ISSN 2224-087X. Електроніка та інформаційні технології. 2022. Випуск 20. С. 30–41
Electronics and information technologies. 2022. Issue 20. P. 30–41

УДК 004.03, 004.9 DOI: https://doi.org/10.30970/eli.20.4

ДОСЛІДЖЕННЯ ШВИДКОДІЇ ОБРОБКИ ПАРАЛЕЛЬНИХ ЗАПИТІВ ХМАРНИМИ СЕРВІСАМИ AWS

О. Сігунов, Л. Демків

Львівський національний університет імені Івана Франка,
Драгоманова, 50, 79005 Львів, Україна
lidia.demkiv@gmail.com

У статті представлено результати тестування роботи web-застосунку для проведення спортивних змагань у хмарних сервісах на різних типах серверів однакової цінової категорії за сценаріїв найбільш наближених до реальних, зокрема одночасного отримання даних багатьма користувачами. Визначено параметри: стабільність застосунку на певному типі сервера, швидкість опрацювання запиту сервером за різної кількості користувачів та помилки, які виникають під час роботи застосунку. Встановлено найбільш оптимальний для даного застосунку вид серверів, який дає максимальне співвідношення продуктивності та стабільності до ціни для такого типу задач. Визначено, що найкраще для реалізації даного проекту підходять інстанси типів t3.large та r4.large.

Ключові слова: хмарні сервіси, інстанси, AWS EC2, Gatling, типи навантажень, сценарії тестування.

Вступ

Актуальність задачі дослідження функціональності застосунків у хмарних сервісах зумовлена необхідністю забезпечити стабільну роботу сервісів, які зберігають, обробляють, аналізують дані, при непостійному характері споживання ресурсів хмарних сервісів з боку споживачів.

Хмарні сервіси, що дозволяють перенести обчислювальні ресурси й дані на віддалені інтернет-сервери, в останні роки стали одним з основних трендів розвитку IT-технологій. Для вирішення задачі зберігання даних та їх обчислень необхідно підібрати відповідні хмарні сервіси та встановити можливі структури систем, які дадуть максимальне співвідношення продуктивності та стабільності до ціни для визначеного класу задач. Основними вартими оцінки параметрами, є стабільність, швидкість, максимальна кількість користувачів та ціна утримання такої системи. Оскільки для різних типів застосунків необхідно звертати увагу на різні ключові параметри, такі як кількість одночасно працюючих процесорів, доступні об’єми пам’яті, пропускна здатність мережевого порту, локальна затримка в залежності від географічного розташування серверів хмарного сервісу, то доцільно робити порівняння на окремих застосунках. Хмарні сервіси зараз надаються багатьма провайдерами, тож необхідно також встановити відмінності між ними, а також дослідити можливість часткового розміщення системи в різних хмарних середовищах.

Обгрунтування та вибір середовища розгортання web-застосунку

Web-застосунок для проведення змагань, написаний мовою програмування java, має ряд особливостей функціонування, які потрібно врахувати перед запуском проекту. Оптимальним рішенням є запуск сервісу в хмарному середовищі, тому що хмарне середовище має ряд переваг перед використанням стаціонарного сервера. Оскільки система розроблена для проведення змагань, а вони проводяться не надто часто, то при використанні сервера отримуємо проблему, пов’язану з тим, що сервер розрахований на навантаження під час змагань, буде простоювати без роботи більшу частину часу, а сервер розрахований на період часу поза змаганнями, не витримає навантаження під час змагань. Потрібно знайти рішення, яке за мінімальні кошти дасть максимум переваг. Тому для розгортання web-застосунку були обрані хмарні сервіси від Amazon, а саме AWS(Amazon Web Services). У порівнянні з Microsoft Azure та GCP(Google Cloud Platform) у AWS є ряд плюсів та мінусів. Наприклад, до плюсів відносимо те, що всі сервіси AWS мають API, які дозволяють маніпулювати кожним сервісом окремо, що в свою чергу дозволяє сильно знизити витрати на підтримку розгорнутих в хмарі систем. До мінусів AWS можна віднести те, що він є дещо дорожчим в багатьох сценаріях використання, особливо в області зберігання великих об'ємів даних. Тому багато компаній використовують власні локальні сховища для зберігання даних. Також суттєвим мінусом даних хмарних сервісів є складність розгортання проекту, оскільки, щоб розгорнути повноцінну, захищену та стабільну систему, треба володіти багатьма сервісами на досить високому рівні, або заручитись підтримкою AWS партнера, який налаштує та буде підтримувати систему. Оскільки система для змагань не працює з графічними чи зашифрованими даними, то їй не потрібне велике сховище і можна обмежитись однією або кількома реляційними базами даних, які працюватимуть в кластері, що робить в свою чергу ціну на використання AWS досить демократичною. Після вибору хмарного сервісу визначаємо, яку конфігурацію має мати система, щоб навіть за значного напливу користувачів, вона працювала стабільно та не приносила користувачам незручностей. Для даної системи важливими є два класи задач, перший - це стабільний доступ до даних користувачів для їх реєстрації та менеджменту, а другий - це отримання в реальному часі результатів змагань користувачами, що переглядають поточний прогрес змагань. Тобто конфігурація має бути оптимізована під відправлення даних багатьом користувачам в короткі проміжки часу. В роботі створено декілька різних конфігурацій, які включають різні типи серверів, і проведено тестування цих конфігурацій для вибору оптимальних для цього класу задач.

Перелік та особливості використаних технологій

Мова програмування Scala представляє собою нове покоління віртуальної машини Java (JVM). Scala використовує сучасну ефективну модель акторів, яка дозволяє розробнику визначати кожен об’єкт як окрему сутність з властивостями, поведінкою та індексом. Таким чином Scala спрощує взаємодію між потоками, збільшує контроль над даними, тим самим збільшуючи ефективність всього процесу.

Gatling - це інструмент для тестування роботи програми з відкритим кодом, який повністю написаний Scala і дозволяє обробляти великий об'єм трафіку на одному комп'ютері. В процесі тестування роботи програми досліджуємо як система справляється з різною кількістю трафіку. Багато додатків працюють добре лише тоді, коли активна невелика кількість користувачів. Але коли кількість користувачів стрімко зростає, можуть виникати проблеми з роботою певних частин або ж усієї системи в

цілому. Тестування роботи програми є способом виявити та виправити проблемні місця в системі для того щоб збільшити об'єм трафіку до необхідного рівня. Існує декілька типів тестування роботи програми:

- Тестування навантаженням (Load Testing) - тестування системи на запланованому числі користувачів та/або об’ємі трафіку.
- Стрес тестування (Stress Testing) - тестування системи за постійного збільшення навантаження на систему за рахунок збільшення числа користувачів та/або трафіку, щоб знайти точку відмови системи.
- Тестування стабільності (Soak Testing) - тестування з стабільним рівнем навантаження на систему протягом більш тривалого проміжку часу.

Gatling містить всі основні сценарії навантажень, серед яких:

- atOnceUsers - збільшує кількість користувачів за один раз
- rampUsers - поступово збільшує кількість користувачів протягом певного проміжку часу
- constantUsersPerSec - закидає сталу кількість користувачів в секунду
- rampUsersPerSec - що секунди збільшує кількість користувачів
- constantConcurrentUsers - підтримує кількість користувачів на одному рівні

Основні параметри які можна відслідковувати та від яких можна відштовхуватись під час оцінювання системи це:

- Час відгуку транзакції (Transaction Response Time) - кількість часу, яка необхідна серверу, щоб відповісти на запит
- Пропускна здатність (Throughput) - кількість транзакцій, які можуть бути оброблені за певний проміжок часу
- Помилки (Errors) - помилки які можуть з'являтися протягом тестування.

Відсутність візуального інтерфейсу під час роботи з Gatling компенсується інформативним оформленням звітності, що полегшує аналіз результатів тестування.

Amazon Elastic Compute Cloud (Amazon EC2) - це сервіс, який надає обчислювальні потужності завдяки віртуальним/фізичним серверам з попередньо створеними образами операційних систем.

EC2 включає в себе багато типів серверів (інстансів), призначених для роботи з різними задачами.

Вони поділяються на:

1. General Purpose - збалансовані рішення.

2. Compute Optimised - рішення для інтенсивних обчислень та активної роботи з базами даних.

3. Memory Optimised - рішення для задач, що споживають багато оперативної пам’яті.

4. Accelerated Computing - рішення для специфічних обчислень, таких як навчання нейромереж та робота з графікою.

5. Storage Optimised - рішення для програм, які працюють з сховищем, жорстким диском, сховищами файлів.

AWS EC2 використовує AMI (Amazon Machine Image) - образ операційної системи, яку буде застосовувати інстанс EC2. Дуже багато різновидів систем представлено самим Amazon, однак користувачі можуть створювати власні AMI для конкретного типу задач. Представлені як чисті версії ОС, так і з попередньо встановленими пакетами програм, утиліт, драйверів. Важливою особливістю використання EC2 на постійній основі є

наявність декількох планів оплати, серед яких є попередній резерв ресурсів, на вимогу, динамічна оплата залежно від навантаженості та попиту.

AWS Elastic Load Balancer (ELB) - сервіс, який розподіляє вхідний трафік. Він може розподіляти трафік не просто між однотипними сутностями, а може працювати з різними сутностями одночасно, наприклад його можна підключити одночасно до EC2, Lamba, ECS, ASG. У випадку підключення до кластера обчислювальних інстансів EC2 при виході з ладу одного з інстансів за рахунок нього можна перенаправити трафік на інші інстанси, або підняти новий інстанс який буде заміною старому і скерувати трафік на нього, що робить його одним з ключових елементів в побудові системи.

Існує 3 типи інстансів:

Classic Load Balancer - перший балансувальник від AWS підтримує HTTP, HTTPS, TCP, SSL. Він забезпечує базове балансування навантаження між декількома інстансами EC2 і працює як на рівні запитів, так і на рівні з'єднання. Вважається застарілим і не рекомендованим до використання, хоча ніхто не забороняє його створити.

Network Load Balancer - підходить для високого навантаження, працює на рівні з'єднання, підтримує TCP, UDP та TLS. Здатний обробляти мільйони запитів при мінімальних затримках, а також оптимізований для моделей програм де може бути різка зміна трафіку.

Application Load Balancer - працює на рівні запитів, вміє працювати з Lambda, підтримує HTTP та HTTPS. Забезпечує розширену маршрутизацію на рівні запитів, орієнтований на додатки, побудовані згідно сучасних концепцій архітектури, таких як мікросервіси та контейнери. Керує трафіком базуючись на вмісті запиту. Є прямою заміною Classic Load Balancer, оскільки маршрутизація на рівні запиту(HTTP/HTTPS) є значно більш поширеною ніж маршрутизація на рівні з'єднання(TCP/UDP).

ELB можна моніторити в реальному часі завдяки AWS CloudWatch, який дає можливість відслідковувати в якому стані зараз система, та які ресурси вона використовує.

AWS Relational Database Service (RDS) є одним з багаточисельних сервісів Amazon AWS, який орієнтований на розгортку реляційних баз даних в хмарі, де можна обрати все: від двигуна бази даних (DBD, InnoDB, Amazon Aurora) і до типу самої бази даних (MySQL/PostgreSQL). Зокрема, якщо використовувати MySQL з двигуном Aurora, то така комбінація працює до 5 разів швидше ніж MySQL. Крім того, перевагою Aurora є можливість постійного резервного копіювання.

Структура проекту та його розміщення у хмарному сховищі

Web-застосунок для проведення змагань написано за допомогою Java фреймворку Spring, а проект для тестування використовує Scala. Також використано конструктор залежностей (Maven) та середовище розробки IntelliJ IDEA. Програма складається фактично з двох незалежних між собою мікросервісів, один з яких відповідає за реєстрацію та менеджмент користувачів, а інший за передачу даних в реальному часі, а маршрутизація вхідних запитів та спілкування між сервісами налагоджено в межах сервісів (рис.1).

Одним з завдань роботи є пошук оптимальної за ціною конфігурації інстансів, яка буде одночасно витримувати велику кількість користувачів. Оскільки EC2 має досить

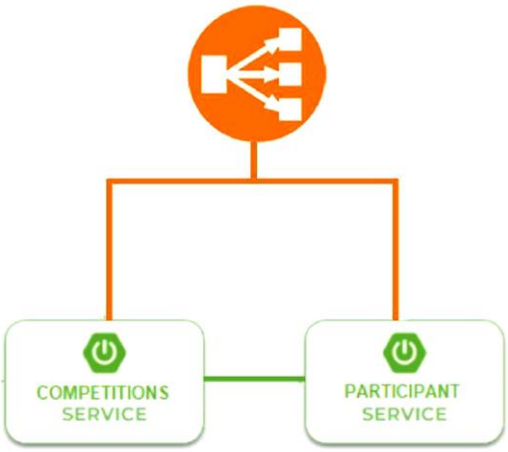

Рис. 1. Схема архітектури web-застосунку для проведення змагань.

багато типів інстансів, призначених для вирішення різноманітних задач у різному ціновому діапазоні, то для об'єктивності оцінювання всі інстанси будуть розглядатись з урахуванням архітектури програми для ведення змагань, сервіси якої виглядатимуть як показано на рис. 2.

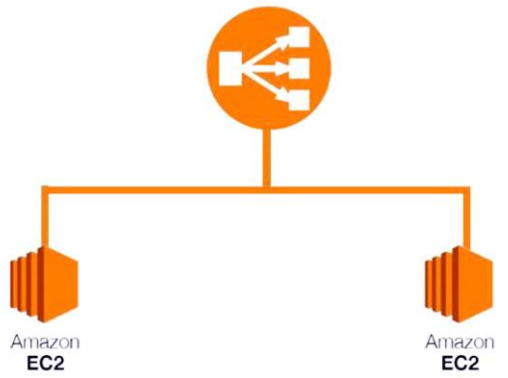

Рис.2. Архітектура хмарних сервісів для web-застосунку.

Така схема є оптимальною для структури програми представленій на рис.1.

Список інстансів для яких проведено тестування

t2.micro - класичний, найбільш поширений тип для невеликих аплікацій, який базується на на x86 архітектурі, належить до лінійки збалансованих інстансів

t3.large - схожий до t2, але значно потужніший.

c4.large - інстанс розрахований на обчислення, та роботу з базою даних

r4.large - інстанс, який, як і c4.large, розрахований на роботу з базою, але з акцентом не на обчислення, а на роботу з пам'яттю.

m4.large - збалансований інстанс для використання у великих проектах. Цей інстанс відрізняється від інстансів t більшою пропускною здатністю мережевого порту та потужністю процесора на один потік.
Характеристики інстансів наведено в таблиці 1.

Таблиця 1. Характеристики інстансів

Назва	vCPU	Ram	Network
t2.micro	1	1	low to moderate
c4.large	2	4	moderate
m4.large	2	8	moderate
t3.large	2	8	5 gbit/s
r4.large	2	16	10 gbit/s

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. Створюємо інстанс серверу, на якому буде працювати програма, та обираємо ОС Linux, на якій буде працювати сервер. Генеруємо RSA ключ для підключення до сервісу через SSH-tunel. Детальну інформацію про сервер показано на рис.3.

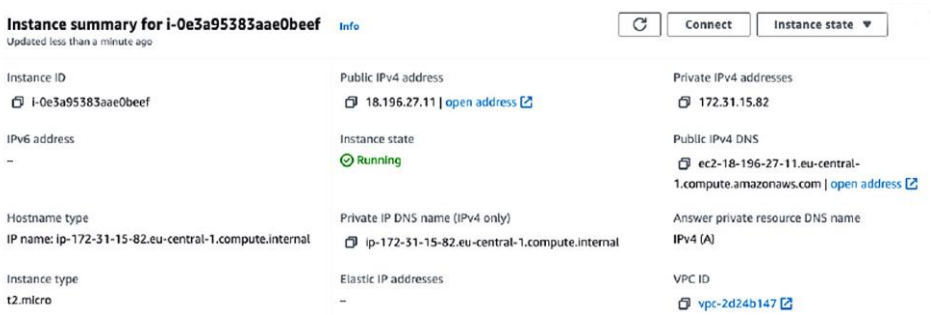

Рис.3. Детальна інформація про створений сервер.

Для підключення до серверу використовуємо файл згенерованого RSA ключа та Public IPv4 DNS серверу. Використовуємо Maven для створення .jar файлу програми та завантажуємо програму на сервер. Встановлюємо Java та запускаємо програму. Після цього, вказавши public IPv4 сервера підключаємось до API сервісу та взаємодіємо з ним.

Тестування web-застосунку у хмарних сервісах

Тестування web-застосунків забезпечує їх коректну роботу. Існує багато видів тестування і всі вони мають особливості реалізації у хмарних сервісах [1]. Для різних видів застосунків підбирають різні стратегії тестування та відповідні архітектури за якими слідує класифікація засобів тестування на основі їх стратегій тестування [2,3]. Для того, щоб уникнути проблеми зменшення швидкодії роботи сайту при збільшенні користувачів розробники проводять спеціальні тести і перевіряють проекти на перформанс, тобто навантажувальне тестування. Результати навантажувального

тестування дозволяють переконатися, що застосунок однаково якісно працює, як з одним, так і з багатьма користувачами. Якщо параметри тестування задовільняють вимоги поставлені до роботи застосунку, наприклад на всі запити для певної кількості користувачів отримано відповіді в межах певного часу, можемо вважати роботу програми якісною та стабільною [4].

Проект для тестування представляє собою ряд сценаріїв використання програми для проведення змагань. Зокрема тестування CRUD операцій та їх послідовностей, таких як реєстрація учасника на змагання, а потім редагування, видалення учасників, отримання інформації про зареєстрованих учасників, а також отримання поточної інформації про змагання з іншого сервісу. Тестувати реєстрацію та менеджмент учасників навантажувальними тестами немає сенсу, тому що користувачами, які мають право це робити, є тільки тренери та адміністратори змагань. Відповідно тестувати POST та PUT запити немає сенсу. Іншим випадком тестування web-застосунку є дослідження його роботи під час проведення змагань. У змаганнях приймають участь багато спортсменів, змагання відбуваються на декількох майданчиках одночасно, судді виставляють оцінки, користувачі можуть отримати результати змагань online. Отримувати дані від сервісу можуть глядачі, тренери, адміністратори та учасники. Тому оптимальним обрано сценарій отримання даних про результати змагань та оцінки суддів, а потім отримання детальних даних про учасника кожним з користувачів з другого сервісу. Сервіс менеджменту учасників та сервіс з отримання поточних даних змагань використовують одну базу даних. Сервіс для отримання даних в реальному часі є значно більш завантаженим, тому проводити тестування доцільно саме на ньому.

Важливим фактором тестування є вибір стратегії тестування. Для тестування web-застосунку вибрано сценарій стабільної кількості користувачів за певний період часу. Така стратегія реалізовується rampUsers(N) during M, яка дозволяє протягом певного часу M викликати загалом N користувачів.

У такому сценарії користувач послідовно виконує два запити. Це дає певну девіацію активних користувачів на сервері та відповідає поставленій задачі тестування.
Програма для тестування написана мовою Scala за використання фреймворку для тестування gatling, Gatling – це фреймворк для проведення навантажувального тестування. Gatling дозволяє швидко створювати тести, які не тільки тестують API сервіси з точки зору правильного виконання запитів, а також з точки зору швидкості виконання цих запитів. Загалом було створено два типи тестів з різними параметрами для навантажувального тестування розробленого програмного додатку.

Тестування web-застосунку навантаженням 500 користувачів в секунду протягом години

500 користувачів на секунду є оптимальною кількістю користувачів, для якої тести ще максимально об’єктивні на тривалому проміжку часу, оскільки при тривалому навантаженні припиняють працювати або JVM з програмою тестування з помилкою про забагато відкритих файлів, або виключається мережеве обладнання. Зведені результати тестування для всіх типів серверів представлено у таблиці (Таблиця 2). У лівій колонці таблиці представлені параметри: мінімальний, максимальний та середній час відповіді сервера у ms, кількість успішних чи невдалих відповідей сервера. При тривалому тестуванні також можливо оцінити завантаженість процесора серверу.

Таблиця 2. Результати тестування навантаженням 500 користувачів в секунду протягом години

	t2.micro	t3.large	r4.large	m4.large	c4.large
Min response time,ms	29	27	29	28	28
Max response time,ms	60012	20857	29142	55860	60014
Mean response time,ms	488	97	127	179	92
Successful request	2799387	3599983	3598062	3591530	3558084
Unsuccessful request	800613	17	1938	108470	41916
Max CPU Load	85%	20%	60%	47%	45%

З аналізу параметрів результатів тестування видно, що інстанси t2.micro, m4.large, та c4.large показують себе значно гірше ніж t3.large та r4.large за швидкодією обробки запитів та дають досить значну кількість невдалих запитів. Через це вважаємо, що система побудована на t2.micro, m4.large, та c4.large не може вважатися стабільною системою. Серед t3.large та r4.large з тривалим навантаженням краще справився саме t3.large, оскільки у нього менший середній час обробки запиту, та менша кількість помилок, а саме 17 помилок на 3.6 млн запитів.

Gatling генерує звіти в графічній формі. Звіти html доступні в каталозі build/gatling-results. На рис.4 представлено звіт для інстансу t3.large. Графічний звіт включає: графік із кількістю запитів, згрупованих за середнім часом, графік з загальною кількістю запитів і максимальною відповіддю за часом в процентилях, графік із кількістю запитів, успішно оброблених програмою за секунду та графік із кількістю відповідей за секунду. Візуальне представлення підтверджує той факт, що інстанс t3.large краще справився з навантаженням ніж інші інстанси. Для порівняння використано графік представлений на рис.5 із кількістю відповідей за секунду для інстансу t2.micro. З графіка видно, що більшість відповідей t2.micro класифікуються, як невиконані.

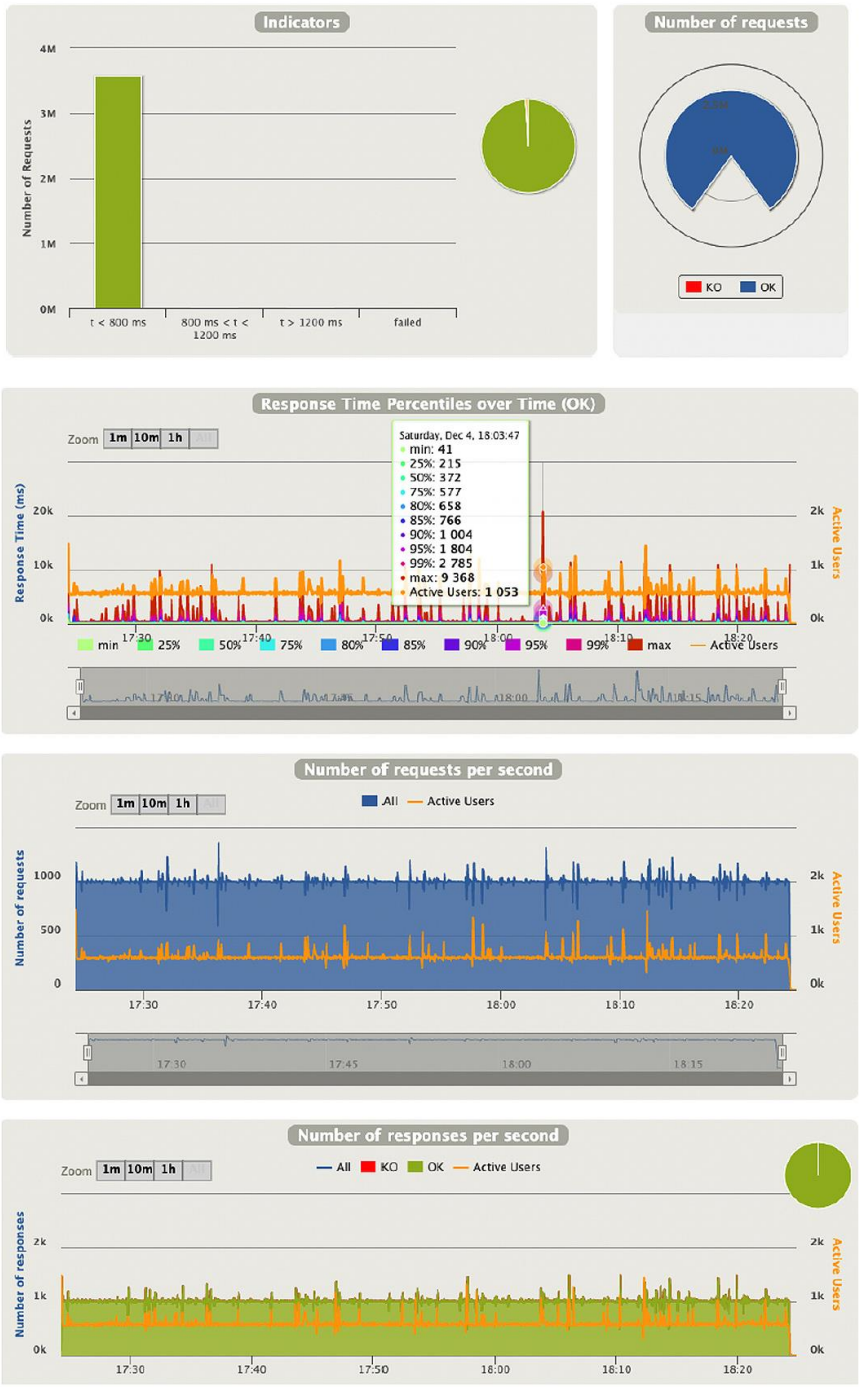

Рис.4. Візуалізація результатів тестування для t3.large

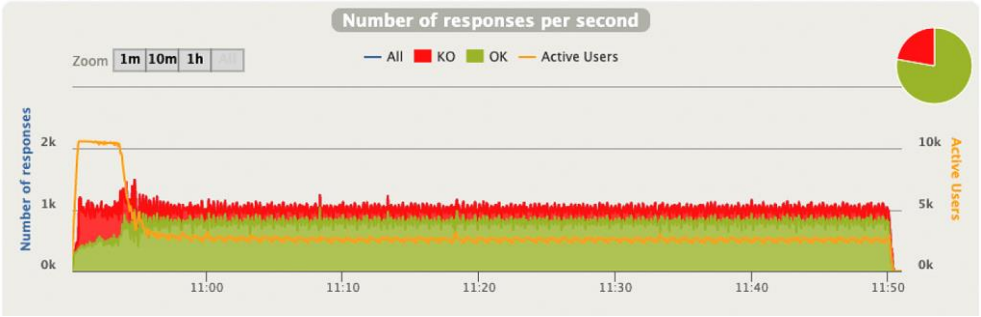

Рис.5. Візуалізація результатів тестування для t2.micro

Тестування web-застосунку швидкими тестами

Для різних типів інстансів проведено тести з кількістю користувачів в межах від 100 до 600 користувачів на секунду з кроком100 користувачів. Результати тестування представлені в таблиці (Таблиця 3).

Таблиця 3. Результати тестування

	n	t2.micro	t3.large	r4.large	m4.large	c4.large
Min response time,ms	100	30	27	29	27	27
	300	30	27	29	28	28
	400	31	28	29	28	28
	600	30	28	28	28	28
Max response time,ms	100	3297	1056	1064	1083	458
	300	656	153	414	7679	514
	400	859	8650	902	8337	38229
	600	58949	59937	59638	59979	60107
Mean response time,ms	100	191	51	54	52	50
	300	66	49	50	54	53
	400	66	51	56	76	2014
	600	3188	2125	1618	2359	1585
Successful request	100	19999	20000	20000	20000	20000
	300	60000	60000	60000	59988	20000
	400	79187	79937	80000	79424	77782
	600	86140	94974	105993	93394	105504
Unsuccessful request	100	1	0	0	0	0
	300	0	0	0	12	0
	400	813	63	0	576	2218
	600	33860	25026	14007	23686	14496

Тестування роботи web-застосунку при навантаженні 100 користувачів за секунду показує, що всі типи інстансів показують задовільний стабільний результат. Дещо довший середній час опрацювання одного запиту для t2.micro може бути пов’язаний з

слабшим vCPU та меншою кількісьтю оперативної пам'яті (Таблиця 1). Результати тестування на рівні 200 користувачів за секунду не сильно відрізняються від попередніх результатів для 100 користувачів за секунду, однак час запиту на m4.large несподівано виріс, а його графік показує декілька скачків в сторону росту часу запиту. На рівні 300 користувачів за секунду інстанси t3.large та r4.large працюють стабільно, а у m4.large і надалі швидко зростає кількість помилок та середній час відповіді на запити. Варто відзначити, що незважаючи на власні параметри і нижчу ціну t2.micro також демонструє задовільні результати тестування на рівні з t3.large та r4.large. При тестуванні на рівні 400 користувачів за сек тип c4.large втратив стабільність, через що майже 3% запитів не були успішними, а також середній час обробки запиту у 2014мс робить c4.large значно гіршим за інші типи, включаючи t2.micro. Найкраще показав себе r4.large, який не тільки стабільний, але навіть на цьому рівні завантаженості відпрацював без помилок. При тестуванні на рівні 500 користувачів за секунду t2.micro вже не може дати хороших результатів, оскільки просто не може їх обробити. Неочікувано багато помилок дав m4.large, який перестав відповідати на запити, а c4.large навпаки дав дуже стабільні результати з 0 кількістю помилок. t3.large та r4.large показали стабільні результати з невеликою кількістю помилок. Результати тестування на рівні 600 користувачів за сек показують, що усі типи інстансів почали давати велику кількість помилок та значно збільшився середній час обробки запиту. Найкращі результати дали інстанси r4.large та c4.large. Результати для цих інстансів є подібними як за парметрами, так і за графічним представленням результатів. Інстанси t3.large та m4.large не справляються з таким потоком користувачів, через що кожен 5 запит є невдалим.

Висновок

У результаті виконання роботи створено веб-застосунок для проведення спортивних змагань та протестовано його швидкодію та стабільність роботи на основі різних типів серверів у хмарних сервісах AWS EC2. Після проведених тестів визначено, що найкраще для реалізації даного проекту підходять інстанси типів t3.large та r4.large. Серед переваг r4.large є більша стійкість до неочікуваних напливів користувачів завдяки більш швидкій пам'яті та мережі. Однак, t3.large є більш збалансованим та стабільним, а також ключовою перевагою t3.large є значно менша ціна за годину ніж у r4.large. Це в свою чергу дає досить суттєву різницю в ціні при роботі в кластері і чим більшим буде кластер, тим суттєвішою буде різниця. Також два t3.large будуть працювати суттєво продуктивніше і стабільніше ніж один r4.large. Отже, після проведеного тестування роботи програми для ведення змагань у хмарних сервісах будуть використані t3.large інстанси, оскільки вони дешевші, стабільніші та витримують без збоїв і проблем інтенсивний трафік до 400 користувачів на секунду на 1 сервер при стрибках трафіку та до 500 користувачів при стабільному навантаженні протягом тривалого часу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] *Gao J.* Cloud Testing-Issue, Challenges, Needs and Practice / J. Gao, X. Bai, W. Tsai // Software Engineering: An International Journal (SEIJ). -Vol. 1, No. 1. -P. 9-23.

[2] *Lnenicka M.* Classification and Evoluation of Cloud-Base Testing Tools: The Case Study of Web Application Security Testing/ M. Lnenicka, J. Capek// Acta Informatica Pragensia, 2018, 7(1), 40–57. DOI: 10.18267/j.aip.113

[3] *Liu W.* Performance Test and Improvement of Computer Network Virtualization Software in Cloud Computing Environment//Volume 2022 | Article ID 6965880 https://doi.org/10.1155/2022/6965880

[4] *Li H.* Research on Cloud Performance Testing Model/ Li H., Li X., Wang H., Zhand J. //Conference: IEEE 19th International Symposium on High Assurance Systems Engineering, 2019. DOI:10.1109/HASE.2019.00035

BENCHMARKING PARALLEL REQUEST PROCESSING SPEEDS ON AWS CLOUD SERVICES

O. Sihunov, L. Demkiv

Ivan Franko National University of Lviv,
50 Drahomanova St., 79005 Lviv, Ukraine
lidia.demkiv@gmail.com

The article presents the results of testing the web application for conducting sports competitions in cloud services. Testing on different types of servers of the same price category under scenarios closest to real ones, in particular simultaneous data acquisition by many users, was carried out.The web application is written using the Spring Java framework, the Maven dependency builder, and the IntelliJ IDEA development environment. The project uses Scala and Gatling for testing. The program consists of two independent microservices, one of which is responsible for user registration and management, and the other for real-time data transmission. An important factor in testing is the choice of testing strategy.

A scenario of a stable number of users over a certain period of time was chosen for testing the web application. Such a strategy is implemented by rampUsers(N) during M, which allows during a certain time M to call a total of N users. Performance tests with the number of users ranging from 100 to 600 users per second in steps of 100 users and performance tests of 500 users per second for an hour were performed for different types of instances. The parameters are defined: the stability of the application on a certain type of server, the speed of processing the request by the server for different numbers of users, and errors that occur during the operation of the application.

The most optimal type of servers for this application is determined, which gives the maximum ratio of performance and stability to the price for this type of tasks. Instances of the t3.large and r4.large types are best suited for the implementation of this project. Among the advantages of r4.large is greater resilience to unexpected influxes of users thanks to faster memory and network. However, t3.large is more balanced and stable, and the key advantage of t3.large is its significantly lower hourly price than r4.large.

Keywords: cloud services, instances, AWS EC2, Gatling, types of loads, test scenarios.

Стаття надійшла до редакції 22.11.2022.
Прийнята до друку 29.11.2022.