляти великі обсяги запитів без суттєвих затримок, що є важливим показником продуктивності та надійності системи.

З даних таблиць видно, що кількість подій за секунду, які обробляє ECS, значно менша за кількість подій, які може обробити EKS. Зокрема, ECS обробляє лише 18 подій на секунду, тоді як EKS обробляє 299 подій на секунду.



а)



б)

Рис. 4. Графік відношення кількості активних користувачів до часу відповіді для а) ECS та б) EKS
Fig. 4. Graph of the relationship between the number of active users and response time for (a) ECS and (б) EKS.

Час відповідей для 50-го та 75-го процентилів зростає дуже незначно для EKS (73 мс та 82 мс відповідно). Проте для ECS цей розрив є величезним: 63 мс для 50-го процентиля та 3203 мс для 75-го процентиля. Також 95- й та 99-й процентилі для ECS майже не відрізняються і дуже близькі до максимального часу відповіді. Це свідчить про те, що система на ECS вже працює за межами своєї продуктивності, а кількість помилок перевищує допустиму норму. В той самий час, 95-й та 99-й процентилі для EKS все ще плавно зростають і знаходяться досить далеко від максимального часу відповіді. З рис.5 видно відмінності в типах помилок для обох систем. У випадку з ECS через надмірну кількість запитів система перестає їх приймати, що призводить до виникнення помилки ConnectionTimeOutException. У випадку з EKS помилка має дещо інший характер. Вона свідчить про те, що сервіс не встиг віддати відповідь протягом однієї хвилини, через що отримуємо помилку RequestTimeoutException.

ВИСНОВОК

Створено веб-застосунок, а також протестовано його швидкодію та стабільність роботи з урахуванням різних засобів для контейнеризації за допомогою AWS ECS та AWS EKS. Виявлено, що найбільшою різницею між способами контейнеризації є механізм обробки вхідних запитів. З порівняння видно, що система на ECS показала себе значно стабільнішою у підтримці однакової кількості користувачів без значних

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	5527	4655	872	16%	18.062	51	63	3203	20022	20030	20037	4365	7421
Get Info	5527	4655	872	16%	18.062	51	63	3203	20022	20030	20037	4365	7421

ERRORS

Error	Count	Percentage
i.n.c.ConnectTimeoutException: connection timed out: ecs3-2009852195.eu-central-1.elb.amazonaws.com/3.64.203.82:8080	442	50.688 %
i.n.c.ConnectTimeoutException: connection timed out: ecs3-2009852195.eu-central-1.elb.amazonaws.com/52.58.41.221:8080	430	49.312 %

а)

STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	90087	90045	42	0%	299.292	51	73	82	1063	1131	87388	225	2030
Get Info	90087	90045	42	0%	299.292	51	73	82	1063	1131	87388	225	2030

ERRORS

Error	Count	Percentage
i.g.h.c.i.RequestTimeoutException: Request timeout to a032ce26345cb4ea5b299d26356bb7d5-184144236.eu-central-1.elb.amazonaws.com/3.67.200.26:80 after 60000 ms	24	57.143 %
i.g.h.c.i.RequestTimeoutException: Request timeout to a032ce26345cb4ea5b299d26356bb7d5-184144236.eu-central-1.elb.amazonaws.com/3.66.208.62:80 after 60000 ms	13	30.952 %
i.g.h.c.i.RequestTimeoutException: Request timeout after 60000 ms	5	11.905 %

б)

Рис. 5. Детальні результати тестування для а) ECS та б) EKS
Fig. 5. Detailed testing results for (a) ECS and (б) EKS.

стрибків протягом роботи. ECS може мати перевагу в обробці запитів, які вимагають тривалих обчислень. Однак, EKS, хоч і обробляє запити значно швидше, однак приймає їх хвилями, що приводить до значних стрибків у кількості активних користувачів. Водночас, кількість неуспішних запитів, які оброблялися значно довше за середній час на стороні EKS, була значно меншою порівняно з ECS. При обробці багатьох одночасних запитів за допомогою ECS відбувається сповільнення обробки всіх запитів на певний час. Це є критичним недоліком для розглядуваного класу задач, зокрема для відображення даних про змагання. Таким чином, з порівняння видно, що при використанні однакових ресурсів, EKS підходить значно краще для обробки великої кількості простих запитів. Після проведених тестів визначено, що найкраще для реалізації даного проекту підходить AWS EKS.

АВТОРСЬКИЙ ВНЕСОК

Концептуалізація [О.С., Л.Д.]; методика [О.С., Л.Д.]; дослідження [О.С., Л.Д.]; написання початкової версії статті [О.С.]; перегляд та редагування [Л.Д.]; підготовка графічного матеріалу [О.С.]; загальне керівництво [Л.Д.].

Усі автори ознайомилися та погодилися з опублікованою версією статті.

ЛІТЕРАТУРА

[1] Bhupesh Kumar Dewangan S,Amit Agarwal, Tanupriya Choudhury, Ashutosh Pasricha Cloud Resource Optimization System Based on Time and Cost // International Journal of Mathematical, Engineering and Management SciencesVol. 5, No. 4, 758-768, 2020. https://doi.org/10.33889/IJMEMS.2020.5.4.060

[2] Sumanth Tatineni Cost Optimization Strategies For Navigating The Economics Of Aws Cloud Services //International journal of advanced research in engineering & technology, 10(6):827-842, 2019. https://iaeme.com/Home/article_id/IJARET_10_06_084
[3] Shimon Ifrah Deploy Containers on AWS: With EC2, ECS, and EKS // ISBN: 978-1-4842-5100-3, 2019. https://doi.org/10.1007/978-1-4842-5101-0
[4] Li, W., Li, X., Chen, L. Pattern learning for scheduling microservice workflow to cloud containers. // Int. J. Mach. Learn. & Cyber., 2024. https://doi.org/10.1007/s13042-024-02115-5
[5] Dasith Edirisinghe, Kavinda Rajapakse, Pasindu Abeysinghe, Sunimal Rathnayake Cost-Optimal Microservices Deployment with Cluster Autoscaling and Spot Pricing // arXiv:2405.12311. 2024. https://doi.org/10.48550/arXiv.2405.12311

RESEARCH ON THE EFFICIENCY OF PROCESSING PARALLEL REQUESTS BY AMAZON WEB SERVICES USING CONTAINERS

Oleg Sigunov, Lidiia Demkiv
Ivan Franko National University of Lviv, 50 Dragomanov St., Lviv 79005, Ukraine

ABSTRACT

Background. The article explores the efficiency of a web application deployed in AWS cloud services using containerization based on AWS ECS and AWS EKS. Particular attention is paid to analyzing system performance under high load conditions typical for online services that provide real-time sports competition updates. The ability of the application to handle a large number of user requests simultaneously is assessed, which is critically important for ensuring proper operation in real-time mode.

Methods and tools. The research methodology is based on load testing using modern tools for generating requests and collecting metrics. A series of experiments were conducted to model different usage scenarios, including peak loads, stable operating conditions, and situations with sudden traffic changes. Key performance parameters were analyzed: server request processing speed, system stability under increasing load, and the proportion of unsuccessfully processed requests.

Results and discussion. The experimental results demonstrated that AWS EKS provides the best ratio of performance, stability, and cost for deploying similar solutions. The use of Kubernetes clusters allows for flexible infrastructure scaling, adapting to variable loads without significant losses in speed. While AWS ECS ensures sufficient performance, it proved to be less flexible compared to AWS EKS, particularly in cases of dynamic load changes.

Conclusions. Thus, the study confirmed the feasibility of using AWS EKS as the primary platform for containerizing a high-load web application. The obtained results may be useful for developers and cloud solution architects working with real-time systems requiring high performance, flexibility, and reliability. Additionally, the presented data may serve as a basis for further research in optimizing cloud resources and improving the efficiency of web services.

Keywords: cloud services, containers, instances, AWS ECS, AWS EKS, Gatling, testing scenarios.

Received / Одержано	Revision / Доопрацьовано	Accepted / Прийнято	Published / Опубліковано
09 October, 2024	15 January, 2025	20 January ,2025	29 March, 2025