Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерної інженерії та управління_____
(повна назва)

Кафедра _____Автоматизації проектування обчислювальної техніки_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____
(рівень вищої освіти)

_____
_____Система балансування навантаження веб-застосунків на базі_____
_____хмарної інфраструктури_____
(тема)

Виконав: студент 2 курсу, групи СКСм-18-2
_____Ковріжний О.В._____
(прізвище, ініціали)

Спеціальність _123 Комп'ютерна інженерія
_____
(код і повна назва спеціальності)

Тип програми___Освітньо-професійна____
_____
(освітньо-професійна або освітньо-наукова)

Освітня програма_____
__Спеціалізовані комп'ютерні системи_____
_____
(повна назва освітньої програми)

Керівник ____Проф. Кривуля Г.Ф._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
Кафедра Автоматизації проектування обчислювальної техніки
Рівень вищої освіти другий (магістерський)
Спеціальність 123 – Комп'ютерна інженерія
Тип програми Освітньо-професійна
Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Ковріжному Олексію Владиславовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Система балансування навантаження веб-застосунків на базі хмарної інфраструктури

затверджена наказом по університету від 04 листопада 2019 р. № 1624 Ст
2. Термін подання студентом роботи до екзаменаційної комісії 11 грудня 2019 р.
3. Вихідні дані до роботи
Amazon Web Services
Системи балансування навантаження
Хмарні обчислення
4. Перелік питань, що потрібно опрацювати в роботі
Хмарні обчислення та масштабування
Балансування навантаження – методи та алгоритми
Розгортання веб-додатку
Тестування методів балансування навантаження

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

21 слайд

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	01.09.19	
2	Аналіз предметної області та опрацювання джерел	02.09.19 – 1.10.19	
3	Побудова системи та розгортання веб-додатку	2.10.19 – 20.10.19	
4	Тестування методів балансування навантаження	20.10.19 – 1.11.19	
5	Оформлення пояснювальної записки	11.11.19 – 27.11.19	
6	Оформлення графічного матеріалу	28.11.19 – 04.12.19	
7	Перевірка виконаного проекту керівником	05.12.19 – 11.12.19	
8	Захист роботи	.12.19	

Дата видачі завдання _____

Студент _____
(підпис)

Керівник роботи _____ ____Проф. Кривуля Г.Ф.____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить: 96 сторінок, 59 рисунків, 1 таблицю, 32 джерела за переліком посилань.

БАЛАНСУВАННЯ НАВАНТАЖЕННЯ, ХМАРНІ ОБЧИСЛЕННЯ, AMAZON AWS, МАСШТАБУВАННЯ, МЕРЕЖЕВА МОДЕЛЬ OSI.

Метою дослідження є порівняльний аналіз існуючих систем балансування шляхом побудови хмарної інфраструктури для обраного вебдодатку, вибір критеріїв для аналізу і порівняння методів та алгоритмів балансування навантаження у хмарному середовищі, дослідження конфігурації систем балансування навантаження та надання практичних рекомендацій.

У роботі побудована хмарна інфраструктура, використовуючи Amazon Web Services. Розгорнуто веб-додаток виду системи керування контентом – WordPress. Розглянути існуючі методи та алгоритми балансування навантаження трафіку у мережі. Проведено тестування продуктивності з обраними критеріями для визначення коректності роботи балансувальника, автоматичного масштабування та стабільності системи загалом.

РЕФЕРАТ

Пояснительная записка содержит: 96 страниц, 59 рисунка, 1 таблицу, 32 источника согласно перечню ссылок.

БАЛАНСИРОВКА НАГРУЗКИ, ОБЛАЧНЫЕ ВЫЧИСЛЕНИЯ, AMAZON AWS, МАСШТАБИРОВАНИЕ, СЕТЕВАЯ МОДЕЛЬ OSI

Целью исследования является сравнительный анализ существующих систем балансировки путем построения облачной инфраструктуры для выбранного веб-приложения, выбор критериев для анализа и сравнения методов и алгоритмов балансировки нагрузки в облачной среде, исследование конфигураций систем балансирования нагрузки и предоставление практических рекомендаций.

В работе построена облачная инфраструктура, используя Amazon Web Services. Развернуто веб-приложение вида системы управления контентом – WordPress. Рассмотрены существующие методы и алгоритмы балансировки нагрузки сетевого трафика. Произведено тестирование продуктивности с выбранными критериями для определения корректности работы балансировщика, автоматического масштабирования и стабильности системы в целом.

ABSTRACT

The explanatory note contains: 96 pages, 59 figures, 1 table, 32 sources according to the list of links.

LOAD BALANCING, CLOUD COMPUTING, AMAZON AWS, SCALING, OSI NETWORK MODEL

The purpose of the work is a comparative analysis of existing balancing systems by building a cloud infrastructure for the selected web application, selecting criteria for analyzing and comparing load balancing methods and algorithms in the cloud, exploring the configuration of load balancing systems and providing practical recommendations.

The cloud infrastructure was built using Amazon Web Services. A web application of the form of a content management system - WordPress has been deployed. Existing methods and algorithms for balancing the load of network traffic are considered. Productivity testing was performed with the selected criteria to determine the correct operation of the balancer, automatic scaling and stability of the system as a whole.

ЗМІСТ

ВСТУП 10
1 ХМАРНІ ОБЧИСЛЕННЯ 13

1.1 Моделі хмарного обслуговування (IaaS, PaaS, SaaS) 16
1.2 Переваги та недоліки хмарних обчислень 21
1.3 Масштабування у хмарному середовищі 23
1.3.1 Горизонтальне масштабування 24
1.3.2 Вертикальне масштабування 26
1.4 Постачальники хмарних обчислень 28
2 БАЛАНСУВАННЯ НАВАНТАЖЕННЯ 31
2.1 Пояснення термінології 33
2.2 Різновиди балансувальників навантаження 34
2.3 Мережеві рівні балансування навантаження 36
2.4 Принцип роботи балансувальника навантаження 38
2.4.1 Моніторинг працездатності (Health check) 40
2.4.2 Підтримка сеансу з’єднання (Session Persistence) 42
2.4.3 Надлишкові балансувальники навантаження (High Availability) 43
2.5 Топології систем балансування навантаження 44
2.5.1 Проксі по середині (Middle proxy) 45
2.5.2 Крайове розташування (Edge proxy) 46
2.5.3 Використання вбудованої клієнтської бібліотеки 46
2.5.4 Sidecar проксі-сервер 47
2.5.5 Глобальне балансування навантаження (GSLB) 48
2.6 Методи та алгоритми балансування навантаження 51
2.6.1 Round-Robin та Weighted Round-Robin 54
2.6.2 Least Connection 57
2.6.3 Source IP hash та Sticky Sessions 59
3 РОЗГОРТАННЯ ВЕБ-ДОДАТКУ НА БАЗІ ХМАРНОЇ ІНФРАСТРУКТУРИ 61
3.1 Amazon Web Services 61
3.2 Побудова інфраструктури 65
3.2.1 Application Load Balancer 65
3.2.2 Amazon Relational Database Service (RDS) 75
3.2.3 AWS Elastic Beanstalk 76

3.2.4 Auto Scaling Group 81
3.2.5 Network Load Balancer 84
4 ТЕСТУВАННЯ МЕТОДІВ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ 85
4.1 Балансування з Application Load Balancer 87
4.2 Балансування з Network Load Balancer 95
ВИСНОВКИ 97
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 100
ДОДАТОК А ..... **Ошибка! Закладка не определена.**
Графічна частина проекту ..... **Ошибка! Закладка не определена.**

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AWS – Amazon Web Services, сервіс який надає платформу хмарних обчислень в оренду приватним особам, компаніям та урядам на основі платної підписки.

Azure – Microsoft Azure, сервіс який надає платформу хмарних обчислень в оренду приватним особам, компаніям та урядам на основі платної підписки.

OSI – мережева модель набору мережевих протоколів, за допомогою якого здійснюється взаємодія між різноманітними пристроями.

IP – Internet Protocol, протокол мережевого рівня для передавання датаграм між мережами.

L4 – Layer 4, транспортний рівень моделі OSI, призначений для доставки даних без помилок, втрат і дублювання.

L7 – Layer 7, прикладний рівень моделі OSI, призначений для забезпечення взаємодії мережі та користувача.

TCP/IP – мережева модель передачі даних, представлених у цифровому вигляді, описує спосіб передачі від джерела до отримувача інформації.

SSL – Secure Sockets Layer, протокол, що забезпечує конфіденційність та цілісність даних, котрі пересилаються між веб-серверами і веб-браузерами.

HTTP – HyperText Transfer Protocol, протокол прикладного рівня, для обміну повідомленнями по схемі «запит-відповідь».

HTTPS – HTTP Secure, протокол який є розширенням HTTP, використовується для захищеного з’єднання між пристроями.

NAT – Network Address Translation, механізм у мережах TCP/IP який дозволяє перетворювати IP-адреси транзитних пакетів.

Бекенд – Backend, програмно-апаратна частина веб-додатку.

ВСТУП

Розвиток інформаційних технологій неспинно нарощує темпи свого зростання. Невід’ємною частиною цих технологій є всесвітня система сполучених комп’ютерних мереж, яка отримала назву – Інтернет. Кількість користувачів мережі також зростає з кожним роком, за результатами досліджень видання “wearesocial.com”, кількість інтернет-користувачів зросла на 366 млн. у порівнянні з 2018 роком та досягла відмітки у 4.39 млрд. загалом [1]. Більшість систем доступні для відкритого користування у вигляді веб чи мобільних додатків. Відкритість інформації та доступність сервісів приводить до неминучого зростання кількості користувачів, особливо це стосується таких ресурсів як: інформаційні сервіси, інтернет-магазини, відеосервіси тощо. – отже потреба у забезпеченні достатньої продуктивності та безперервності роботи, навіть при великій кількості запитів, являється невід’ємною частиною успіху конкретного сервісу.

Останнім часом все більшої популярності набувають хмарні обчислення (cloud computing), особливо завдячуючи таким проектам як: Amazon Web Services, Google Cloud, Microsoft Azure. Саме компанія Amazon, яка представила сервіс Elastic Compute Cloud у 2006 році, дала початок у розвитку хмарних обчислень. Найбільш відомі сервіси типу Google Docs чи Gmail та багато інших побудовані на базі хмарної інфраструктури, перевагою якої є масштабування та гнучкість. Прикладом використання хмарних технологій є послуга віртуально виділеного серверу, яка являє собою технологію хостингу, що поєднує в собі обчислювальну здатність окремо виділеного серверу з гнучкістю і простотою управління.

Концепція хмарних обчислень полягає у забезпеченні зручного мережевого доступу до якогось загального фонду обчислювальних ресурсів, які можна конфігурувати під власні потреби. Такими ресурсами може бути мережі передачі даних, сервери, пристрої зберігання даних і т. ін. – головною умовою для яких є оперативне надання і звільнення у разі потреби, з мінімальними експлуатаційними затратами та звертаннями до провайдера. Постачальники хмарних сервісів, такі як Amazon Web Services (AWS) [2], володіють підключеним до мережі обладнанням і виконують його обслуговування, у той час коли користувач розподіляє та використовує необхідні йому ресурси за допомогою спеціального веб-додатку.

Особливою перевагою хмарних технологій є можливість оперативного масштабування та розгортання інфраструктури за лічені хвилини. Масштабованість являється дуже важливим аспектом електронних систем, систем баз даних, маршрутизаторів, мереж і т. ін. при необхідності їх роботи під великим навантаженням. До появи хмарних обчислень при потребі забезпечення роботи при пікових навантаженнях приходилось виділяти надлишкові обчислювальні ресурси, що приводило до зростання грошових витрат та ускладнювало обслуговування. Наразі використовуючи віртуальні

виділені сервери у хмарі є можливість миттєвого масштабування у випадку недостатньої обчислювальної здатності. Це дозволяє зменшити витрати і покращити можливість забезпечення потреб користувачів.

Недоступність сервера у зв’язку з надмірною кількістю запитів загрожує серйозними наслідками – як моральними, так і матеріальними. Користувач не буде використовувати нестабільний сервіс, якщо той підвів його у важливий момент. На початку, проблеми недостатньої продуктивності сервера в зв’язку з зростанням навантаження можна вирішувати шляхом вертикального масштабування – нарощування потужності сервера, або ж оптимізацією алгоритмів, програмних кодів і т. ін. З часом такі заходи стають недостатніми та доводиться вдаватися до кластеризації: кілька серверів об’єднуються у кластер; навантаження між ними розподіляється за допомогою комплексу спеціальних методів, які називаються балансуванням.

Балансування навантаження розподіляє трафік між декількома обчислювальними пристроями, такими як: комп'ютери, кластери, мережі, центральні процесори або диски. Метою балансування навантаження є: оптимізація використання ресурсів, максимізація пропускної здатності, зменшення часу відгуку та запобігання перевантаження якось одного ресурсу. Використання декількох компонентів балансування замість одного може підвищити надійність і доступність за рахунок резервування. Балансування зазвичай передбачає наявність спеціального програмного забезпечення чи апаратних засобів, таких як: багаторівневий комутатор чи система доменних імен, у вигляді серверного процесу [3].

Мета дослідження – порівняльний аналіз існуючих систем балансування шляхом побудови хмарної інфраструктури для обраного веб-додатку, вибір критеріїв для аналізу і порівняння методів та алгоритмів балансування навантаження у хмарному середовищі, дослідження конфігурації систем балансування навантаження та надання практичних рекомендацій.

Об’єкт дослідження – процеси побудови системи балансування навантаження у хмарному середовищі.

Предмет дослідження – методи і алгоритми балансування навантаження веб-додатків у просторі хмарних обчислень, критерії їх порівняння та визначення оптимального варіанту.

Наукова новизна – дослідження та порівняння засобів балансування навантаження у хмарному середовищі та створення конфігурації однієї із систем балансування навантаження.

Практична цінність - надання рекомендацій щодо вибору конфігурації систем балансування навантаження та проведення експериментального дослідження побудованої системи методом тестування продуктивності.

Зміст дослідження – побудова хмарної інфраструктури, розгортання вебдодатку на базі побудованої інфраструктури з метою порівняння систем балансування, створення практичних рекомендацій по конфігурації, проведення тестування продуктивності шляхом навантажувального тестування та подальшим аналізом отриманих результатів.

1 ХМАРНІ ОБЧИСЛЕННЯ

Хмарні обчислення (англ. Cloud Computing) – модель забезпечення повсюдного та зручного доступу на вимогу через мережу до спільного пулу обчислювальних ресурсів, що підлягають налаштуванню (наприклад, комунікаційних мереж, серверів, засобів збереження даних, прикладних програм та сервісів), і які можуть бути оперативно надані та звільнені з мінімальними управлінськими затратами та зверненнями до провайдера [4]. Така модель дозволяє прискорити поширення інновацій, підвищує гнучкість

ресурсів і забезпечують економію завдяки високому рівню масштабованості. Користувач зазвичай платить за використані ресурси, що значить немає необхідності в оплаті повного пакету послуг, які надаються центрами обробки даних чи фізичними серверами. Плата за використаний трафік є значною перевагою у виборі хмарних сервісів для різного роду компаній-стартапів та нових проектів, успіх яких однозначно непередбачений.

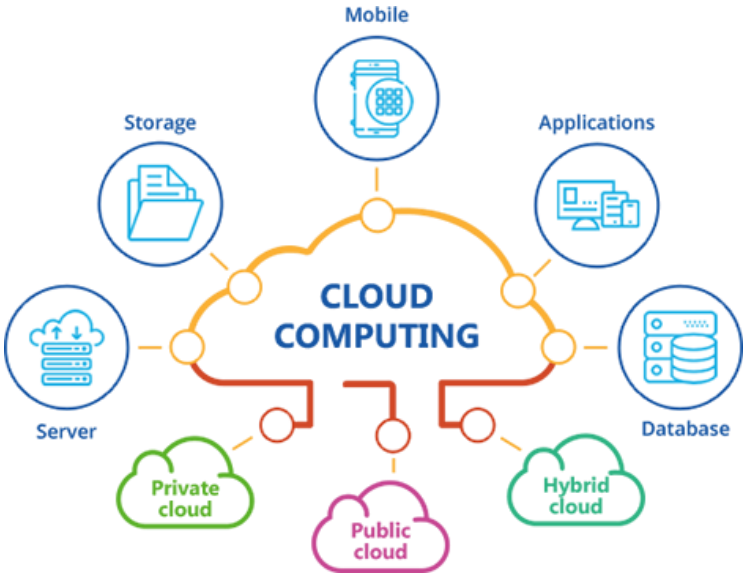


Рисунок 1.1 – Хмарні обчислення [5]

Одним із перших широкодоступних хмарних сервісів стала електрона пошта з веб-інтерфейсом, усі дані якої зберігаються та оброблюються на віддалених серверах, в той же час користувач має змогу взаємодіяти з цим сервісом через зручний інтерфейс у браузері на своєму персональному комп’ютері чи смартфоні.

Хмарні обчислення мають наступні ключові характеристики.

1. Гнучкість: організації, які використовують хмарні обчислення можуть покращити маневреність користувачів завдяки повторному використанню, додаванню чи розширенню обчислювальних ресурсів технологічної інфраструктури.

2. Продуктивність: постачальники послуг мають спеціалістів, які спостерігають та обслуговують технологічну інфраструктуру вашої системи, в той же час необхідні замовнику архітектури будуються завдяки веб-сервісам. 3. Масштабованість та еластичність: за рахунок динамічного надання ресурсів (“за запитом”), практично у реальному часі, існує можливість збільшувати продуктивність, коли є така потреба, та зменшувати її, коли ресурси не використовуються.

4. Надійність: використання надлишкових ресурсів робить гарно спроектовані хмарні структури придатними до забезпечення неперервності обслуговування користувачів сервісу та аварійного відновлення у разі надзвичайних ситуацій.

5. Фінансові витрати: хмарні обчислення дозволяють уникнути капітальних витрат на придбання обладнання та програмного забезпечення, налаштування та експлуатацію локальних центрів обробки даних, які тягнуть за собою витрати на цілодобову подачу струму, систему охолодження та штат кваліфікованих спеціалістів для обслуговування цієї інфраструктури. Постачальник послуг автоматично обчислює спожиті ресурси на певному рівні абстракції (наприклад, обсяг збережених даних, пропускна здатність, кількість користувачів, кількість транзакцій), і на основі цих даних оцінює обсяг наданих споживачам послуг.

З точки зору постачальника, завдяки об'єднанню ресурсів та непостійному характеру споживання з боку споживачів, хмарні обчислення дозволяють економити на масштабах, використовуючи менші апаратні ресурси, ніж при виділенні окремих потужностей для кожного споживача, а за рахунок автоматизації процедур модифікації виділення ресурсів істотно знижуються витрати на абонентське обслуговування [6]. З точки зору споживача, ці характеристики дозволяють отримати послуги з високим рівнем доступності (англ. high availability) і низькими ризиками непрацездатності,

забезпечити швидке масштабування обчислювальної системи завдяки еластичності без необхідності створення, обслуговування і модернізації власної апаратної інфраструктури.

1.1 Моделі хмарного обслуговування (IaaS, PaaS, SaaS)

Більшість служб хмарних обчислень поділяються на три загальні групи: інфраструктура як послуга (англ. Infrastructure as a Service – IaaS), платформа як послуга (англ. Platform as a Service – PaaS) та програмне забезпечення як послуга (англ. Software as a Service – SaaS). Ці моделі пропонують все більшу абстракцію, тим самим часто зображуються у вигляді слоїв стеку рис.1.2 [7].

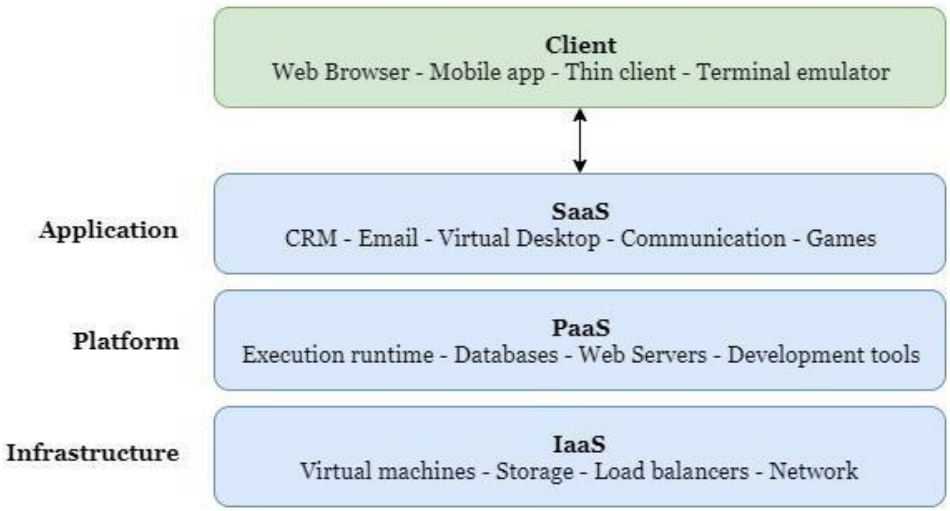


Рисунок 1.2 – Моделі хмарних обчислень, організовані у вигляді слоїв стеку
Розглянемо більш детально кожну із моделей.

Інфраструктура як сервіс (IaaS) – це модель хмарних обчислень, яка надає віртуалізовані обчислювальні ресурси через Інтернет. В моделі IaaS сторонній постачальник розміщує своє обладнання, програмне забезпечення, сервери, системи зберігання та інші компоненти інфраструктури від імені своїх користувачів. Постачальники IaaS також розміщують веб-додатки

користувачів та забезпечують послуги резервного копіювання, обслуговування системи, планування відмовостійкості. Споживач може контролювати операційні системи, віртуальні системи зберігання даних і встановлені програми, а також обмежений контроль набору доступних послуг. Серед доступних послуг є такі: балансування навантаження, міжмережевий екран та система доменних імен (DNS – Domain Name System). Балансування надає єдину точку доступу для декількох серверів, які працюють за нею, забезпечуючи розподілення мережевого трафіку. DNS створює ієрархічну систему імен для пристроїв будь-якого вигляду, які використовують IPадресацію для ідентифікації у мережі – система DNS зв'язує доменні імена з IP-адресами.

Структурно IaaS можна поділити на три частини: апаратні засоби, які включать себе: сервери, системи зберігання даних, мережеве обладнання тощо; операційні системи та системне програмне забезпечення: засоби віртуалізації, автоматизації, управління ресурсами; об'єднуюче програмне забезпечення: управління системами, засоби спостереження.

Завдяки технологіям віртуалізації провайдер має можливість об'єднати своє обладнання в єдину систему, розділюючи її обчислювальні потужності на частини віртуально, тим самим забезпечуючи використання всіх ресурсів без простою. Результатом такої системи є можливість змінити модель оплати за наданий сервіс. Замість придбання, управління та амортизації апаратних засобів можна перейти до моделі почасової оплати процесорного часу, дискового простору, пропускної здатності мережі, що може допомогти зекономити при правильному використанні.

Найбільшими гравцями на ринку інфраструктури як послуги є: Amazon, Microsoft, VMware, Rackspace та Red Hat. Хоча деякі з них пропонують більше, ніж просто інфраструктуру, їх об'єднує мета продавати базові обчислювальні ресурси.

Платформа як сервіс (PaaS) – це модель хмарних обчислень, яка доставляє дані веб-додатку через Інтернет. Постачальник надає платні послуги апаратного і програмного забезпечення, зазвичай які необхідні при розробці веб-додатків, у якості послуги. Апаратне і програмне забезпечення розміщено в інфраструктурі постачальника, що звільнює користувача від необхідності встановлювати власні ресурси для розробки чи запуску нового додатку. Користувач не має змоги змінювати конфігурацію наданої системи власноруч, однак може звернутися із спеціальним запитом на зміни до провайдера. Наприклад, Google App Engine являється типовим представником PaaS, який створений для надання послуг по розробці та розміщенню клієнтський вебдодатків у центрах обробки даних (дата-центрах), які обслуговуються компанією Google. Веб-додатки поміщаються в ізольовану середу (sandbox – англ. пісочниця) і виконуються на декількох серверах. App Engine пропонує автоматичну масштабованість в залежності від кількості вхідних запитів до веб-додатку. Усі популярні мови програмування та фреймворки доступні у даному сервісі, однак слід враховувати, детальне налаштування та зміна версії на будь-яку бажану часто неможлива, що є обмеженням самої концепції PaaS.

Програмне забезпечення як сервіс (SaaS) – це модель розповсюдження програмного забезпечення, в котрій додаток розміщується виробником чи спеціальним постачальником послуг через Інтернет. Користувачі сплачують кошти за використання даних сервісів (наприклад, Gmail, Office365 та ін.). Доцільність використання SaaS пояснюється тим, що провайдер такого сервісу обслуговує єдине програмне ядро, яким користуються всі клієнти, і тому витрачає меншу кількість ресурсів у порівнянні з управлінням окремими копіями для кожного замовника. Крім того, використання єдиного програмного ядра дозволяє планувати обчислювальні потужності і зменшує проблему пікових навантажень для окремих замовників. Все це дозволяє постачальникам SaaS рішень істотно знизити вартість обслуговування

програмного забезпечення. У кінцевому результаті, періодична вартість послуг для кінцевого замовника стає нижче витрат, що виникають при використанні класичної моделі ліцензування.

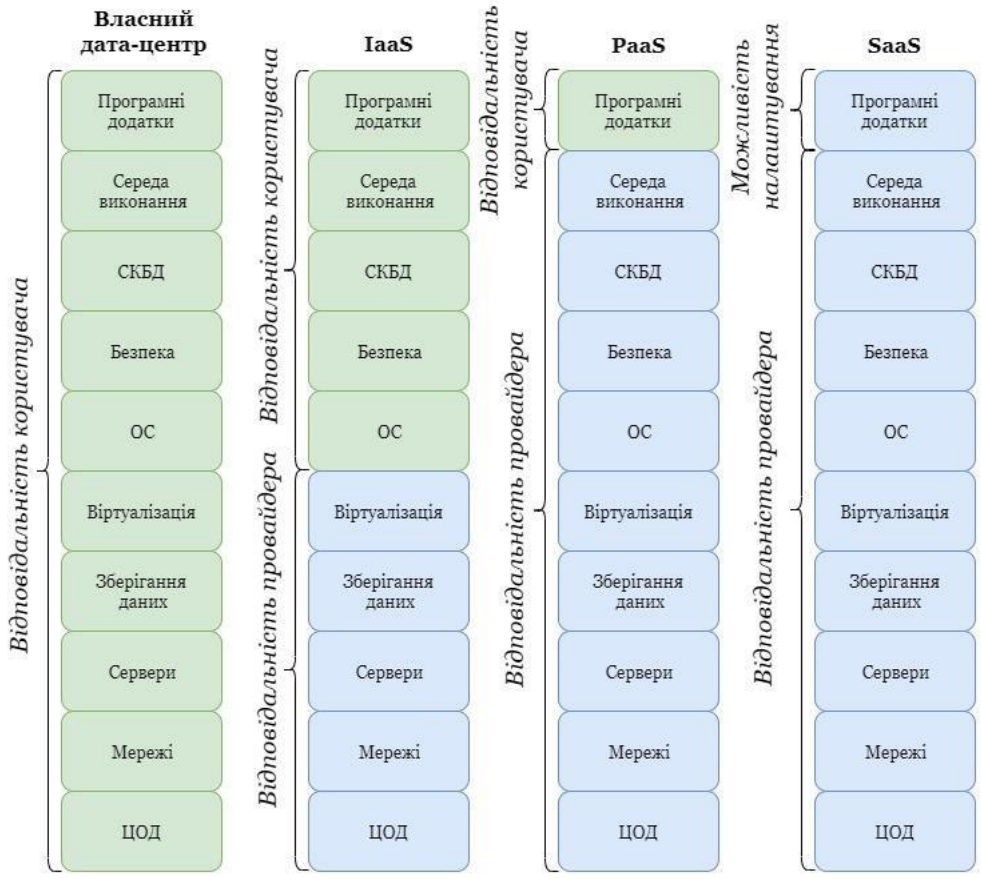


Рисунок 1.3 – Моделі хмарного обчислення. Межі відповідальності

Кожна з трьох моделей хмарного обслуговування можуть слугувати корисними інструментом для розробки та постійного використання бізнесом. Однак концепція хмарних обчислень полягає у розділені зони відповідальності між користувачем та постачальником послуг. В залежності від моделі обслуговування відповідальність збільшується як у випадку з IaaS чи зменшується – SaaS. Незалежно від рівня відповідальності застосування хмарних обчислень не повинно супроводжуватися зниженням захищеності даних. Провайдер як постачальник послуги повинен забезпечувати безпеку

користувача на своєму рівні, але все ж таки захист починається з методів та засобів з боку користувача.

Стосовно розгортання хмарної інфраструктури існують три найбільш поширені моделі, розглянемо кожну з них.

1. Локальна інфраструктура (англ. On-premises) – мається на увазі розгортання обчислювальних ресурсів на місцях (у власному дата-центрі) з використання засобів віртуалізації та керування ресурсами, дану модель також часто називають “приватною хмарою”. При локальному розгортанні немає можливості використовувати ряд переваг хмарних обчислень, але інколи саме функція виділення ресурсів за вимогою стає ключовою необхідністю для розгортання подібної системи. Зазвичай локальну інфраструктуру будують на вже існуючій IT-інфраструктурі з використанням функцій керування додатками та технологій віртуалізації для більш продуктивної роботи ресурсів.

2. Гібридні архітектури (англ. Hybrid) – розгортання гібридної архітектури дозволяє пов’язати інфраструктуру та додатки хмарних ресурсів з існуючими ресурсами, які не належать хмарі. Найпоширеніший спосіб розгортання подібної архітектури – комбінація хмари та існуючої локальної інфраструктури з метою розширення та нарощування функціональності системи організації шляхом підключення хмарних ресурсів до внутрішньої системи.

3. Публічна хмара (англ. Public cloud) – розгортання та робота всіх елементів додатків на основі систем хмарних обчислень повністю відбувається дистанційно. Додаток можна одразу створити в хмарі чи перенести з існуючої інфраструктури, далі користуючись всіма перевагами систем хмарних обчислень. Постачальник публічної хмари надає свої обчислювальні ресурси, такі як сервери та сховища через мережу Інтернет. Усі апаратні, програмні та інші послуги підтримуються постачальником.

1.2 Переваги та недоліки хмарних обчислень

Технології та концепції створюються для певних областей використання звертаючи перш за все увагу на різноманітні потреби та обмеження цих областей. Будь-яка технологія не прив'язується до конкретної області та може бути використана за бажанням розробника де завгодно на його розсуд, але слід зауважити, що кожний продукт має свої переваги та недоліки. Саме це поєднує існуючу технологію з певною областю використання та дає зрозуміти, що вона неприйнятна у іншій області. Хмарні обчислення, як модель забезпечення доступу до обчислювальних ресурсів, не є винятком, та може бути доцільною тільки у разі нівелювання її недоліків та використання усіх переваг.

Говорячи про переваги хмарних обчислень, можна виділити деякий перелік останніх.

1. Доступність: завдяки великому поширенню мережі Інтернет у наш час для багатьох користувачів хмарних сервісів зникає питання у розташуванні корпоративних площ, роблячи підприємство та співробітників більш мобільними, бо отримати доступ до обчислювальних ресурсів можна у будьякому місці де є інтернет. Враховуючи це керівництво підприємства має можливість не замислюватися над придбанням та розташуванням власних апаратних засобів у виді серверів та мережевого обладнання.

2. Надійність: постачальники хмарних обчислень зазвичай – великі компанії, які мають велику кількість дата-центрів у різних регіонах. Спеціально обладнаний ЦОД (центр обробки даних) забезпечує великий рівень надійності за рахунок переліку наступних засобів: резервні джерела живлення, спеціально-обладнанні приміщення, штат професійних працівників, система охорони, висока пропускна здатність мережі, резервування даних. Навіть у разі надзвичайної ситуації завдяки гарно

побудованій системі провайдери здатні відновити роботу обладнання автоматично.

3. Заощадження коштів: використання віртуальних серверів у хмарному середовищі зазвичай відбувається за моделлю погодинної орендної плати. Дана модель дозволяє заощадити кошти налаштувавши гнучку систему автоматичного масштабування, що забезпечить економію за рахунок звільнення надлишкових обчислювальних ресурсів. Компанії які не можуть собі дозволити придбати апаратні засоби, обслуговувати їх та утримувати команду спеціалізованих співробітників є одними з основних клієнтів для постачальників хмарних обчислень.

4. Безпека: дуже спірним є питання безпеки у хмарному середовищі, бо воно має великі ризики для бізнесу. Безпека у позитивному розрізі цього питання полягає у тому, що провайдер має спеціальні засоби для створення безпечної системи, запобіганню несанкціонованого доступу та стійкість до DoS-атак (англ. Denial of Service – відмова в обслуговуванні).

Деякі можливості, які вважаються перевагами можуть мати зворотній бік та перетворитися у недоліки для певного кола користувачів. Отже розглянемо далі перелік недоліків хмарних обчислень.

1. Доступність безперечно є плюсом, але інша її сторона для деяких користувачів стає недоліком, а саме у випадку поганого чи нестабільного інтернет з’єднання. Небезпека втрати підключення до серверів компанії може призвести до серйозних наслідків для бізнесу.

2. Аналогічно попередньому пункту безпека також може бути недоліком хмарних обчислень, бо який би надійний сервіс не пропонували провайдери, більш надійною, у деяких випадках, є власна інфраструктура. Той факт, що дані компанії знаходяться в обслуговуванні іншою структурою є вразливим місцем, та не виключається вірогідність незаконного

оприлюднення даних чи блокування роботи цілого дата-центру правоохоронними органами, коли незаконна діяльність обслуговується у цьому ж дата-центрі. Втрата даних, а особливо персональних даних користувачів може коштувати компанії великих збитків або навіть її ліквідації.

1.3 Масштабування у хмарному середовищі

В галузі інформаційних технологій (телекомунікацій та програмного забезпечення), масштабування є важливою властивістю будь-якої системи чи мережі, яка свідчить про здатність системи оброблювати збільшення робочого навантаження (підвищувати свою продуктивність) при додаванні ресурсів (зазвичай апаратних). Масштабованість можна оцінити через відношення збільшення продуктивності системи до приросту обчислювальних ресурсів. Чим ближче це відношення до одиниці, тим краще для системи та її подальшого масштабування [8].

Зазвичай власники веб-додатків починають думати про масштабування, коли один сервер не справляється з покладеною на нього роботою. Причина цього у тому, що робота будь-якого веб-сервера полягає у основному занятті комп’ютерів – обробці даних, а саме прийомі та обробці HTTP або інших запитів. Відповідно, маємо дві сутності – данні, які характеризуються своїм об’ємом, а також – обчислення, характеризовані їх складністю. Веб-сервер може не справлятися зі своєю роботою за великого об’єму даних, чи за причини великого обчислювального навантаження. Звісно описані ситуації стосуються сумарного навантаження – складність обробки одного запиту може бути несуттєвою, але велика їх кількість здатна перевантажити сервер, що приведе до його непрацездатності [9].

Розглянувши архітектурні рішення більшості веб-додатків можна виділити той факт, що все починається з доволі простої архітектури – один

веб-сервер, який займається обробкою усіх запитів користувачів. Запити можуть містити команду на передачу статичних файлів, які містяться на диску сервера (зображення – jpg, bmp; файли стилів – css; сценарії – js), також є запити, що вимагають певних обчислень – наприклад, формування HTMLсторінок. На одному фізичному сервері з веб-сервером, розташована і база даних. Недоліком такої архітектури є те, що різноманітні по своїй сутності запити (передача файлів зі сховища та обчислювальна робота сценаріїв) оброблюються одним і тим же веб-сервером. Обчислювальні запити потребують обчислювальних ресурсів, а також необхідність утримувати в оперативній пам'яті сервера багато інформації: інтерпретатор мови сценаріїв, сценарії та дані, з якими вони працюють. Запити на передачу статичних файлів зі сховища, навпаки, потребують невелику кількість обчислювальних ресурсів процесора, але можуть займати тривалий час, якщо клієнт має низьку швидкість з'єднання. Отже в результаті маємо ситуацію, коли тривалі запити гальмують систему і затримують в черзі складні обчислювальні запити. Для рішення даної проблеми можна скористатися розподіленням обов'язків по обробці різним веб-серверам.

З часом успішні веб-додатки набувають все більшої популярності серед все більшого числа користувачів, а отже кількість запитів зростає відповідно. Для будь-якої системи настає момент, коли необхідно реалізувати можливість масштабування, заради спроможності обробки запитів та надання якісного сервісу своїм користувачам. Масштабування може відбуватися за двома принципами: горизонтальним та вертикальним, розглянемо детальніше кожний із них далі.

1.3.1 Горизонтальне масштабування

Горизонтальне масштабування (англ. scaling out) – передбачає розподілення загальної системи на дрібніші функціональні компоненти та рознесення по окремим фізичним серверам з можливістю збільшення кількості серверів, які паралельно виконують одну функцію. Отже, горизонтальне масштабування означає можливість додання нових вузлів до системи, заради збільшення загальної продуктивності. Даний принцип може потребувати змін до архітектури веб-додатку, а саме програмного забезпечення, у разі рознесення функцій на окремі сервери, адже розподілені компоненти необхідно пов’язати новими з’єднаннями, налаштувавши їх взаємодію.

Окремий випадок горизонтального масштабування, до якого вдаються зазвичай, полягає у додаванні обчислювальних ресурсів шляхом збільшення кількості серверів-екземплярів (англ. instance) на яких розміщується однакова версія веб-додатку рис. 1.4.

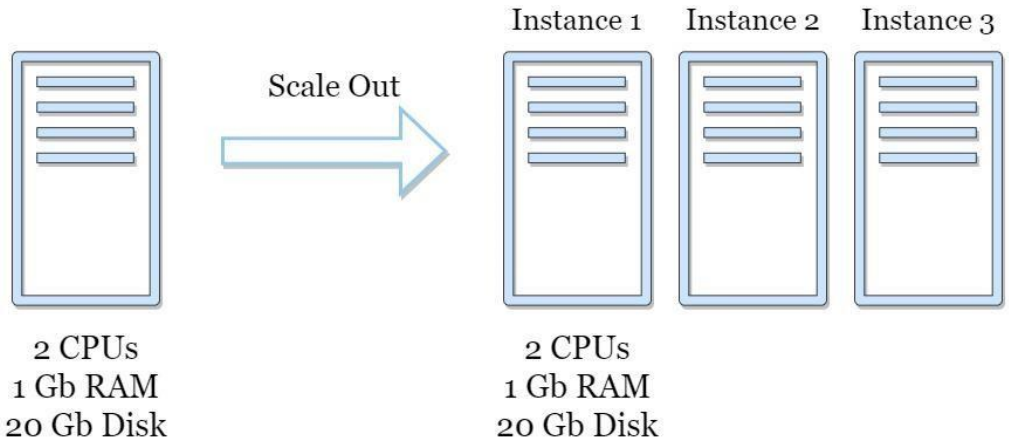


Рисунок 1.4 – Горизонтальне масштабування

У хмарному середовищі сервери-екземпляри являються віртуальними, але кінцевий користувач взаємодіє з ними як з реальними фізичними серверами. Для загальних потреб, коли веб-додаток не потребує спеціалізованих обчислень за допомоги апаратних ресурсів, віртуальні сервери виконують усі поставлені задачі. Якщо ж розробник потребує у тісній

взаємодії з апаратним забезпеченням сервера, то більшість хмарних провайдерів має можливість надати доступ до виділеного фізичного сервера, у вигляді окремої послуги. У хмарних середовищах за моделями IaaS чи PaaS, подібне масштабування можна налаштувати для виконання в автоматичному режимі. Так у разі недостатньої продуктивності одного екземпляру буде запущений додатковий, та навпаки зупинений у разі надмірної продуктивності. Налаштування автоматичного масштабування може відбуватися за різних параметрів, наприклад, відповідно відсотку навантаження процесора або кількісті запитів на сервер.

Як вже було сказано, горизонтальне масштабування передбачає розподілення системи по окремим фізичним чи віртуальним серверам, які забезпечують паралельну обробку запитів. Для побудови подібної структури необхідно включити додатковий компонент – балансувальник навантаження (англ. load balancer). Це компонент у вигляді окремого серверу зі спеціальним програмним забезпеченням, основна функція якого – розподілення запитів між кількома серверами-екземплярами.

Горизонтальне масштабування дозволяє збільшувати інфраструктуру для конкретного веб-додатку за потреби, що є позитивним для невеликих компаній, але розподілення функцій на окремих серверах може ускладнити управління такою інфраструктурою. Вразливим місцем у такій системі може бути пропускна здатність та затримка між компонентами, бо це приводить до низької продуктивності веб-додатку. Хмарні провайдери максимально спрощують налаштування системи, пропонуючи різноманітні сервіси, які допомагають розробникам.

1.3.2 Вертикальне масштабування

Вертикальне масштабування (англ. scaling up) – збільшення продуктивності кожного компонента системи з метою підвищення обчислювальної здатності всієї системи.

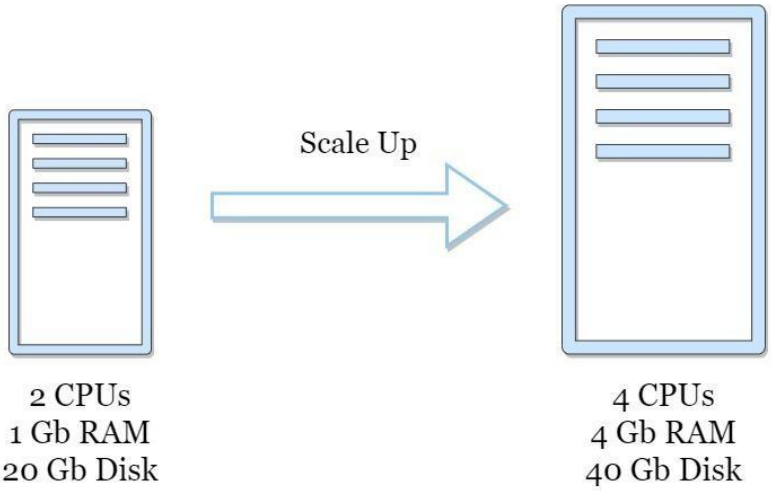


Рисунок 1.5 – Вертикальне масштабування

Згідно наданому поняттю, масштабованість передбачає заміну апаратних компонентів існуючого сервера, більш продуктивними та швидкими (процесори, оперативна пам'ять, носії інформації та ін.). Даний принцип не передбачає зміни та доопрацювання програмного забезпечення веб-додатку. Хоча вертикальне масштабування є найпростішим принципом підвищення продуктивності системи, але постійне оновлення неможливе, з часом такий підхід досягає своєю межі не надаючи необхідного росту.

Вертикальне масштабування у хмарному середовищі для користувача (розробника веб-додатку), здійснюється доволі просто. Якщо розробник користується певним віртуальним сервером, то для виконання вертикального масштабування необхідно змінити тип віртуальної машини. Зазвичай постачальники хмарних обчислень мають велику кількість різноманітних конфігурацій, які користувач може обрати для своїх потреб. Відповідно до збільшення продуктивності типу сервера, збільшується і погодинна плата.

Однак слід зазначити, що при суттєвому підвищенні типу віртуальної машини відбувається перенесення даних на новий більш продуктивний сервер, а старий видаляється.

Розглянувши два принципи масштабування можна зробити висновки стосовно необхідності використання одного чи іншого принципу. Звісно, що розпочати слід з вертикального масштабування, бо підвищення продуктивності сервера може вирішити проблему недостатньої обчислювальної здатності, не вдаючись до суттєвих змін у структурі та програмному забезпеченні. Однак з часом даний принцип не дає бажаного результату, в такому випадку слід переходити до горизонтального масштабування. Розширення обчислювальної здатності за рахунок збільшення кількості серверів-екземплярів та рознесення функціональних компонентів, що потребує модифікації архітектури додатку та певних змін до програмного забезпечення веб-додатку. Найкращим рішенням для масштабування є комбінування різних принципів.

1.4 Постачальники хмарних обчислень

Останні десять років хмарні обчислення тільки набувають своєї популярності, та ще зовсім не досягли піку. Впровадження мікросервісних (англ. microservices) та безсерверних (англ. serverless) архітектурних моделей, які широко використовуються у сучасній розробці веб-додатків, стимулює розвиток провайдерів хмарних обчислень. На ринку існує значна кількість постачальників, та за основою вони надають схожі послуги, які можуть відрізнятися тарифними планами чи характеристиками обладнання. Справжня боротьба за користувача відбувається за кількістю і якістю різноманітних сервісів. Більш розвинутий постачальник пропонує великий перелік доступного функціоналу, який може забезпечити швидку розробку, просте

розгортання та якісну підтримку встановленого веб-додатку. Розглянемо кілька найбільш відомих та широко поширених постачальників хмарних обчислень:

1. Amazon Web Services (AWS) – надає послуги хмарних обчислень (IaaS і PaaS) та API-інтерфейси для приватних осіб, компаній та урядів країн на основі принципу платної підписки [10]. Користувачі мають можливість створити інфраструктуру для власного проекту, включаючи вибір дата-центру для розміщення, створення приватних та публічних мереж, розміщення віртуальних серверів і багато іншого. Загалом AWS надає більш ніж 70 сервісів, що охоплюють широкий спектр, включаючи обчислення та зберігання даних, їхню передачу по мережі, аналітику, мобільні застосунки, інструменти для розробників і т. ін. Найпопулярнішими сервісами є Amazon Elastic Compute Cloud (EC2) – для створення віртуальних серверів, і Amazon Simple Storage (S3) – сховище даних. Дата-центри Amazon розташовані в багатьох географічних регіонах, у тому числі в США, Канаді, Німеччині, Ірландії, Сінгапурі, Токіо, Сіднеї, Пекіні, Лондоні і т. д.

2. Microsoft Azure – це хмарна платформа та інфраструктура корпорації Microsoft, призначена для розробників застосунків хмарних обчислень і покликана спростити процес створення веб-додатків [11]. Бізнес хмарних обчислень компанії Microsoft включає себе сервіси від Azure до Office365 (SaaS) та LinkedIn. Платформа Azure займає друге місце за популярністю після AWS, однак відома своїм темпом зростання та спектром доступних послуг. Останнім часом дана технологія більше розвиває сервіси надання послуг штучного інтелекту, аналітиці та Інтернеті речей. Головними сервісами є Azure Virtual Machine, SQL Database, Azure Blob Storage.

3. Google Cloud Platform (GCP) – набір хмарних послуг, який надається компанією Google та працює на одній інфраструктурі з основними продуктами компанії: Google Search та YouTube [12]. Платформа забезпечує

користувача рядом модульних хмарних служб за моделям IaaS та PaaS: обчислення, зберігання даних, аналіз даних та машинне навчання. Доволі популярним продуктом є Google App Engine (GAE) – платформа як послуга для розташування власних веб-додатків. Cloud Platform надає більшість найнеобхідніших послуг, аналогічних AWS та Azure, однак за популярністю і кількістю додаткових послуг значно відстає від них.

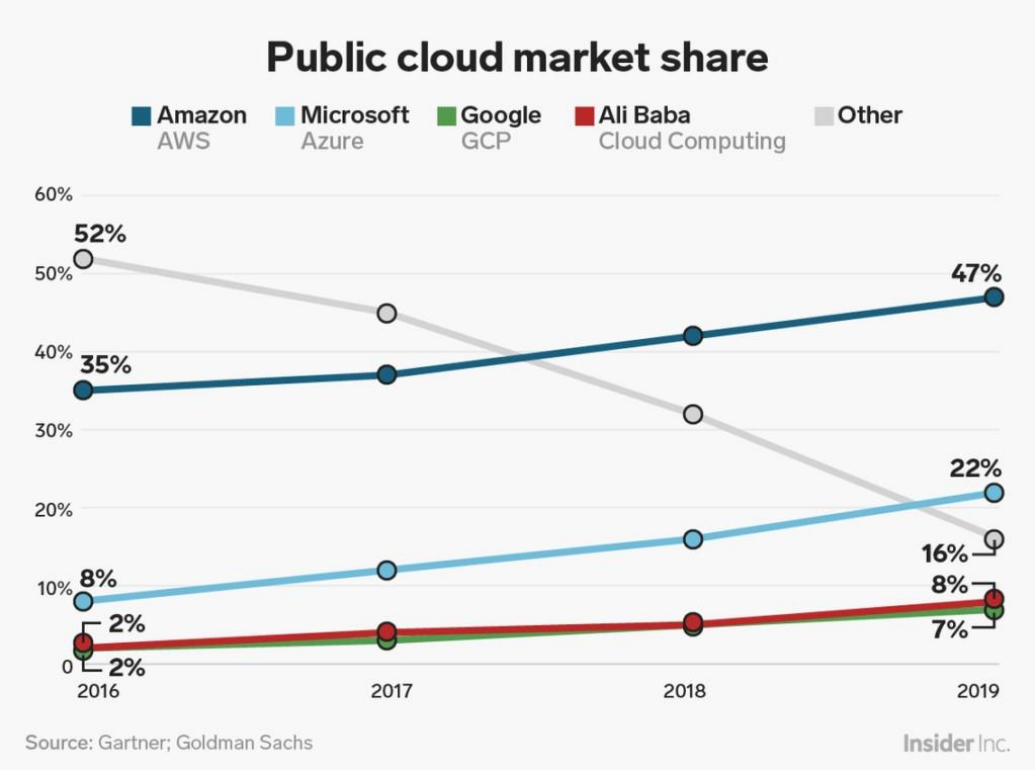


Рисунок 1.6 – Частка ринку хмарних обчислень

Виходячи з графіків зображених на рис. 1.6 можна зробити висновок, що популярність хмарних платформ трьох описаних раніше платформ невпинно зростає відповідно до росту частки на ринку [13]. Найсильнішим гравцем є Amazon Web Services, який значно випереджає своїх конкурентів. Також слід відмітити платформу Ali Baba, яка більш націлена на Азійський ринок, але конкурує з Google Cloud Platform на одному рівні.

При вирішенні питання вибору інфраструктури для розміщення вебдодатку слід зважувати всі переваги та недоліки, порівнювати наявні

сервіси. В такому випадку стане у нагоді таблиця у якій зібраний перелік усіх доступних сервісів найпопулярніших хмарних постачальників [14]. Також досить важливо звертати увагу на наявність довідкової інформації по налаштуванню та керуванню сервісами, як офіційної так і аматорських. Чим більша аудиторія користувачів та спеціалістів певного хмарного провайдера, тим легше знаходити відповіді на суто спеціалізовані запитання, що також підвищує привабливість у разі вибору постачальника.

2 БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

Сучасні веб-сайти з високою популярністю повинні обробляти сотні, тисячі, якщо навіть не мільйони одночасних запитів від користувачів, та повертати відповідь виду: тексту, зображень, відео чи даних – коректно та швидко. Для економічно ефективного масштабування відповідно з такими об’ємами запитів, сучасні кращі практики комп’ютингу потребують додавання більшої кількості серверів, тобто горизонтального масштабування. Для реалізації даного типу масштабування системи, необхідно використовувати додатковий вузол, який забезпечить регулювання трафіку між групою серверів – балансувальник навантаження.

В термінології комп’ютингу балансування навантаження (англ. load balancing) – забезпечує розподілення робочих навантажень по кільком обчислювальним ресурсам таких, як: комп’ютери, кластери, мережеві з’єднання, центральні процесори чи диски зберігання даних. Балансування навантаження направлене на оптимізацію використання ресурсів, максимізацію пропускної здатності, зменшення часу відгуку та ухилення від перевантаження будь-якого окремого ресурсу. Використання декількох компонентів з балансування навантаження замість одного монолітного

компонента, може підвищити надійність та доступність ресурсів за рахунок надмірності. Балансування навантаження зазвичай передбачає використання виділеного програмного чи апаратного забезпечення, таких як багаторівневих комутатор чи як процес серверу системи доменних імен.

Стосовно комп’ютерних мереж, то балансування навантаження полягає у ефективному розподіленню вхідного мережевого трафіку між групою бекенд серверів, також відомими як ферма серверів чи пул серверів. Отже можна виділити три функції, які виконує балансувальник навантаження.

1. Ефективне розподілення клієнтських запитів чи навантаження на мережу між декількома серверами. Алгоритми балансування навантаження є активною областю досліджень і варіюються від спрощених, таких як випадковий вибір чи циклічний алгоритм, до більш складних – які враховують перемінну затримку та навантаження на сервіс.

2. Забезпечення високої доступності та надійності. Балансування навантаження дозволяє забезпечити надлишковість ресурсів для кращої стабільності роботи сервісу. Тобто у разі непрацездатності одного ресурсу, навантаження може бути направлене на резервний вузол. Перевірка працездатності (англ. health checking) – це процес, за допомогою якого балансувальник виявляє чи доступний певний сервер для обслуговування трафіку. Перевірка працездатності зазвичай поділяється на дві категорії. Активна – коли балансувальник навантаження періодично відправляє команду ping (наприклад, HTTP-запит кінцевій точці /healthcheck) на сервер та використовує це для оцінки працездатності. Пасивна – балансувальник навантаження визначає стан здоров'я сервера з основного потоку даних. Наприклад, балансувальник L4 може вирішити, що сервер нездоровий, якщо було три помилки з'єднання підряд. Аналогічно балансувальник рівня L7 робить такі висновки при послідовному отриманні коду HTTP 503.

3. Забезпечення гнучкості при додаванні чи видаленні серверів за запитом, в залежності від кількості вхідного трафіку.

2.1 Пояснення термінології

Для подальшого розгляду концепції та принципу роботи балансування навантаження, необхідно пояснити деякі терміни з якими виникає деяка плутанина у суспільства та різне тлумачення між документацією різних розробників.

1. Поняття, що зазвичай називають сервер (англ. server), вузол (англ. node), хост (англ. host) – має на увазі фізичний чи віртуальний сервер, який отримує трафік від балансувальника навантаження. Пов’язано з IP-адресою фізичного сервера, а у разі відсутності балансувальника навантаження є адресою на яку посилається ім’я сервера (наприклад, www.example.com).

2. Інша концепція виражається терміном – елемент, член або сервіс (нажаль, іноді розробники використовують термін вузол). Елемент, зазвичай, більш визначений ніж хост, оскільки він включає у себе IP-адресу разом з TCPпортом існуючого веб-додатку, який буде отримувати трафік. Наприклад, хост з ім’ям www.example.com може дозволити доступ до адреси 172.16.1.10, яка представляє хост та має веб-додаток, працюючий на TCP-порту 80, що робить 172.16.1.10:80 – адресою сервісу. Відмінність сервера (хоста) від сервісів які запущені на ньому, дозволяють балансувальнику навантаження індивідуально взаємодіяти з розгорнутими веб-додатками, а не з базовим обладнанням чи гіпервізором в центрі обробки даних або хмарі. Хост 172.16.1.10 може мати більше однієї служби: HTTP, FTP, DNS і т. ін. Визначивши кожний сервіс окремо, балансувальник може застосувати унікальне балансування та виконувати моніторинг працездатності.

3. Пул, кластер або ферма серверів. Балансування навантаження дозволяє організаціям розподілити вхідний трафік веб-додатків по декільком внутрішнім адресам, включаючи розгортання в публічних чи приватних хмарах. Тому існує концепція яку називають – кластер (англ. cluster), пул (англ. pool) або ферма (англ. farm). Кластер представляє собою набір схожих сервісів, доступних на любій кількості хостів. Наприклад, усі сервіси, які постачають веб-сторінку компанії, будуть зібрані в кластер, названий “Вебсторінка компанії”, а всі сервіси електронної комерції зібрані під назвою “Екомерція”.

4. Віртуальний сервер (англ. virtual server) – це проксі фактичного серверу. У поєднанні з віртуальною IP-адресою це точка з’єднання вебдодатку з зовнішньою мережею.

2.2 Різновиди балансувальників навантаження

Балансування навантаження реалізується за допомогою різного виду пристроїв чи засобів, але можна виділити три різновиди балансувальників навантаження.

Контролери доставки додатків (англ. Application Delivery Controller (ADC)) – обчислювальні машини зі спеціалізованими процесорами та завантаженим пропріетарним програмним забезпеченням. Дані пристрої є багатоцільовими, але цільовими методами забезпечення продуктивності зазвичай включають: балансування навантаження, кешування, стиснення даних та розвантаження обробки протоколу SSL Як правило такі високопродуктивні пристрої здатні надійно обробляти трафік у декілька гігабіт інформації, саме завдяки спеціалізованим процесорам. Вбудовані

засоби віртуалізації, дозволяють створювати численні екземпляри віртуальних балансувальників навантаження на одному й тому ж обладнанні, це забезпечує більш гнучку архітектуру, розраховану на багато користувачів та повну ізоляцію орендаторів. Апаратні засоби такого типу потребують значних фінансових витрат, тому зазвичай їх використовують у дата-центрах великі компанії-постачальники послуг. Однак існують недоліки у використанні таких засобів: потреба у спеціально кваліфікованих спеціалістах; масштабування відбувається шляхом встановлення додаткових такої самої апаратури. Прикладами існуючих ADC є рішення від компаній: Cisco Systems, F5 Networks, TP-Link, Barracuda.

Програмні засоби (англ. software) – спеціалізоване програмне забезпечення, може бути встановлене на будь-який сервер не залежно від виробника. Програмні балансувальники навантаження менш дорогі та більш гнучкі у використані. Можливість масштабування нічим не обмежена, та можливість розгортання також, наприклад, програмний забезпечення може працювати на віртуальному хмарному сервері. Гнучкість програмних рішень полягає у можливості різноманітного налаштування: зміни алгоритмів балансування, використання різних методів балансування, або зовсім зміна програмного забезпечення від іншого розробника. Найбільш широко використовуваними програмні балансувальники навантаження є доступні по вільній ліцензії з відкритим вихідним кодом, наприклад: HAProxy, Nginx, Traefik, Balance, Seesaw.

Хмарні балансувальники навантаження – це є підвид програмних засобів, але головна відмінність полягає у тому, що балансування надається як послуга постачальниками хмарних обчислень (AWS, Google Cloud, Azure, DigitalOcean і т. ін.) та користувач на має замислюватись як реалізована ця послуга: апаратним забезпеченням чи програмним додатком. Сервіс надається за моделлю оплати – “плата за користування”, яка робить цю послугу

найнижчою по ціні серед інших різновидів балансування. Хмарні балансувальники можуть використовуватися як додаткові у парі з особисто розгорнутим програмним чи апаратним засобом.

2.3 Мережеві рівні балансування навантаження

Процедура балансування навантаження відбувається за допомогою цілого комплексу алгоритмів та методів, часто рішення з балансування поділяють на дві категорії: L4 (Layer 4) та L7 (Layer 7), які відповідають транспортному та прикладному рівням мережевої моделі OSI.

Балансувальники транспортного рівня L4, вважаються найпростішими: клієнт звертається до балансувальника, який перенаправляє запит одному з доступних серверів на обробку.

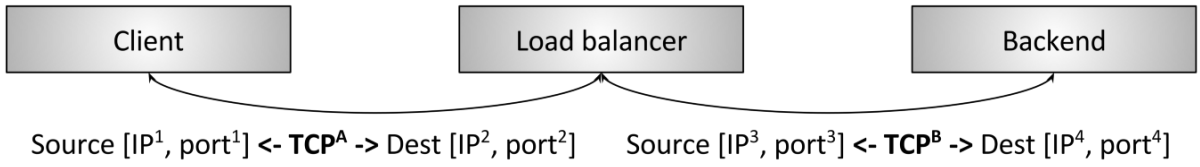


Рисунок 2.1 – Традиційний L4 TCP балансувальник

На рис 2.1 зображено традиційний балансувальник навантаження L4. В даному випадку клієнт встановлює TCP-з’єднання з балансувальником навантаження. Балансувальник завершує це з’єднання (тобто, відповідає на запит SYN), обирає сервер для обробки поточного запиту та встановлює нове TCP-з’єднання з сервером (відправляючи новий запит SYN). Виходячи з цього, L4 балансувальник працює тільки на рівні з’єднання/сеансу L4 TCP/UDP протоколів, грубо переміщаючи байти даних, гарантуючи, що усі байти певного сеансу будуть оброблятися на одному і тому ж сервері. В цьому випадку баласнувальник не оброблює пакети, а лише перевіряє IP-адресу та

номер порту, у випадку виконання одного з встановлених правил розподілу IP та портів, трафік просто перенаправляється на обрані вузли. Непередбачена і особлива перевірка доступності вузлів, виконується проста перевірка на доступність IP та портів, у випадку відкритого порту сервер вважається доступним і до нього передаються запити. Дані веб-додатку ніяк не аналізуються, цими даними можуть бути HTTP, MongoDB чи будь-який інший протокол додатку.

Слід зазначити, що балансувальник на рівні L4 робить одне вихідне TCP-з'єднання для кожного вхідного, приводячи до двох вхідних и двох вихідних з'єднань у ланцюгу клієнт-сервер. Розглянемо ситуацію, коли клієнт А відправляє 1 запит в хвилину, а клієнт Б – 50 запитів у секунду по вже встановленому з'єднанню. В результаті сервер, який був обраний для клієнта А, буде оброблювати приблизно в 3000 разів менше, ніж сервер обраний для клієнта Б. Тому безперервне TCP-з'єднання порушує перш за все мету балансування навантаження і є недоліком.

Балансування рівня L7, яке працює в режимі "розумного проксі", аналізує запити користувачів та перенаправляє їх на різні сервери в залежності від характеру потрібного контенту. Даний режим балансування дозволяє запускати декілька серверів веб-додатків під одним доменним ім'ям та портом. Балансування навантаження рівня L7 потребує більшої витрати обчислювальних ресурсів ніж балансування L4, однак аналізування додаткового контексту запитів дозволяє виконувати розподілення між серверами більш ефективно. Прикладом виконання балансування L7 є модуль Upstream у ПЗ Nginx, або pgpool – який є проміжним шаром між клієнтом та сервером СКБД PostgreSQL. За його допомогою можна розподіляти запити по серверам баз даних в залежності від їх змісту, наприклад, запити на читання передаються на один сервер, а запити на запис – на інший.

Балансування навантаження на рівні L7 ймовірно у майбутньому повністю замінить балансування на рівні L4 для зв'язку між сервісами, але в даний час балансування навантаження рівня L4 досі є важливою часткою дворівневої архітектури балансування інтернет-трафіку, яка поєднує можливості обох рівнів. Переваги розташування окремих засобів балансування навантаження на рівні L4 перед балансувальником рівня L7 при крайовому розташуванні.

Оскільки балансувальні пристрої L7 виконують значно більш складний аналіз пов'язаний з високорівневий протоколами, то продуктивність для перетворення та маршрутизації трафіку може бути недостатньою. Тому в таких випадках, доцільніше використовувати оптимізований балансувальник навантаження рівня L4, який спеціалізований саме для цих цілей. Цей факт, зазвичай, робить цей пристрій кращим місцем для обробки окремих видів DoSатак (наприклад, SYN-флуду [15], загальних атака на передачу пакетів і т. ін.).

Балансувальники навантаження рівня L7, як правило, більш активно розвиваються, встановлюються частіше та мають більше програмних помилок, ніж перевірене часом балансування рівня L4. Наявність у топології системи балансувальника рівня L4, котрий спроможний виконувати перевірку працездатності подальших балансувальників та у випадку їх нестабільної роботи, маршрутизує помилки та аномалії, що призводить до більшої стабільності системи загалом.

2.4 Принцип роботи балансувальника навантаження

Для загального розуміння принципу роботи балансування навантаження, слід розглянути базову транзакцію того, як веб-додаток постачається користувачу.

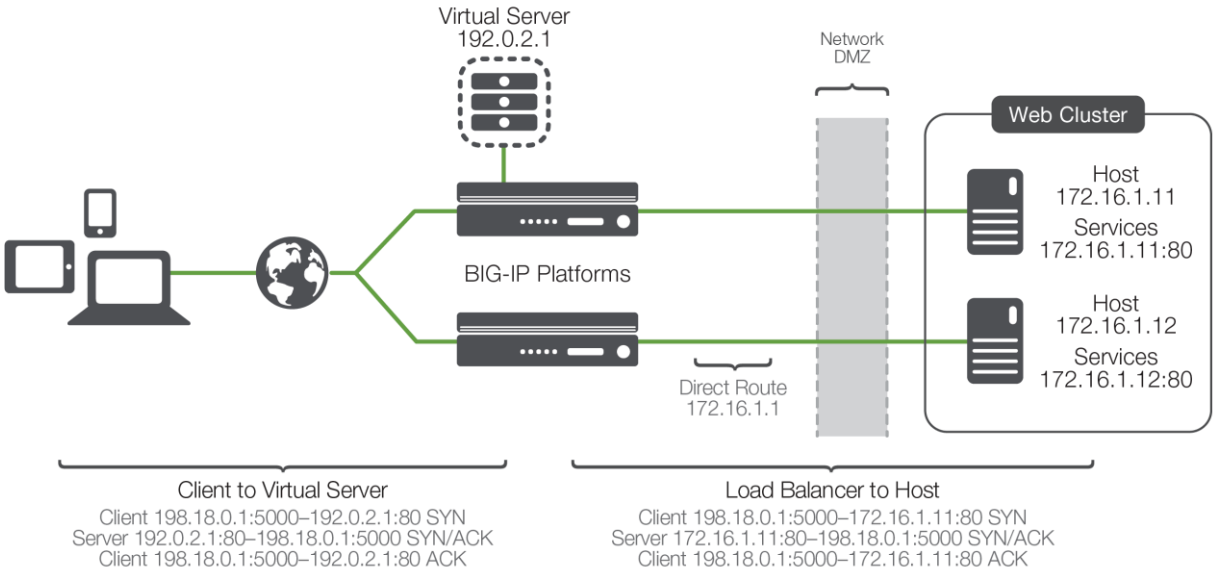


Рисунок 2.2 – Базова транзакція балансування навантаження

На рис. 2.2 зображена проста топологія системи постачання веб-додатка з балансуванням навантаження. Функцію балансування навантаження виконує апаратний ADC пристрій, який зазвичай знаходиться між клієнтом та хостами, які поставляють послуги необхідні користувачам. Як і в більшості випадків, зв’язаних з постачанням веб-додатків, таке розташування балансувальника не є правилом, але вважається кращою практикою при будь-якому типі розгортання. Припустимо, що на ADC вже налаштований віртуальний сервер, який вказує на кластер з двох точок обслуговування. Базова транзакція постачання веб-додатку до користувача виглядає наступним чином [16]:

1. Клієнт намагається підключитися до сервісу з веб-додатком.

2. Балансувальник приймає з’єднання та після вирішення на який хост буде відправлений запит, замінює цільову IP-адресу (можливо разом з номером порту) на внутрішній IP сервісу.

3. Хост приймає запит і відповідає на адресу відправника, через зворотній маршрут.

4. Балансувальник перехоплює надісланий пакет від хоста і тепер, замінює вихідний IP на відповідний IP віртуального сервера.

5. Клієнт отримує пакет-відповідь, вважаючи, що він прийшов з віртуального серверу та продовжує процес.

Даний простий приклад, показує короткий перелік дій, який відбувається при з’єднанні клієнта із сервісом веб-додатку. Оскільки клієнт нічого не знає про внутрішню будову системи, то усі запити до сервісу транслюються за допомогою технології NAT (Network Address Translation), яка змінює цільовий IP на внутрішній IP цільового сервісу. На зворотному шляху, вихідна адреса знову транслюється, щоб клієнт отримав пакет від очікуваної адреси. Якщо не змінити внутрішній IP сервісу на IP віртуального серверу про який знає клієнт, то отримавши такий пакет, він буде відхилений через те, що адреса на яку був відправлений запит не збігається з вихідною адресою отриманого пакету.

2.4.1 Моніторинг працездатності (Health check)

При роботі подібної системи, може виникнути ситуація, що хост обраний балансувальником для обробки запиту не відповідає при цьому спроба підключення до такого сервера завершується помилкою. Звичайно подібна ситуація не є допустимою, бо через це не забезпечується висока доступність. Тому більшість технологій балансування навантаження передбачають деякий рівень моніторингу працездатності для того, щоб

виявити недоступність серверу перед тим, як буде зроблена спроба передати запит. Існує декілька рівнів моніторингу стану здоров’я серверу, кожний із них збільшує рівень деталізації та направленості спостереження. Базовий монітор виконує команду ping на адресу сервера. Якщо хост не відповідає на запит, то вважається, що всі сервіси які наявні на цьому сервері, вірогідно, не працюють та повинні бути видалені зі списку доступних. Нажаль, навіть якщо хост відповідає на команду ping, це не означає, що самий сервіс працює. Високорівневі монітори працездатності не тільки забезпечують більшу впевненість у доступності реальних сервісів (у відмінності від хосту), але і дозволяють балансувальнику навантаження розрізняти декілька сервісів на одному сервері. Балансувальник розуміє, що при недоступності однієї служби, інші на тому ж хості можуть працювати нормально та повинні використовуватися, а не переводитися у недоступних.

Кожний віртуальний сервер має окремий виділений кластер сервісів, який створює список з серверів та розміщених на них сервісів. Моніторинг працездатності корегує цей список, щоб створити перелік серверів – ”доступних в даний момент” які мають необхідний сервіс. Саме цей змінений список, з якого балансувальник обирає хост для передачі запитів. Вибір серверу для передачі запита на обробку, залежить від алгоритму розподілення навантаження зв’язаного з цим конкретним кластером. Деякі з таких алгоритмів використовують принципи найменшої кількості з’єднань, динамічне розподілення, чи просте циклічне перелічення, в якому балансувальник передає запити серверам по черзі згідно списку доступних серверів, досягнувши останнього – повертається на початок. Хоча даний принцип розподілення дуже простий та передбачуваний, вважається, що всі з’єднання будуть мати однакове навантаження та тривалість обробки, однак це не завжди правда. Більш складні алгоритми використовують дані

моніторингу, такі як кількість наявних з’єднань, використання хоста, а також реальний час відгуку, щоб обрати найбільш доступний сервер з доступних.

Виконання головної задачі балансування навантаження – розподілення запитів на доступні сервери у групі, забезпечуються різноманітними алгоритмами балансування. Встановлення з’єднання з сервером веб-додатку це тільки частина роботи, адже необхідно підтримувати підключення клієнта та спостерігати за навантаженням подальших запитів. Як правило, є дві специфічні проблеми з обробкою подальшого трафіку після розподілення навантаження: підтримка з’єднання та постійність.

2.4.2 Підтримка сеансу з’єднання (Session Persistence)

Підтримка сеансу з’єднання користувачів між запитами, цей процес називається – збереженням сеансу (англ. session persistence) або прив’язкою до сервера. Все частіше зустрічається ситуація, коли клієнт використовує декілька короткоживучих TCP-з’єднань (наприклад, порт 80: HTTP) для виконання однієї задачі. В деяких випадках, наприклад при звичайному перегляді веб-сторінок, це не має значення та кожний новий запит може відправлятися балансувальником на будь-який з серверів. Однак, у випадку спеціалізованих веб-додатків, наприклад електронної комерції, дуже важливо, щоб усі запити від користувача приходили до одного і того ж сервісу, та не були перенаправлені через балансування.

Існує декілька способів вирішення проблеми збереження сеансу, які відрізняються в залежності від протоколу та очікуваних результатів. Наприклад, у сучасних HTTP-транзакціях сервер може передавати “keepalive”, що перетворює декілька короткотривалих з’єднань в одне довготривале. Однак це забезпечить лише невелике полегшення, так як по мірі збільшення використання веб-сервісів та мобільних сервісів, підтримання усіх з’єднань

відкритими довше ніж це необхідно збільшить навантаження на ресурси всієї системи. Ось чому, сьогодні заради масштабованості та переносимості, багато організацій-розробників переходять до створення вебдодатків без збереження поточного стану, замість цього використовують API (Application Programming Interface). За основою це означає, що сервер не зберігає інформацію про сеанс клієнта, в цілях зменшення навантаження на ресурси. В таких випадках використовується передача ідентифікаторів сеансу, а також шляхом концепції постійності.

Однією з основних форм постійності з'єднання є прив'язка до вихідної адреси, яка включає простий запис вихідної IP-адреси вхідних запитів та хоста сервісу, до якої вони були направлені балансувальником, та відправлення усіх наступних транзакцій на той же хост. Це можна реалізувати двома способами: за допомогою ідентифікаторів сеансів SSL або файлів cookie (кешу). Постійність SSL, відстежує сеанс за допомогою ідентифікаторів з'єднання, що робить можливим розпізнавання, необхідного клієнту сервера, балансувальником навіть у випадку зміни IP-адреси клієнта. Постійність на основі кешу дозволяє додати файл cookie на комп'ютер клієнта, щоб однозначно ідентифікувати сеанс, а після посилатися на цей файл у запитах для встановлення з'єднання з необхідним сервером.

Інтелектуальні можливості балансувальників навантаження дозволяють організаціям досліджувати пакети даних та створювати таблиці постійних з'єднань, використовуючи унікальну інформацію, таку як ім'я користувача. Однак, розробники веб-додатків повинні гарантувати, щоб дана інформація була присутня у кожному запиті, бо в противному випадку трафік буде збалансовано, що може привести до порушення роботи додатку.

2.4.3 Надлишкові балансувальники навантаження (High Availability)

Проблемою будування системи з балансуванням навантаження є факт, що вузол балансування вважається єдиною точкою відмови (англ. single point of failure). Оскільки, весь трафік системи проходить через балансувальник навантаження, то будь-яка його нестабільність призводить до проблеми доступу до веб-додатку. Для вирішення проблеми єдиної точки відмови, можна використовувати надлишкові балансувальники гарантуючи високу доступність (англ. high availability) до сервісів, які підключені паралельно до першого, таким чином формуючи кластер де кожний з них контролює стан інших. Така схема підключення працює за принципом активний / пасивний. Основний балансувальник обробляє трафік веб-додатку, а у випадку його несправності, DNS повинен перевести користувачів на інший (пасивний) балансувальник навантаження. Оскільки зміна DNS може займати значний час для розповсюдженні в Інтернеті та для автоматичного переключання при несправності, багато адміністраторів таких систем допускають гнучке перепризначення IP-адрес, наприклад “floating IPs”. Перерозподіл IP-адрес за необхідністю усуває проблеми розповсюдження та кешування, які характерні для зміни DNS, надаючи статичну IP-адресу, яка може бути легко перепризначена за необхідності. Доменне ім’я може залишатися прив’язаним до однієї IP-адреси, в той час як сама IP-адреса переміщається між балансувальниками.

2.5 Топології систем балансування навантаження

Системи балансування навантаження будуються за різноманітних топологій, розташовуючи елементи одним чи іншим способом можна отримати певні переваги та недоліки. Далі розглянемо декілька існуючих топологій, слід зауважити кожна з них може використовуватися як для баласнувальників рівня L4, так і для – L7.

2.5.1 Проксі по середині (Middle proxy)

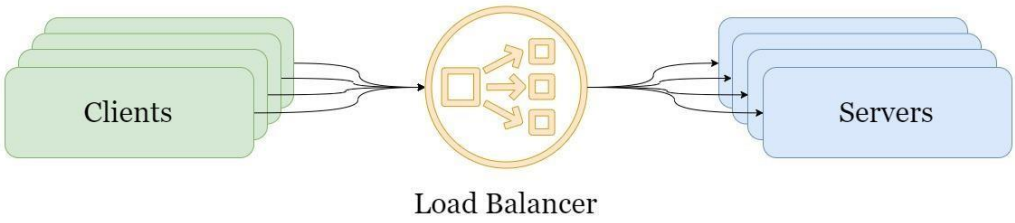


Рисунок 2.3 – Топологія балансування навантаження “проксі по середині”

Топологія проксі посередині, яка показана на рис. 2.3, являється найбільш відомим та використовуваним способом балансування навантаження, може бути реалізована використовуючи.

1. Апаратні пристрої від Cisco, Juniper, F5 і т. ін.;
2. Програмне забезпечення, наприклад, HAProxy, Ngnix та Envoy;
3. Хмарні програмні рішення, такі як Amazon ALB та NLB, Amazon Cloud Load, Microsoft Azure Load Balancer, Google Cloud Load Balancing;

Перевагою такої топології – простота побудови та використання. Клієнт підключається до балансувальника шляхом DNS та взаємодіє з системою не підозрюючи про її внутрішню будову. Слабким місцем даної топології є той факт, що проксі (балансувальник навантаження) між клієнтом та сервісом, навіть у випадку кластеризації може вийти з ладу, що призводить до непрацездатності всієї системи. Також слід зазначити, що проксі є своєрідним чорним ящиком, що затрудняє роботу з ним, так як важко дати однозначну відповідь щодо місця ініціації ймовірної проблеми: на стороні клієнта, фізичної мережі, проксі чи сервісі?

2.5.2 Крайове розташування (Edge proxy)

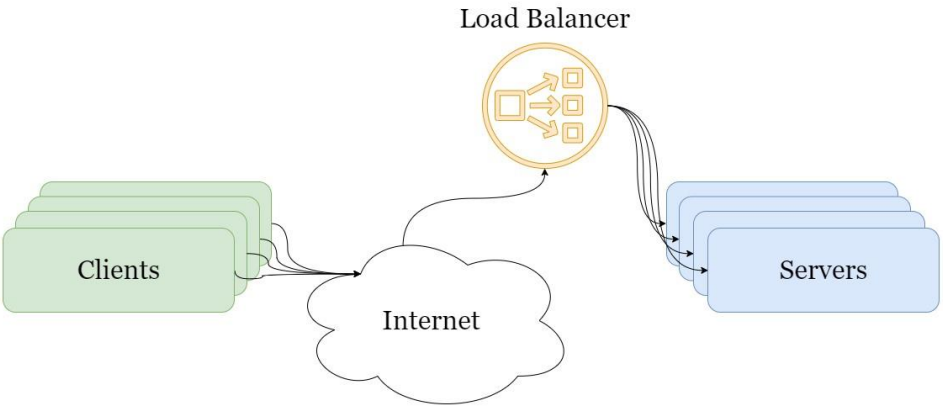


Рисунок 2.4 – Крайове розташування проксі

На рис. 2.4 приведена топологія з крайовим розташування балансувальника навантаження, яка насправді є варіантом топології з серединним розташуванням, описаним раніше, але з відмінністю у тому, що балансувальник доступний через Інтернет. В даному випадку балансувальник повинен надавати додаткові функції так звані “Api gateway”, наприклад: TLS термінування [17], обмеження швидкості, аутентифікація і складна маршрутизація трафіку. Переваги і недоліки аналогічні до топології серединного розташування.

2.5.3 Використання вбудованої клієнтської бібліотеки

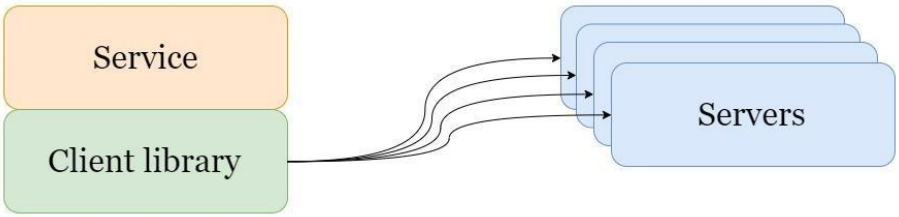


Рисунок 2.5 – Балансування навантаження через вбудовану клієнтську бібліотеку

Щоб запобігти присутніх топології проксі по середині проблем відмови та масштабування, більш складні інфраструктури вбудовують балансуваник навантаження безпосередньо в служби через бібліотеку, як зображено на рис. 2.5. Бібліотеки сильно відрізняються за підтримуваними функціями, але деякі з них є найбільш функціональними – Finalge, Hystrix та gRPC [18].

Основною перевагою рішень на базі використання бібліотек є те, що функціональність балансувальника навантаження розподіляється кожному клієнту, тим самим усуваючи проблеми відмови балансувальника та масштабування. Головний недолік такого рішення полягає у тому, що бібліотека повинна бути реалізована на кожній мові програмуванні, яка використовується системою. Розподілені архітектури стають все частіше багатоязиковими, тому реалізація об’ємної бібліотеки може коштувати дуже дорого. Розгортання оновлення такої бібліотеки приводить до проблем зв’язаних з використанням різних версій цієї бібліотеки.

2.5.4 Sidecar проксі-сервер

Варіантом топології з використанням вбудованої клієнтської бібліотеки є топологія з використанням шаблону проектування додатків – Sidecar проксісервер, зображена на рис. 2.6.

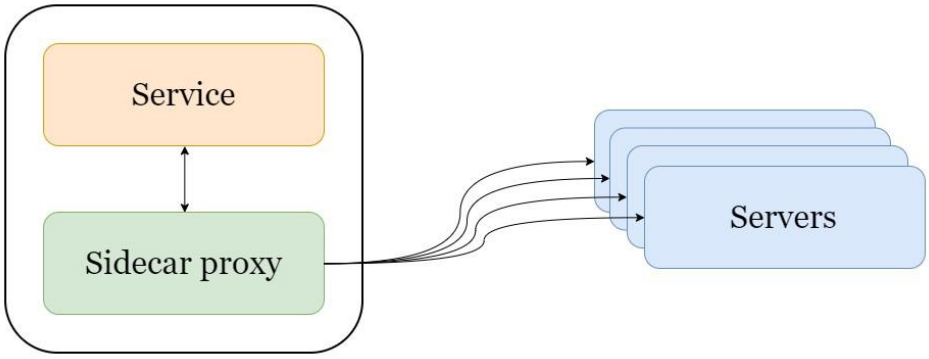


Рисунок 2.6 – Балансування навантаження через проксі-сервер sidecar

Останнім часом дана топологія популяризована як “service mesh” (Service mesh – це виділений шар інфраструктури для забезпечення безпечної, швидкої та надійної взаємодії між сервісами). Ідея sidecar проксі-сервера полягає у тому, що за рахунок невеликого зниження затримки через перехід на інший процес, усі переваги підходу з використанням вбудованої бібліотеки можуть бути використанні без будь-якої прив’язки до окремої мови програмування. Sidecar проксі-сервер абстрагує деякі функції, такі як між сервісні комунікації, моніторинг та безпека, від основної архітектури, щоб спростити процес спостереження та обслуговування веб-додатку в цілому. Аналогічно, як коляска прикріплюється до мотоцикла, sidecar проксі-сервер прикріплюється до основного веб-додатку для розширення та додавання нової функціональності. Найбільш популярні sidecar проксі балансувальники:
Envoy, Ngnix, HAProxy та Linkerd.

2.5.5 Глобальне балансування навантаження (GSLB)

Глобальне балансування навантаження (англ. Global server load balancing) – це процес розподілення навантаження між серверами, розташованими у різних географічних точках. GSLB керує веб-трафіком та доставкою веб-додатків через декілька центрів обробки даних (ЦОД) або шляхом приватних чи публічних хмарних інфраструктур в різноманітних географічних областях. Завантаження додатків в кожній області зазвичай виконується локальними балансувальниками навантаження, тому запити клієнтів відправляються на близько розташовані сервери, щоб забезпечити мінімальну затримку та максимальну продуктивність.

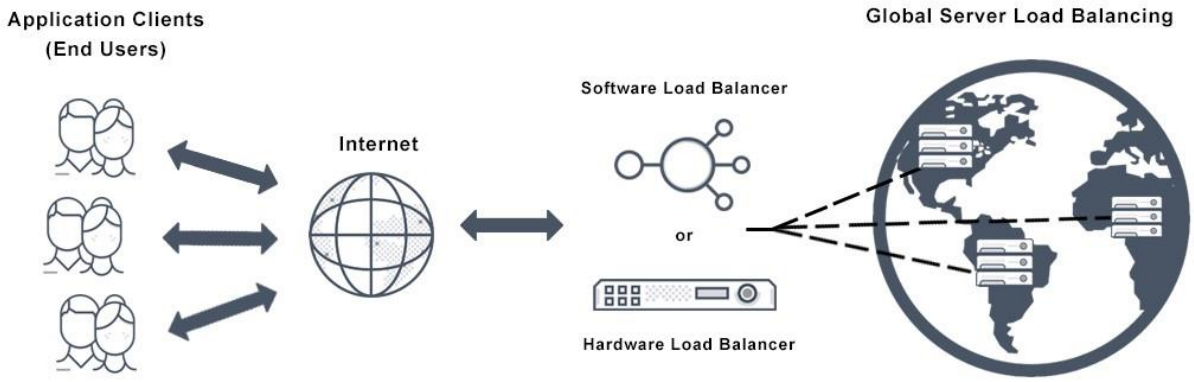


Рисунок 2.7 – Глобальне балансування навантаження

Розглянемо перелік дій, які відбуваються коли клієнтський HTTP-запит відправляється веб-додатку, який використовує глобальне балансування навантаження. Спочатку балансувальник отримує вихідну IP-адресу користувача, аналізуючи інформацію про місцезнаходження клієнта. Одночасно перевіряється працездатність кінцевих вузлів для оцінки продуктивності та швидкості відповіді. Після цього запит передається на локальний балансувальник найоптимальнішого для користувача регіону.

Аварійне відновлення є головною причиною того, що велика кількість компаній використовують серверні ресурси у декількох місцях. В найбільш розповсюдженій конфігурації данні обслуговуються з одного активного місцеположення, але дублюються в одному чи декількох резервних місцях. Роль глобального балансування в даному випадку полягає в виявленні збоїв на активних вузлах та автоматична переадресація запитів на резервні.

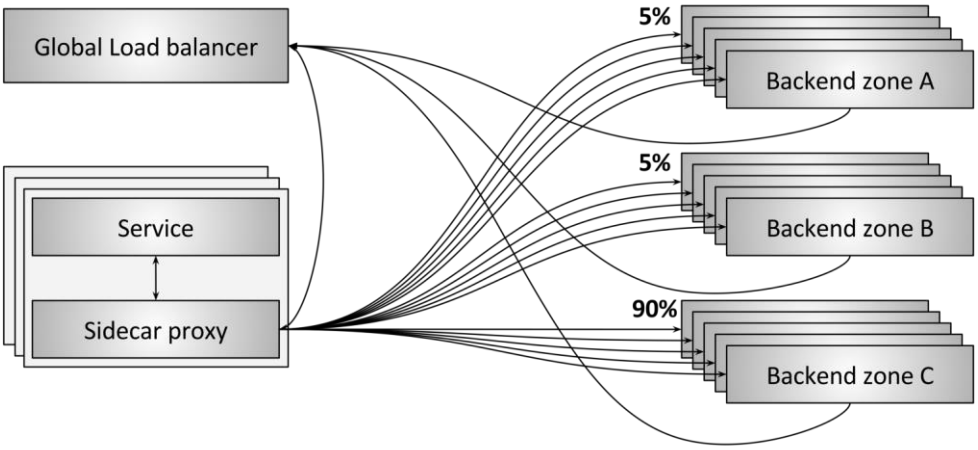


Рисунок 2.8 – Приклад глобального балансування

На рис. 2.8 зображений приклад глобальної системи балансування навантаження, в даному прикладі можна розглянути декілька речей.

1. Кожний проксі-сервер sidecar зв’язується з серверами в трьох різних зонах (A, B та C).

2. 90% трафіку направляється в зону C, та по 5% в зони A та B.

3. Sidecar проксі-сервери та внутрішні сервери (вузли) повідомляють про періодичний стан глобального балансувальника навантаження. Це дозволяє приймати рішення по розподіленню трафіку враховуючи інформацію про затримки, завантаження, поточні збої та ін.

4. Глобальний балансуваник навантаження періодично налаштовує кожний sidecar проксі-сервер на основі поточної інформації про маршрутизацію.

В подальшому розвитку глобального балансування навантаження для популярних веб-додатків, які використовуються великою кількістю користувачів, будуть можливі складні функції, які не можуть бути виконані одним балансувальником самостійно, наприклад.

1. Автоматичне розпізнавання та маршрутизація від зональних збоїв.

2. Застосування глобальних політик безпеки та маршрутизації

3. Виявлення та зменшення аномалій трафіку, включаючи DDoS-атаки,

з використанням машинного навчання та нейронних мереж.

4. Забезпечення централізованого інтерфейсу користувача та візуалізації, які дозволяють інженерам розуміти всю розподілену систему взагалі та керувати нею.

2.6 Методи та алгоритми балансування навантаження

Головна мета балансування навантаження – розподілення навантаження веб-додатку на декілька серверів для можливості обробки більшої кількості запитів. Для виконання своєї роботи балансувальники використовують алгоритми розподілення, хоча основна мета всіх алгоритмів полягає у тому, щоб рівномірно розподілити навантаження по мережі, але базуючись на принципах роботи вони підрозділяються на дві категорії: статичні та динамічні алгоритми балансування навантаження.

Статичні алгоритми рівномірно розподіляють трафік між серверами. Розподілення здійснюється з використанням попередніх знань про системні ресурси та продуктивність процесорів. Тому, для балансування непотрібна інформація стосовно поточного стану системи. Кожний вузол системи отримає однакову кількість запитів, незалежно від його поточного завантаження. Такий підхід пришвидшує процес балансування навантаження та не потребує високопродуктивних обчислювальних засобів для його реалізації. Однак такі алгоритми не аналізують вхідний трафік та не перевіряють доступність серверів, що приводить до небажаних проблем з продуктивністю та стабільністю роботи системи.

Динамічні алгоритми балансування навантаження працюють аналізуючи інформацію про поточний стан системи, та на основі цього

приймають рішення щодо розподілення навантаження між вузлами. В динамічних алгоритмах визначається сервер з низьким рівнем завантаження, який саме обирається для балансування. Стан системи перевіряється у реальному часі, тому балансування відбувається більш продуктивно, однак перевірка стану приводить до додавання службового трафіку всередині системи. Динамічний підхід застосовується для розподілених систем, таких як хмарні обчислення.

Існує велика кількість алгоритмів балансування навантаження, які відрізняються продуктивністю роботи та складністю реалізації. Перш за все перед застосуванням певного алгоритму, необхідно визначитися з цілями, які планується досягти для конкретного проекту. Основні цілі, які переслідуються при використанні балансування навантаження.

1. Економічна ефективність: балансування навантаження допомагає підвищити продуктивність системи при невеликих затратах.

2. Масштабованість та гнучкість: Система для якої використовуються алгоритми балансування навантаження, напевно з часом зміняться за кількістю обчислювальних вузлів. Таким чином, алгоритм повинен обробляти подібні ситуації, тому він повинен бути гнучким та масштабованим.

3. Ефективність: необхідно забезпечувати зайнятість серверів обробкою вхідних запитів на 100%, бажано не допускати ситуації простою обчислювальних можливостей серверів, але на практиці 100%-ва зайнятість це еталон який важко досягти.

4. Скорочення обробки: необхідно забезпечувати якомога менший час між початком обробки запиту та його виконанням.

Оцінка та характеристика алгоритмів відбувається завдяки переліку метрик та політик, для визначення ефективності алгоритмів балансування навантаження застосовуються певні метрики, які описані далі [19].

1. Пропускна здатність – міра, яка відображає кількість виконаних запитів. Як правило, очікується, що висока пропускна здатність покращує продуктивність системи.

2. Час міграції – це час, необхідний для міграції процесів чи ресурсів з одного вузла на інший. Для досягнення високої продуктивності в системі, час міграції повинен бути мінімальним.

3. Накладні витрати – об’єм навантаження, який створюється алгоритмом балансування навантаження. Зазвичай очікується, що накладні витрати будуть мінімальними.

4. Час відповіді – це час, який витрачається алгоритмом балансування навантаження, щоб надати відповідь на запит в системі. Для кращої продуктивності системи час відповіді повинен бути мінімальним.

5. Масштабованість – це здатність системи працювати рівномірно, коли в мережі додано обмежена кількість вузлів.

6. Відмовостійкість – це міра, яка представляє поведінку системи у випадку виникнення збоїв. Алгоритм балансування навантаження повинен бути відмовостійким.

7. Використання ресурсів – алгоритми балансування навантаження повинен максимально використовувати доступні ресурси системи.

Розглянемо перелік політик, які створюють систему принципів алгоритмів балансування навантаження, необхідних для адекватного та продуктивного розподілу трафіку між наявними обчислювальними вузлами [20].

1. Інформаційна політика – визначає яка інформація потребується для роботи алгоритму, та яким чином вона збирається. Також визначає проміжок часу, коли дана інформація збирається.

2. Політика ініціації – визначає період часу, коли операція балансування навантаження починає керувати трафіком.

3. Типізація ресурсів – визначає усі типи ресурсів, які доступні під час балансування навантаження.

4. Політика розташування – використовує усі результати політики типізації ресурсів. Застосовується у пошуку додаткових ресурсів для сервера чи отримувача.

5. Політика вибору – використовується для визначення задачі (запиту), яка переходить від перенавантаженого вузла до вільного.

Функція балансування навантаження може забезпечуватися різноманітними методами, але слід виділити два різних підходи до забезпечення балансування. По-перше, балансування може відбуватися прямо на стороні клієнта, якому доставляється список IP-адрес доступних серверів, для подальшого балансування використовуючи певний алгоритм розподілення задач. Другим підходом до балансування, та більш застосовуваним у практиці побудови систем доставки веб-додатків, є балансування на стороні серверів. Реалізовується даний підхід за допомогою пристроїв чи програмних засобів із застосуванням тих самих алгоритмів розподілення задач. Отже розглянемо деякі найбільш відомі та популярні алгоритми та методи балансування навантаження.

2.6.1 Round-Robin та Weighted Round-Robin

Алгоритм Round-Robin відноситься до категорії статичних, та не перевіряє поточний стан системи. Принцип роботи алгоритму полягає у циклічному переборі виходячи із пулу доступних вузлів. Даний концепція знаходить своє застосування у різноманітних конфігураціях, наприклад, у виді

алгоритму розподілення задач для апаратних балансувальників навантаження чи за використання DNS-серверу для балансування.

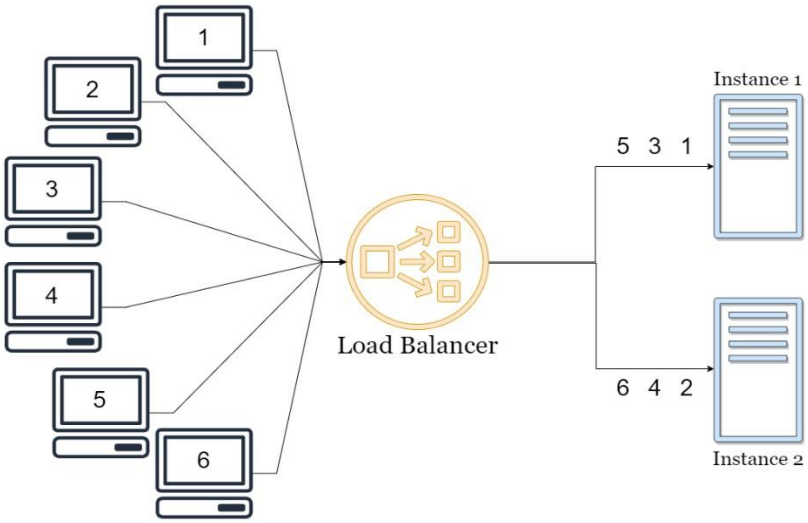


Рисунок 2.10 – Рівномірне розподілення запитів за алгоритмом Round-Robin

Найбільш розповсюдженим методом є Round-Robin DNS, який заснований на тому, що специфікація DNS дозволяє створювати декілька однакових записів с відмінними IP-адресами. Наприклад, можна створити два записи для доменного ім’я “example.com” з різними IP-адресами, які належать різним серверам.

Таблиця 2.1 – Перелік DNS записів

Ім’я запису	Тип запису	Значення
example.com	A	10.10.1.1
example.com	A	10.10.2.2

Додатково для роботи цього методу, необхідно щоб на DNS-сервері була встановлена опція “Round-Robin”, яка забезпечить розподілення трафіку. В результаті, при кожному новому запиті на ім’я “example.com” будуть

видаватися користувачеві різні IP-адреси серверів, що приведе до рівномірного розподілення підключень між вузлами.

Однак, при усіх перевагах виду простоти реалізації та дешевизни, мається ряд обмежень. По-перше, даний метод не передбачає ніякої перевірки доступності серверів, тобто вузол може вийти з ладу, але DNS все одно буде передавати його IP-адресу користувачам. По-друге, ніяк не враховується число поточних активних сесій на тому чи іншому сервері. Може виникнути ситуація, коли на одному з вузлів, відкритих сесій значно більше ніж на інших, при цьому запити все одно будуть розподілятися рівномірно. По-третє, DNS не враховує до якого з серверів користувач був підключений у минулу сесію. Тому можливо, що при кожному новому підключенні до веб-додатку, буде відкриватися нова сесія на іншому сервері, що може привести до неактуальності кеш-даних [21].

Модифікацією даного алгоритму є Weighted Round-Robin, який дозволяє присвоювати ваговий коефіцієнт кожному вузлу відповідно до його продуктивності та потужності. Вагове циклічне розподілення схоже на просте циклічне розподілення в тому сенсі, що спосіб за яким запити передаються вузлам, залишається циклічним, але з додатковою умовою.

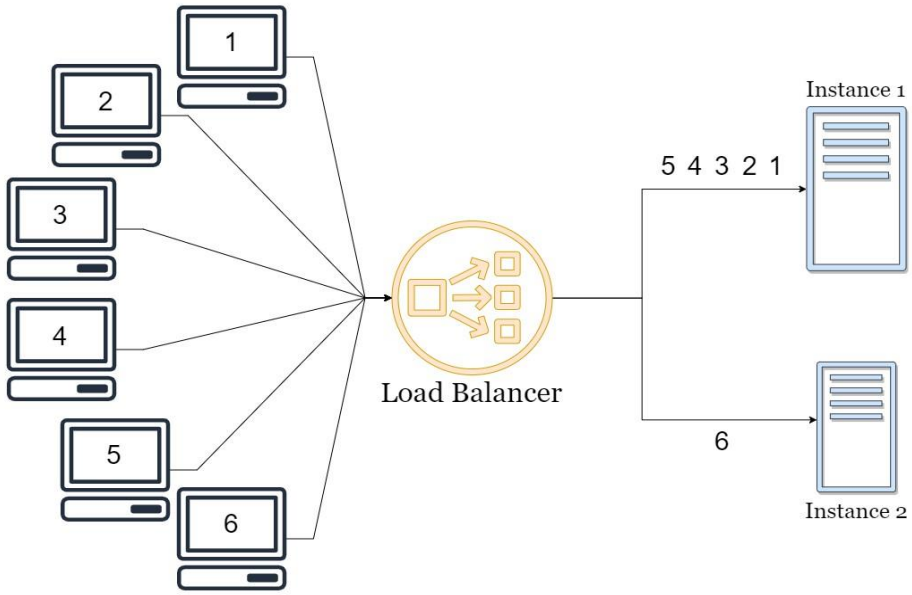


Рисунок 2.11 – Розподілення запитів за алгоритмом Weighted Round-Robin

Серверу з більш високими характеристиками та більшим ваговим коефіцієнтом, буде передано більша кількість запитів. Адміністратор мережі повинен налаштувати балансувальник навантаження, встановивши необхідні коефіцієнти, зазвичай які пропорційні фактичній потужності.

Вибір алгоритму Weighted Round-Robin не обов'язково може бути пов'язаний з використанням вузлів різної потужності. Можлива ситуація, коли необхідно щоб один сервер отримував меншу кількість запитів, ніж інший такий самий сервер по характеристикам, за причини того, що на першому працює критично важливе для бізнесу програмне забезпечення, перевантаження якого неприпустиме.

2.6.2 Least Connection

Невинятковими можуть бути випадки, коли два однакових за потужністю сервера в кластері, але все ж таки один сервер є більш навантаженим ніж інший.

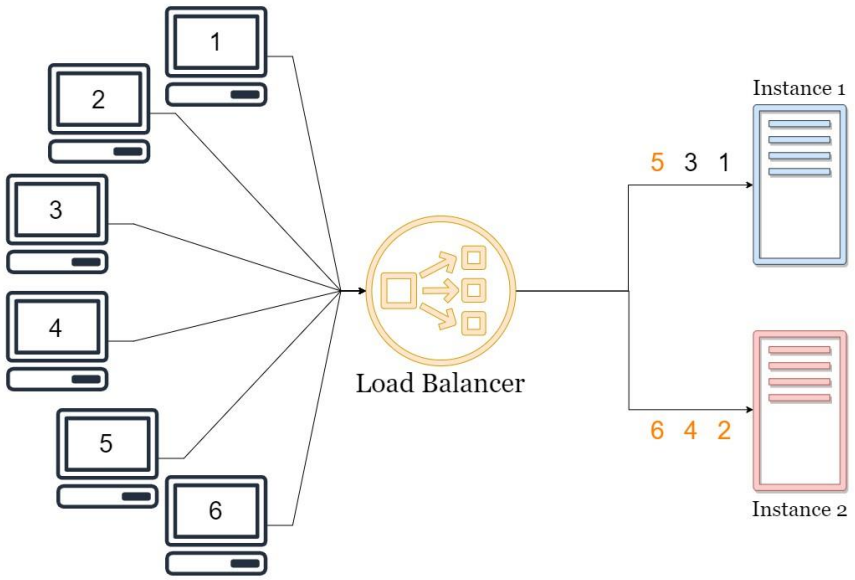


Рисунок 2.12 – Перевантаження серверу з довготривалими з’єднаннями

Однією із можливих причин є те, що користувачі залишаються підключеними до одного сервера набагато довший час, аніж – до іншого. Дана ситуація може привести до того, що загальна кількість поточних з’єднань на сервері-2 буде накопичуватися, в той час коли з’єднання на сервері-1 (з підключення та відключенням користувачів за короткий час) практично не змінюються. В результаті, ресурси сервера-2 можуть працювати швидше. На рис. 2.12 зображений даний випадок, де користувачі 1 та 3 вже завершили з’єднання, а 2, 4, 5 та 6 все ще підключені.

В подібних випадках, застосовується алгоритм розподілення Least Connection. Цей алгоритм враховує кількість поточних з’єднань до кожного вузла. Коли клієнт відправляє запит, балансувальник навантаження намагається визначити, який з серверів має найменшу кількість з’єднань, щоб назначити йому нове [22].

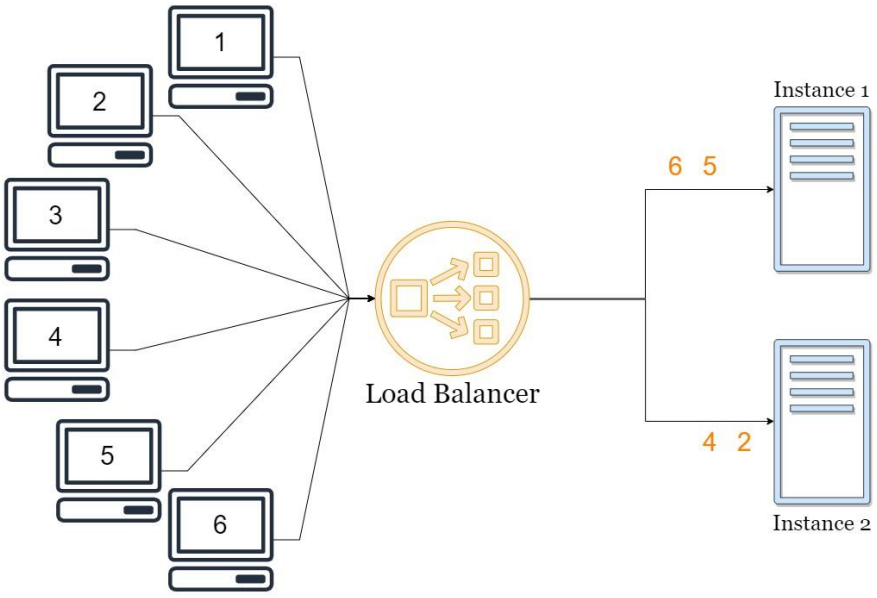


Рисунок 2.13 – Балансування за алгоритмом Least Connection

Таким чином, якщо продовжити розглядання попереднього прикладу, то клієнт 6 намагаючись підключитися після того, як 1 та 3 вже завершили з’єднання, але 2 та 4 все ще підключені, балансувальник навантаження назначить клієнта 6 на сервер-1 замість сервера-2.

Існує модифікація до алгоритму Least Connection, яка за своїм принципом аналогічна модифікації Weighted Round-Robin. Алгоритм Weighted Least Connection розрахований на кластери які складаються з різних за своєю продуктивністю серверів. Кожний з вузлів отримує свій ваговий коефіцієнт, який необхідно встановлювати адміністратору системи. Балансувальник навантаження, який працює за алгоритмом Weighted Least Connections, враховує ваговий коефіцієнт та кількість активних з’єднань підключених до сервера.

2.6.3 Source IP hash та Sticky Sessions

Алгоритм розподілення вхідних запитів Sticky Sessions забезпечує концепцію балансувальників навантаження під назвою – підтримка сеансу з’єднання (англ. session persistence), яка була описана у розділі 2.4.2. Сутність алгоритму полягає у тому, що з’єднання передаються на один і той самий сервер групи. Даний алгоритм знайшов застосування у веб-сервері Nginx. Сесія користувача може бути закріплена за певним сервером на основі методу Source IP hash, який об’єднує вихідну та цільову IP-адресу в запиті для створення хеш-ключа, згодом назначається певному серверу. Це дозволяє клієнту відновити розірване з’єднання з тим самим сервером, який обробляв його запити спочатку. Якщо закріплений за конкретною адресою сервер недоступний, то запит буде перенаправлений на інший.

Використання даного методу може бути ускладнено у випадку використання клієнтом динамічної IP-адреси. В ситуації, коли велика кількість

запитів проходить через один проксі-сервер, балансування навряд чи можна назвати продуктивним.

Висновки до розділу 2

Балансування навантаження стає невід’ємною частиною сучасних систем та мереж, які оброблюють великий трафік задач та запитів. Використання хмарних обчислень, може знизити витрати та підвищити продуктивність навіть при піковому навантаженні. На початку технологія балансування навантаження була направлена на розподілення робочого навантаження по мережі та забезпечення доступності до веб-додатків та сервісів. Однак, з розвитком технології, балансувальники стали засобами для доставлення додатків, забезпечуючи високу доступність та безпечність критично важливих сервісів певної організації. В розділі 2 – “Балансування навантаження” було розглянуто перелік різноманітних топологій за використанням балансувальників навантаження, також зроблений огляд основних функцій, алгоритмів та методів.

3 РОЗГОРТАННЯ ВЕБ-ДОДАТКУ НА БАЗІ ХМАРНОЇ ІНФРАСТРУКТУРИ

Кожний з існуючих відомих постачальників хмарних обчислень надає своїм користувачам величезний перелік різноманітних сервісів, які допомагають розгортати власні програмні проекти, обслуговувати, та спостерігати за продуктивністю і коректністю їх роботи. Найвідоміші три платформи: Amazon Web Services, Microsoft Azure та Google Cloud Platform – є конкурентами на ринку хмарних обчислень та мають схожу функціональність своїх сервісів, відрізняючись в деталях. Для подальшого аналізу балансування навантаження на базі хмарних обчислень, необхідно побудувати систему та розгорнути тестовий веб-додаток. В якості постачальника хмарних обчислень був обраний Amazon Web Services, на якому буде розгорнута WordPress система керування контентом (англ. Content Management System – CMS). Протягом реалізації даної системи, надаються відомості щодо сервісів які використовуються та поширені практики їх застосування.

3.1 Amazon Web Services

Постачальник хмарних обчислень Amazon Web Services (AWS) надає послуги за моделлю IaaS, що дозволяє створювати власні різноманітні конфігурації системи для розгортання веб-додатків. Більше ніж 160 сервісів станом на 2019 рік, розподілені за категоріями базуючись на області використання, серед них наявні сервіси для: обчислення, зберігання даних, керування базами даних, машинного навчання, інтернету речей та ін. Також AWS надає вичерпну документацію для кожного сервісу, що забезпечує

можливість швидкого та осмисленого ознайомлення з функціональністю та методами використання.

Для подальшого розгортання веб-додатку, необхідно розглянути базові сервіси, які будуть використовуватися.

1. Elastic Compute Cloud (EC2) – це веб-сервіс, що забезпечує масштабовану обчислювальну потужність в хмарній інфраструктурі Amazon Web Services. Використання EC2 позбавляє від необхідності оплати наперед за використання обладнання, так як відбувається погодинна оплата, тому користувачі мають змогу швидше розробляти та розгортати додатки за рахунок економії. EC2 дозволяє використовувати велику кількість віртуальних серверів, необхідних будуючи масштабовану та високонавантажену архітектуру. Можливість збільшення чи зменшення кількості серверів в автоматичному режимі дає змогу справлятися зі змінами в потребах чи зростанням трафіку, знижуючи необхідність у прогнозуванні навантаження.

2. Simple Storage Service (S3) – являється сховищем даних в Інтернеті. Amazon S3 забезпечує доступ до надійної, швидкої та недорогій інфраструктурі зберігання даних. Даний сервіс призначений спрощення обчислень в масштабах мережі, дозволяючи зберігати та витягувати будь-яку кількість даних в будь-який час. S3 має можливість зберігати резервні дані для надійного та швидкого відновлення роботи системи в разі збоїв сервера чи додатку.

3. Elastic Load Balancer (ELB) – автоматично розподіляє вхідний трафік додатків по декільком вузлам, таким як екземпляри EC2, контейнери, IPадреси чи функції Lambda. ELB здатен оброблювати різноманітне навантаження трафіку веб-додатка в одній зоні доступності чи декількох, що забезпечує більшу надійність. Даний сервіс пропонує три типи балансувальників навантаження, які характеризуються високою доступністю,

автоматичним масштабуванням та надійним захистом, необхідних для забезпечення відмовостійкості додатків.

4. Virtual Private Cloud (VPC) – віртуальна приватна хмара, в якій розміщуються екземпляри EC2. Використовуючи Amazon VPC, розробники можуть запускати ресурси AWS в топології віртуальної мережі. VPC надає повний контроль над віртуальною мережею, включаючи вибір діапазонів IPадрес, створення підмережі та налаштування таблиць маршрутизації та мережевих шлюзів.

5. Relational Database Service (RDS) – це веб-сервіс, який спрощує налаштування, роботу та масштабування реляційних баз даних в хмарній інфраструктурі AWS. Забезпечує економічну, масштабовану ємність при автоматизації трудомістких адміністративних задач, таких як підготовка обладнання, налаштування бази даних, виправлення помилок та резервне копіювання. Це дозволяє розробнику сконцентрувати увагу на веб-додатку, задля забезпечення продуктивності, високої доступності, безпеки та сумісності, яких необхідно досягти.

6. Route 53 – це високодоступний та масштабований веб-сервіс DNS. Даний сервіс можна використовувати для виконання трьох головних функцій в будь-якій комбінації: реєстрація доменів, маршрутизація DNS та перевірка працездатності вузлів.

7. Elastic Beanstalk – це веб-сервіс для розгортання та масштабування веб-додатків та сервісів, розроблених з використанням Java, .NET, PHP, Python і т. ін., на широковживаних веб-серверах – Apache, Nginx та IIS.

Amazon Web Services використовується розробниками по всьому світу, що пояснює існування переліку регіонів – окремих географічних областей для розміщення інфраструктури (дата-центрів). Широке розповсюдження дає змогу клієнтам обрати найближчий до них регіон, що максимально скоротить затримки в мережі для кінцевих користувачів додатку. Amazon має свої

датацентри у 23 регіонах, що охоплюють Північну та Південну Америку, Азію та Західну Європу.

Зона доступності (англ. Availability zone – AZ) – є частиною AWS регіону, яка представляє собою ізольовані місця – центри обробки даних. Кожний регіон має декілька дата-центрів, тобто декілька зон доступності. При проектуванні власної інфраструктури розробники використовують різні зони для забезпечення резервного копіювання та створення ефективної моделі відмовостійкості, в чому полягає основна концепція хмарних обчислень. Дублювання обчислювальних вузлів EC2 веб-додатку у різних дата-центрах, забезпечує безперервну роботу в разі виникнення збоїв роботи вузлів однієї з зон доступності.

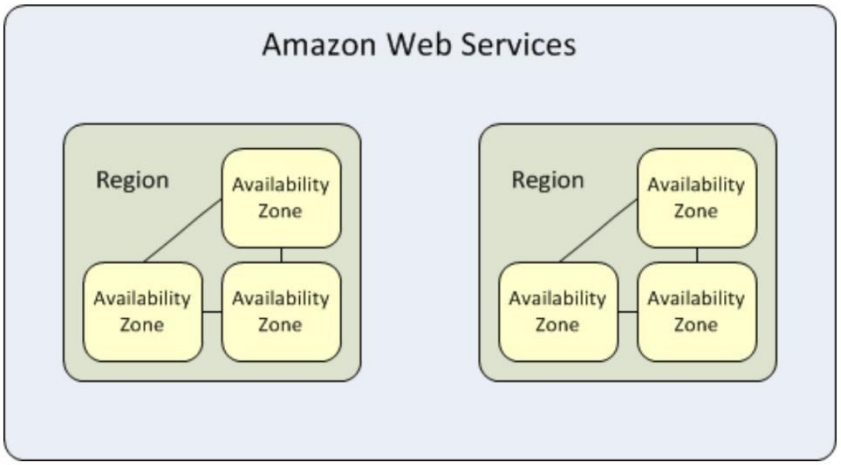


Рисунок 3.1 – Візуальне уявлення зон доступності в регіонах

Для побудови та керування інфраструктури, користувачу пропонується AWS Management Console – веб-додаток для керування веб-службами Amazon. Зручний веб-інтерфейс, що містить список різноманітних сервісів на вибір, дозволяє спостерігати за станом системи, будовою та змінювати її. Окрім вебдодатку, також розробник має можливість взаємодії через інтерфейс командної строки (англ. Command Line Interface – CLI) – це уніфікований

інструмент для керування сервісами AWS. Завдяки AWS CLI можна автоматизувати використання сервісів за допомогою сценаріїв (англ. script).

В подальших підрозділах описується особливості побудови системи та розгортання веб-додатку при різних конфігураціях.

3.2 Побудова інфраструктури

Як вже зазначалося раніше Elastic Load Balancer – це сервіс, який надає послуги програмного балансувальника навантаження, що забезпечує розподілення мережевого трафіку між вузлами інфраструктури AWS. Даний сервіс має три різновиди балансувальників навантаження: Application Load Balancer, Network Load Balancer та Classic Load Balancer, кожний з яких має певні можливості за рахунок різної реалізації. Для подальшого тестування методів балансування необхідно побудувати систему з балансуванням навантаження між декількома вузлами, а потім розгорнути веб-додаток.

3.2.1 Application Load Balancer

Балансувальник навантаження слугує єдиною точкою з’єднання для клієнтів, що забезпечує доступність веб-додатку. Автоматичне масштабування дозволяє додавати та видаляти екземпляри серверів за потреби, в залежності від кількості оброблюваного трафіку, не порушуючи загальний потік запитів до веб-додатку.

На початку існування сервісу Elastic Load Balancer не було вибору серед різних типів балансувальників, тільки у 2016 році був розширений цей перелік представивши: Application Load Balancer (ALB) та Network Load Balancer (NLB). Початкові функціональні можливості були збережені шляхом створення окремого третього типу Classic Load Balancer.

Application Load Balancer має декілька ключових понять, які необхідно розуміти для його налаштування. Перше – це правила (англ. rule), кожне правило задає певні умови (англ. conditions), цільову групу (англ. Target group) та пріоритет.

Прослуховувач (англ. listener) балансувальника, перевіряє запити на підключення від клієнтів, використовуючи попередньо заданий протокол та порт. Правила, що задаються слухачу, визначають, які запити та яким чином балансувальник направляє до цільових груп. В ALB можна створити до 10 правил на базі URL.

Кожна цільова група направляє запити до однієї чи декількох зареєстрованих цілей, таким як екземпляри EC2, використовуючи вказаний протокол та номер порту. Для цільової групи налаштовується функція перевірки працездатності (англ. health check), що виконує перевірку усіх вузлів зареєстрованих в групі.

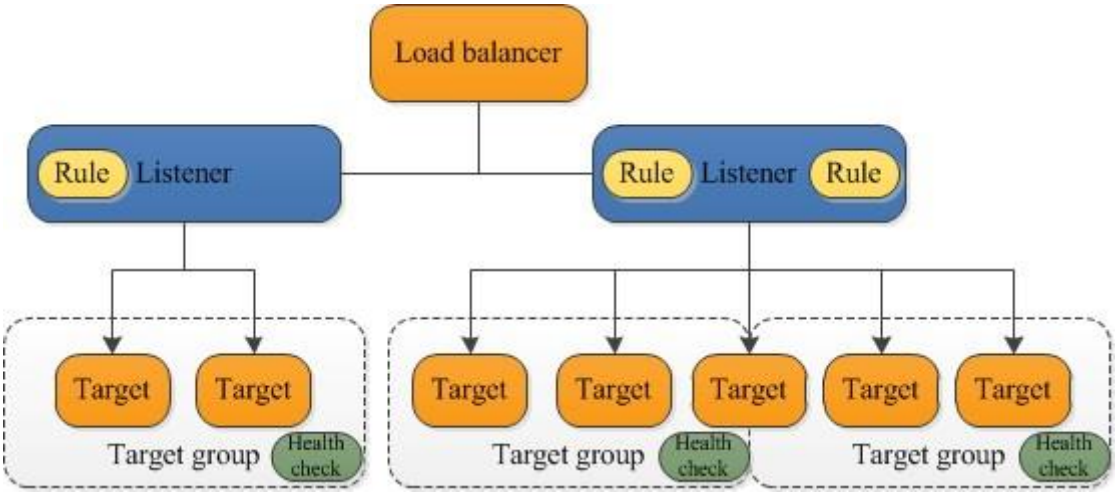


Рисунок 3.2 – Схема системи з використанням Application Load Balancer

На рис. 3.2 зображена схема, що ілюструє основні компоненти системи. Зверніть увагу, що кожний прослуховувач містить правило за замовчуванням,

а один слухач містить додаткове правило, яке направляє запити в іншу цільову групу вузлів [25].

Application Load Balancer функціонує на прикладному рівні L7 моделі OSI. Він перевіряє пакети та створює точки доступу до заголовків HTTP та HTTPS, розпізнаючи тип навантаження, розподіляє його по вузлам з більшою продуктивністю базуючись на аналізі трафіку веб-додатку, що передається в HTTP запитах. Після того, як балансувальник навантаження отримує запит, він оцінює правила слухача в пріоритетному порядку, щоб визначити яке з них використати, а потім обирає ціль з групи для виконання цього правила. Налаштовуючи правила можна задати критерії для маршрутизації до кожної цільової групи. Розподілення трафіку визначається заданим алгоритмом балансування, який за замовчуванням встановлений як Round-Robin – циклічний перебір. В якості альтернативи є можливість змінити алгоритм розподілення на least outstanding requests – найменш очікуючих запитів.

ALB порівняно з Classic Load Balancer, який працює на транспортному рівні L4, має більш розвинутий перелік функцій, завдяки можливості аналізу вхідних пакетів. Розглянемо деякі можливості, що надає ALB розробнику вебдодатків.

1. Підтримка додатків на основі контейнерів: все популярнішою стає концепція контейнеризації, що дозволяє упаковувати свої мікросервіси в контейнери та розміщувати їх в екземплярах EC2. Це дозволяє одному вузлу EC2 запускати декілька сервісів. ALB здатний підтримувати такі контейнерні додатки завдяки концепції цільових груп, що дозволяє розподіляти запити базуючись на номері порту. Наприклад, маючи два вузли EC2 на одному з яких є два контейнери з відповідними номерами порту, а інший має три контейнери, тоді аналізуючи вхідні запити балансувальник буде рівномірно розподіляти навантаження між всіма п’ятьма сервісами.

2. Маршрутизація базуючись на шляху: ALB підтримує маршрутизацію відповідно до URL на який клієнт робить запит. Тому можлива ситуація, коли декілька доменів направляють запити на один балансувальник навантаження, який може розподілити ці запити на різні вузли EC2.

3. Підтримка додаткових протоколів: ALB підтримує два додаткових протоколи – HTTP/2 та WebSocket. HTTP/2 – даний протокол підтримує мультиплексування запитів по одному з'єднанню. WebSocket – цей протокол дозволяє встановити довготривале TCP-з'єднання між клієнтом та сервером, є більш ефективним у порівнянні зі старими методами.

Для створення та налаштування балансувальника навантаження ALB потребується застосування як мінімум двох зон доступності, в яких будуть працювати вузли EC2. Кожна із зон має віртуальну приватну хмару VPC, з хоча б одною налаштованою підмережою, які використовуються для конфігурації балансувальника. Розглянемо перелік дій для створення балансувальника ALB.

1. Користуючись веб-інтерфейсом AWS Management Console відкриємо підрозділ Load Balancers сервісу Elastic Compute Cloud (EC2) та створимо новий Application Load Balancer.

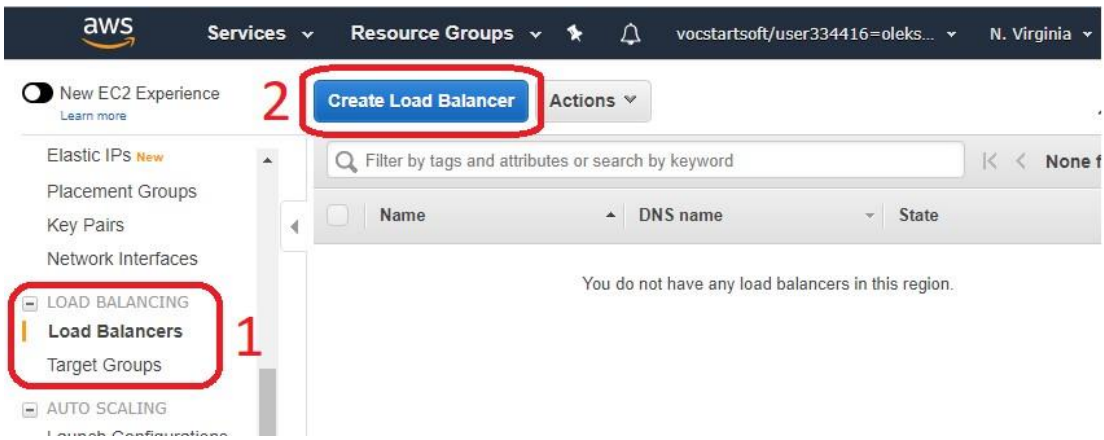


Рисунок 3.3 – Інтерфейс сервісу Elastic Compute Cloud

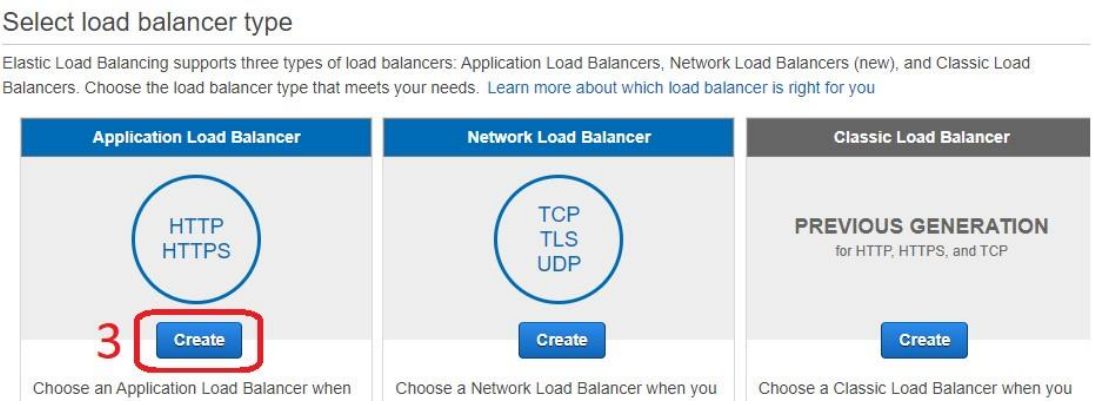


Рисунок 3.4 – Вибір типу балансувальника навантаження

2. На сторінці конфігурації балансувальника навантаження необхідно вказати базові налаштування включно з слухачами портів та зонами доступності.

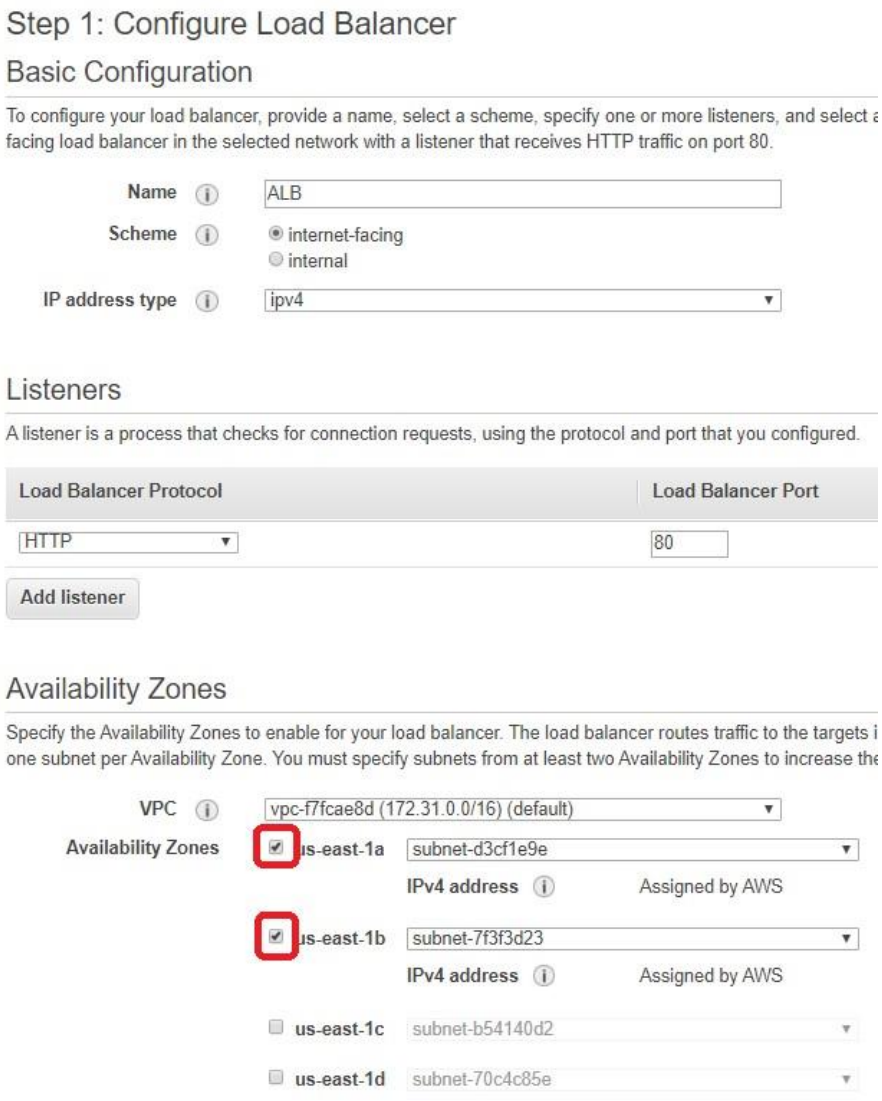


Рисунок 3.5 – Базова конфігурація та встановлення переліку зон доступності

Для того, щоб новий балансувальник навантаження був точкою дотику між системою та користувачем, необхідно встановити тип схеми – internetfacing, що зробить доступним підключення через Інтернет.

Наступний блок налаштування слухачів, що має можливість розподіляти трафік за типом протоколу та порту на який приходять запити.

Блок вибору зон доступності надає перелік доступних, серед яких можна обрати декілька, які будуть підключені до нового ALB.

3. Налаштування групи безпеки для нового балансувальника, яка повинна дозволяти йому обмінюватися даними з зареєстрованими вузлами, як на порту слухача, так і на порту перевірки працездатності.

Так, як обмін даними буде відбуватися через HTTP запити на базі TCP протоколу, то створюємо правило яке обмежить перелік доступних протоколів.

Рисунок 3.6 – Налаштування групи безпеки

4. Створення цільової групи, яка буде використовуватися при маршрутизації запитів. Попередньо створений у пункту 1 прослуховувач, який розрахований на HTTP запити буде переправляти їх саме в дану цільову групу.

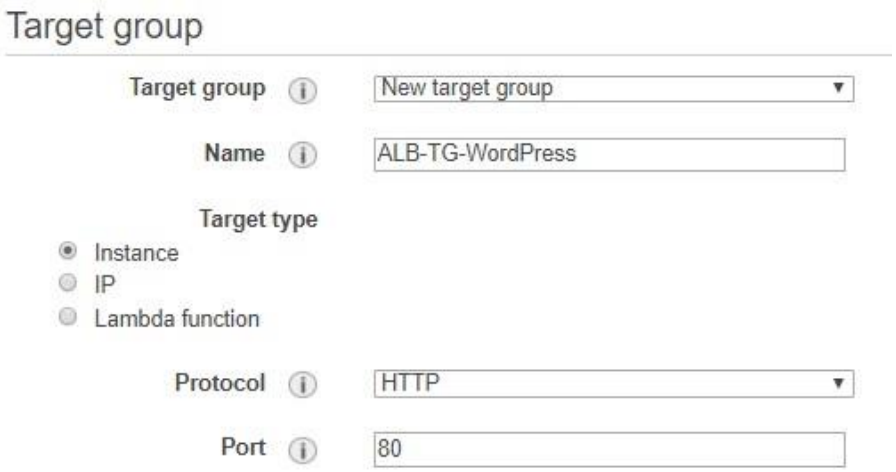


Рисунок 3.7 – Налаштування цільової групи

Балансувальник навантаження перевіряє працездатність вузлів у цільовій групі, базуючись на параметрах які задаються в наступному блоці.

Рисунок 3.8 – Налаштування перевірки працездатності вузлів

Спочатку задається посилання на ресурс, це може бути головна сторінка веб-додатку або службова. За даним посиланням балансувальник перевіряє можливість створення сеансу зв’язку з конкретним вузлом. В додаткових секціях задаються параметри, що описують кількість необхідних успішних перевірок та допустима кількість провальних, інтервали між перевірками, очікуваний код HTTP-відповіді.

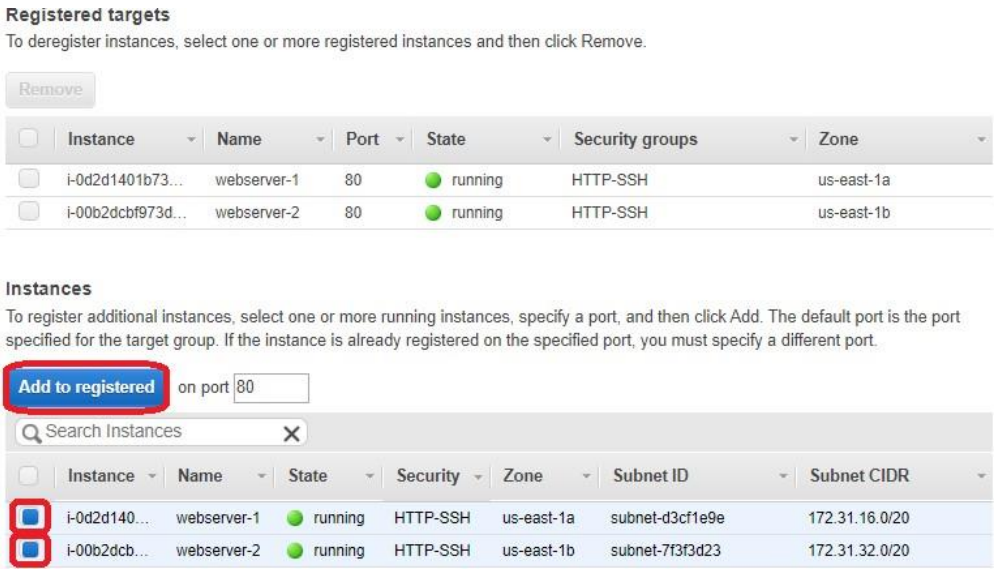


Рисунок 3.9 – Реєстрація вузлів у цільовій групі

Останнім кроком в створенні цільової групи є реєстрація вузлів, розглянувши рис. 3.9 можна побачити, що були створені два вузли EC2 – webserver-1 та webserver-2, які розміщені в двох зонах відповідно: us-east-1a та us-east-1b. Обираючи їх та натиснувши кнопку “Add to registered”, з’являється запис про реєстрацію вище в блоці “Registered Targets”.

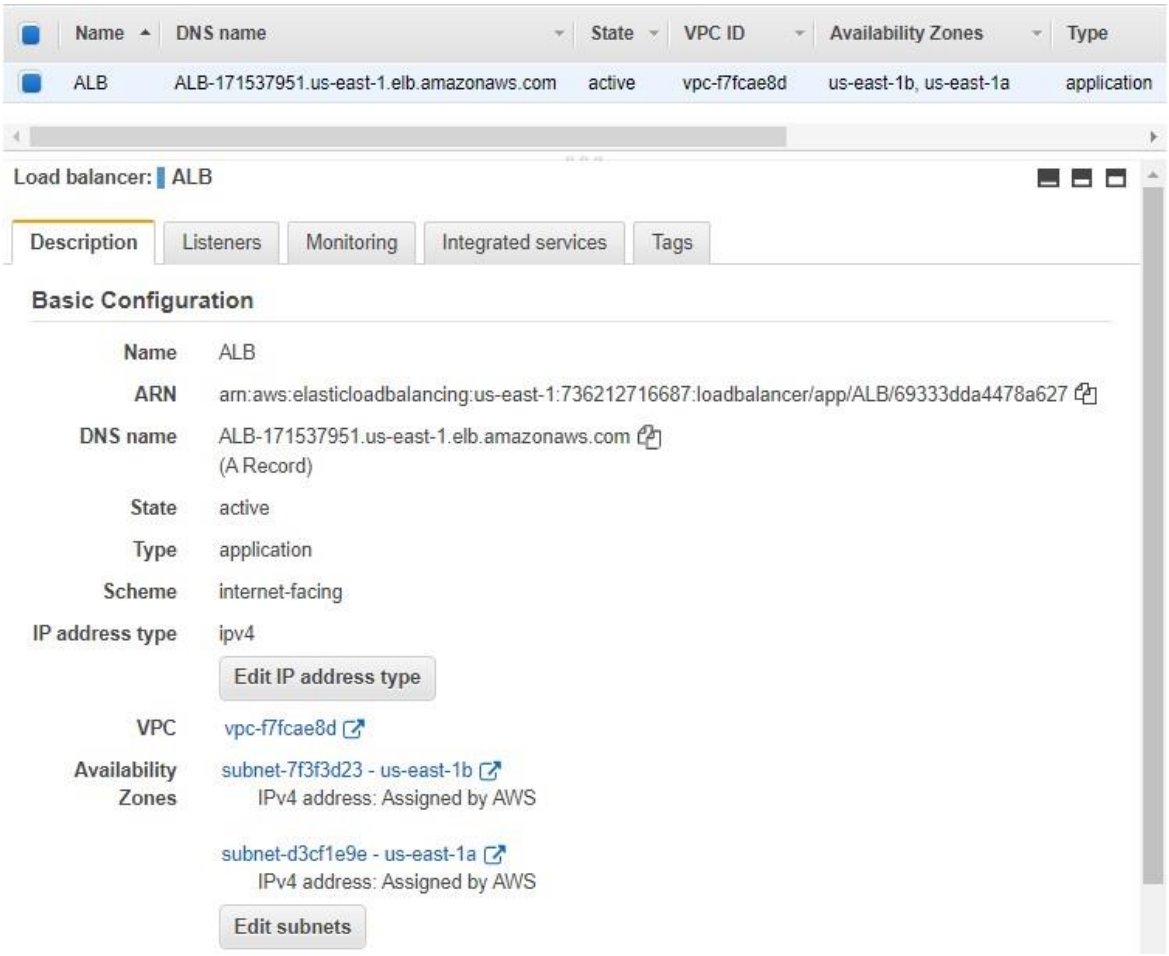


Рисунок 3.10 – Створений Application Load Balancer

Після завершення налаштування балансувальника навантаження йому надається DNS ім’я для доступу користувачів. Саме через це посилання користувачі звертаються до веб-додатку, розміщеному на вузлах EC2.

Останнє оновлення ALB у листопаді 2019 року надало можливість змінювати алгоритм розподілення запитів для цільових груп. Раніше єдиним доступним алгоритмом був Round-Robin (циклічний перебір), який є гарним вибором у випадку коли запити схожі та вузли ідентичні. Однак тепер доступний алгоритм Least Outstanding Requests (найменш невиконаних запитів), що обирає наступний вузол для пересилання запиту, базуючись на тому який з екземплярів має найменшу кількість очікуючих (не завершених) запитів. Тобто запити передаються вузлу, який менш завантажений.

Для того, щоб змінити алгоритм розподілення, необхідно обрати пункт “Edit attributes” в меню “Actions”.

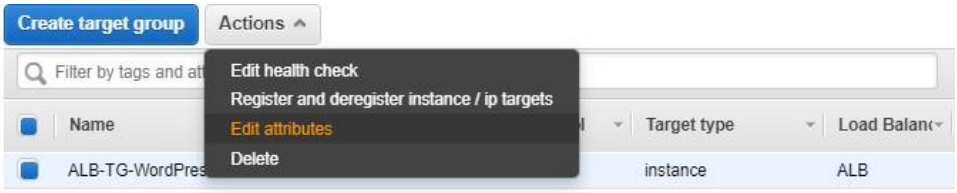


Рисунок 3.11 – Зміна алгоритму балансування (1)

Після чого змінити алгоритм на бажаний у секції “Load balancing algorithm”, рис. 3.12.

Рисунок 3.12 – Зміна алгоритму балансування (2)

При використанні Application Load Balancer оплата стягується погодинно, а також за використання одиниць ємності балансувальника (англ. Load Balancer Capacity – LCU). LCU вимірює кількість нових з’єднань в секунду, кількість активних з’єднань та передачу даних. Вимір йдеться по всім трьом параметрам, але оплата відбувається тільки на основі найбільшого. Одна одиниця LCU забезпечує: 25 підключень в секунду з сертифікатом 2KB, 3000 активних з’єднань, та 2.22 Mbps передачі даних, або 5 підключень в секунду з сертифікатом 4 KB, 3000 активних з’єднань, та 2.22 Mbps передачі даних.

3.2.2 Amazon Relational Database Service (RDS)

Сервіс розподілених реляційних баз даних, працюючи в хмарному середовищі призначений для спрощення налаштування, експлуатації та масштабування реляційної бази даних, що використовується в додатках. Процеси адміністрування, такі як виправлення програмного забезпечення бази даних, резервне копіювання та ін., відбувається автоматично. Дуже вагомою перевагою є можливість масштабування ресурсів на який розміщена база даних, так як процесор, пам’ять, сховище не пов’язані між собою, тому адміністратор системи може змінювати їх продуктивність чи ємність незалежно один від одного. Застосування поширених баз даних в сервісі RDS: MySQL, MariaDB, PostgreSQL, Oracle, Microsoft SQL Server, – майже не потребує додаткових вмінь. RDS надає можливість отримати переваги високої доступності (англ. high availability) з первинним екземпляром бази даних та синхронним другим екземпляром, який використовується у разі виникнення проблем. Для більшої надійності синхронний екземпляр може бути розміщений в іншій зоні доступності.

Для подальшого розгортання тестового веб-додатку, необхідно створити екземпляр RDS та приєднати його до системи.

1. Користуючись веб-інтерфейсом AWS Management Console відкриємо підрозділ “Databases” сервісу Relational Database Service (RDS) та створимо новий екземпляр RDS.

2. Необхідно обрати одну з представлених баз даних та її версію, для WordPress використовується MySQL.

3. Встановити ідентифікатор бази даних, в даному випадку – wordpress.

4. Обрати із запропонованих конфігурації віртуальної машини, на якій буде працювати база даних, - db.t2.micro (1 vCpu, 1GiB RAM), а також об’єм внутрішнього сховища на SSD носіях – 20GiB.

5. База даних поміщається у віртуальну приватну хмару (VPC) в одній з зон доступності.

6. Додатково встановлюються різноманітні налаштування виду: резервного копіювання, моніторингу, журналу записів, тощо.

Створивши екземпляр бази даних, відповідний запис з’явився в розділі “Databases” сервісу RDS.

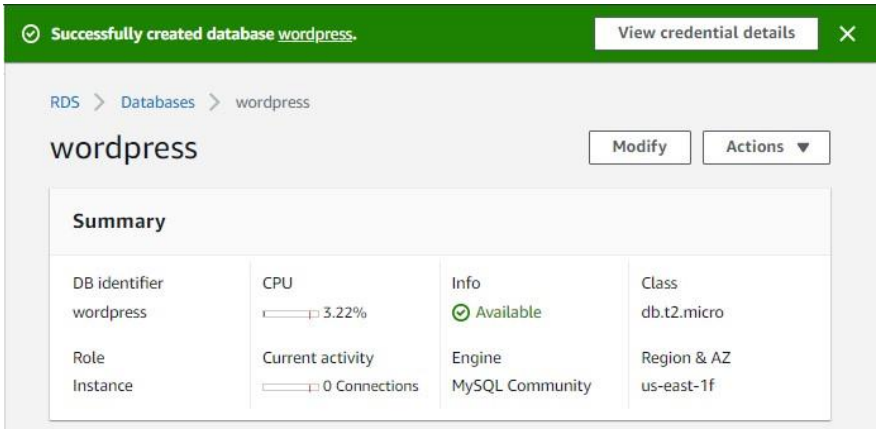


Рисунок 3.13 – Інформація про створену базу даних

3.2.3 AWS Elastic Beanstalk

Сучасні розробники веб-додатків все частіше при проектуванні системи обирають архітектури мікросервісів чи впроваджують інфраструктуру на основу контейнерів, можливість простого зіставлення одної адреси з конкретною службою стає все більш складнішою та важчою для обслуговування.

Application Load Balancer має доступ до заголовків HTTP, що дозволяє відповідним чином розподіляти запити до різних вузлів. Наприклад, запити що

містять /api у шляху URL можуть направлятися на одну цільову групу, а запити /mobile – направляються в іншу. Маршрутизація запитів таким чином дозволяє створювати веб-додатки, які складаються з декількох мікросервісів, що можуть працювати та масштабуватися незалежно один від одного.

AWS Elastic Beanstalk – сервіс оркестрування за моделлю PaaS, який надає можливість розробникам швидкого розгортання та керування вебдодатками. Розробники завантажують свої додатки в систему, делегуючи роботу по створенню інфраструктури та обробки конфігурації на AWS Beanstalk. Даному веб-додатку будуть надані ресурси, балансування навантаження, автоматичне масштабування та моніторинг працездатності. Оркестрування сервісів – це координація та організація декількох сервісів, представлених як один агрегований сервіс. Розробники використовують оркестрування для підтримки автоматизації бізнес-процесів шляхом зв'язку між різними додатками з подальшим створенням комплексних додатків. Іншими словами, оркестрування сервісів – комбінація сервісних взаємодій для створення сервісів більш високого рівня.

Beanstalk підтримує значний перелік можливих мов програмування, таких як: PHP, Python, Node.js, Java, .NET і т. ін. Важливо знати, що плата за даний сервіс не збирається окремо, необхідно лише сплачувати за ті ресурси, які використовуються для роботи веб-додатку. Тому вартість не є фіксованою, а тому може коливатись в залежності від кількості екземплярів EC2, розміру сховища даних та конфігурації бази даних.

Для розгортання веб-додатку за допомогою сервісу AWS Beanstalk спочатку необхідно натиснути кнопку "Create New Application" на сторінці сервісу, що відкриє форму для заповнення базової інформації про додаток: назва додатку, назва робочої середи, доменне ім'я та платформа розробки, в даному випадку – PHP. По завершенню всіх внутрішніх налаштувань, сервіс переходить на головну сторінку веб-додатку, яка містить перелік недавніх

подій додатку, меню налаштувань, моніторингу, перевірки працездатності, тощо.

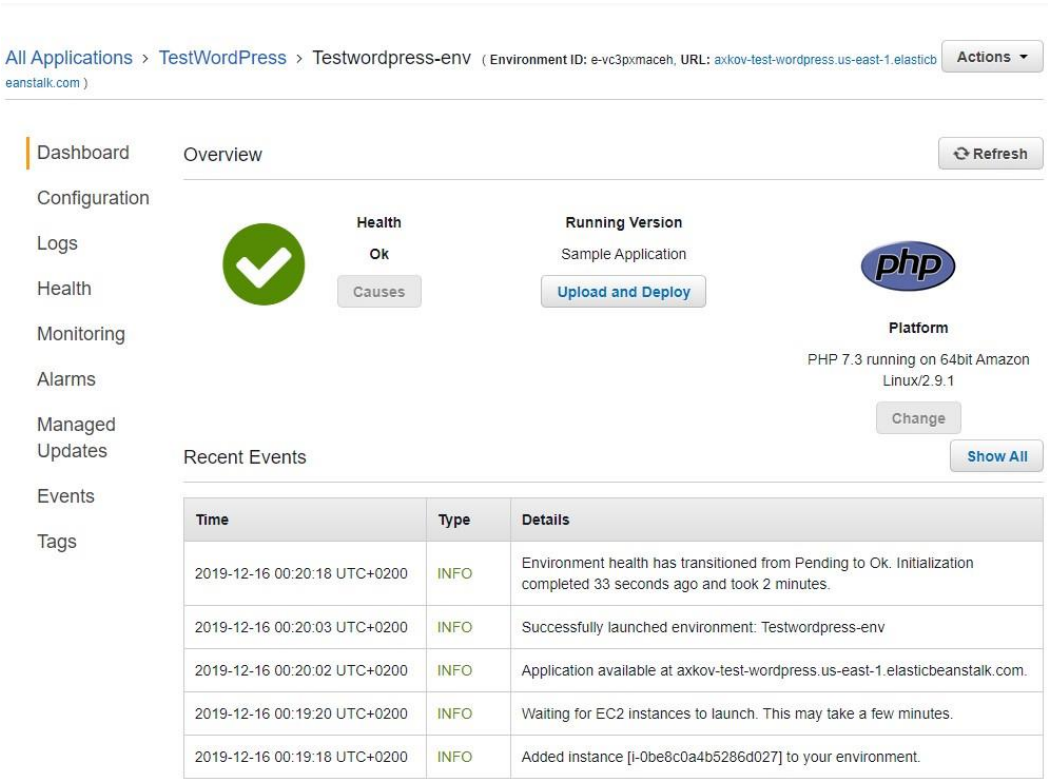


Рисунок 3.14 – Головне меню середи розробки створеного веб-додатку

Перейшовши до розділу “Configuration”, можна побачити повний перелік ресурсів, що використовується додатком та містить відомості.

1. Кількість та конфігурацію вузлів EC2.
2. Групу безпеки (англ. Security Group) до якої належать вузли.
3. Тип балансувальника навантаження.
4. Групу автоматичного масштабування.
5. Повідомлення Amazon CloudWatch, та ін.

Далі додаємо групу безпеки екземпляру бази даних створену раніше в робочу середу веб-додатку. Дана процедура спонукає Elastic Beanstalk повторно надати усі екземпляри робочої середи з приєднаною групою безпеки.

Для цього у розділі “Configuration” натиснути кнопку “Modify” навпроти відомостей про екземпляри та обрати зі списку групу безпеки, що містить базу даних – SG-wordpress, додатково до групи безпеки веб-додатку. При виконанні даної процедури усі створені вузли додатку будуть замінені на нові з актуальними налаштуваннями.

EC2 security groups

	Group name	Group ID	Name
☐	ALB-WP-SecurityGroup	sg-0e1065fad2117a75f	
☑	awseb-e-vc3pxmaceh-...	sg-082bff969b410d0fe	Testwordpress-env
☐	default	sg-0b7cbc5e	
☐	HTTP-SSH	sg-0b63b1504bfa552ad	
☐	launch-wizard-1	sg-0e970abe1ecb793f3	
☑	SG-wordpress	sg-08664704ae91de463	

Рисунок 3.15 – Обрані групи безпеки

Elastic Beanstalk автоматично змінює файли конфігурації веб-додатку, що дозволяє налаштовувати дані про бази даних. Для цього у розділі “Configuration” модифікуємо підрозділ “Software”.

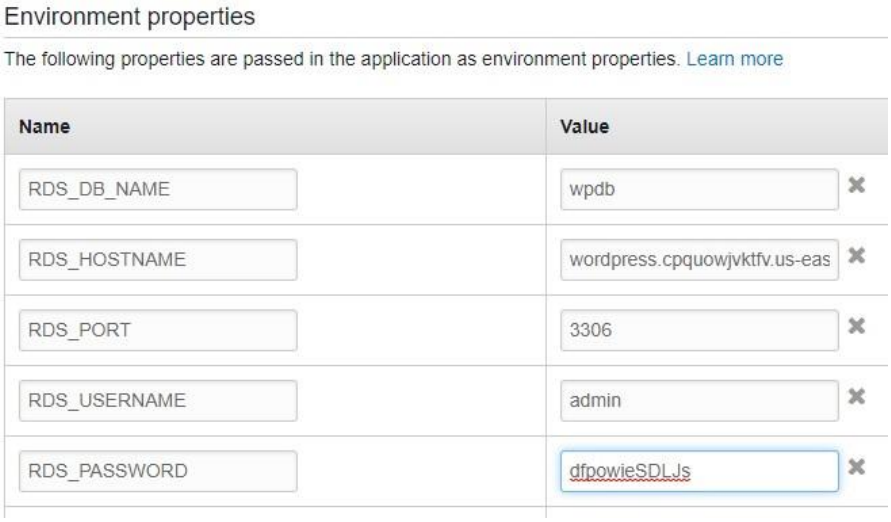


Рисунок 3.16 – Додавання відомостей про базу даних

Щоб завантажити веб-додаток у систему, треба створити Zip-архів, який містить вихідний код додатку. Включно з файлами налаштуваннями для взаємодії з сервісами AWS. Для цього на головній сторінці сервісу необхідно натиснути кнопку “Upload and Deploy” та додати створений архів.

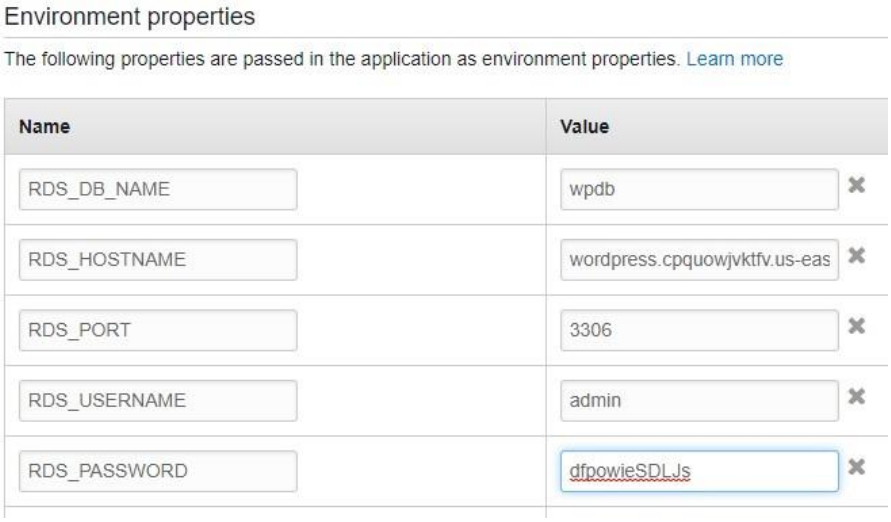


Рисунок 3.17 – Завантаження тестового веб-додатку у створену систему

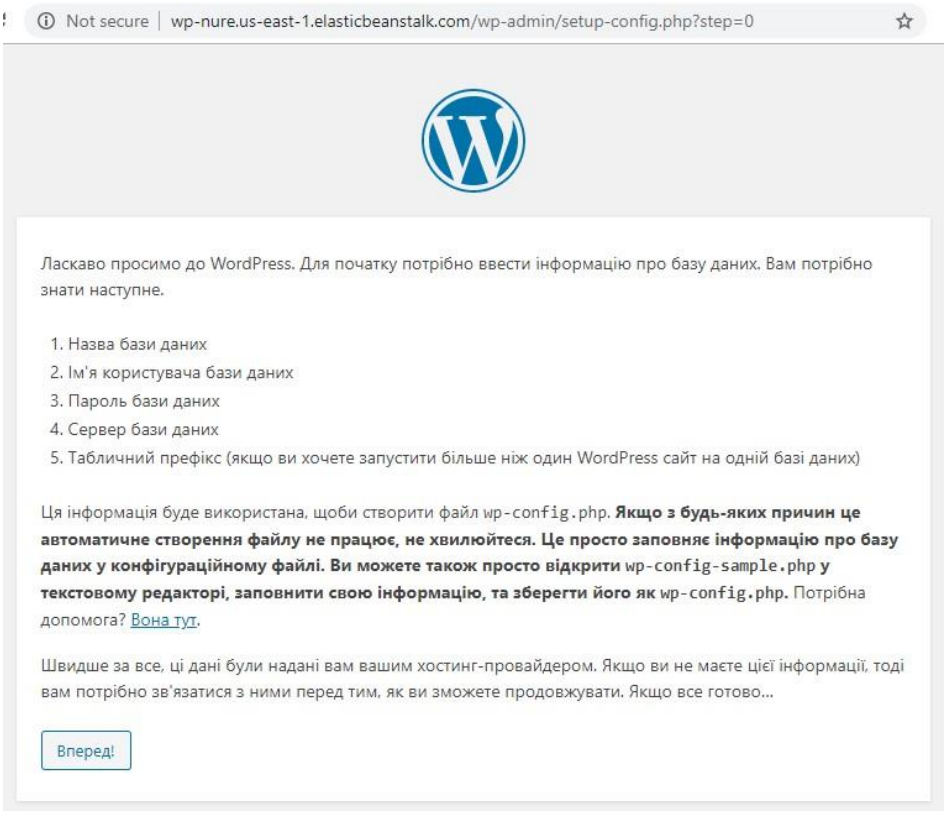


Рисунок 3.18 – Перший запуск додатку

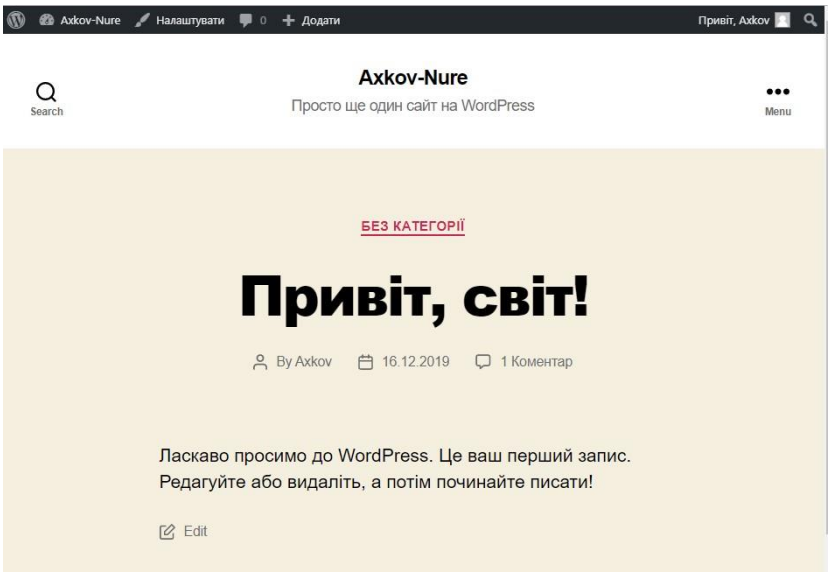


Рисунок 3.19 – Головна сторінка додатку

3.2.4 Auto Scaling Group

Для автоматичного масштабування вузлів при недостатній продуктивності веб-додатку необхідно створити групу автоматичного

масштабування (англ. auto scaling group), яка містить перелік екземплярів EC2, які розглядаються як логічна група для автоматичного масштабування та керування. Перш ніж конфігурувати нову групу масштабування, необхідно створити еталону копію екземпляру EC2 з працюючим веб-додатком, який буде шаблоном для новостворених вузлів. Для цього необхідно перейти до пункту “Instances” сервісу EC2 та обрати дію створення нового образу для еталонного вузла – “Create image”.

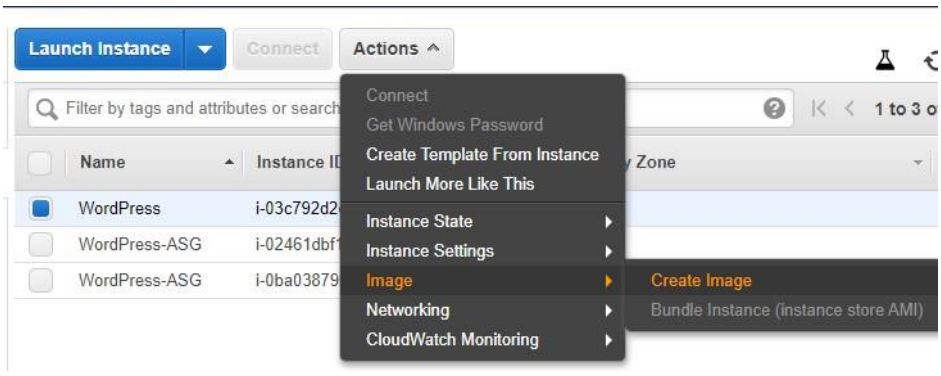


Рисунок 3.20 – Створення образу для масштабування

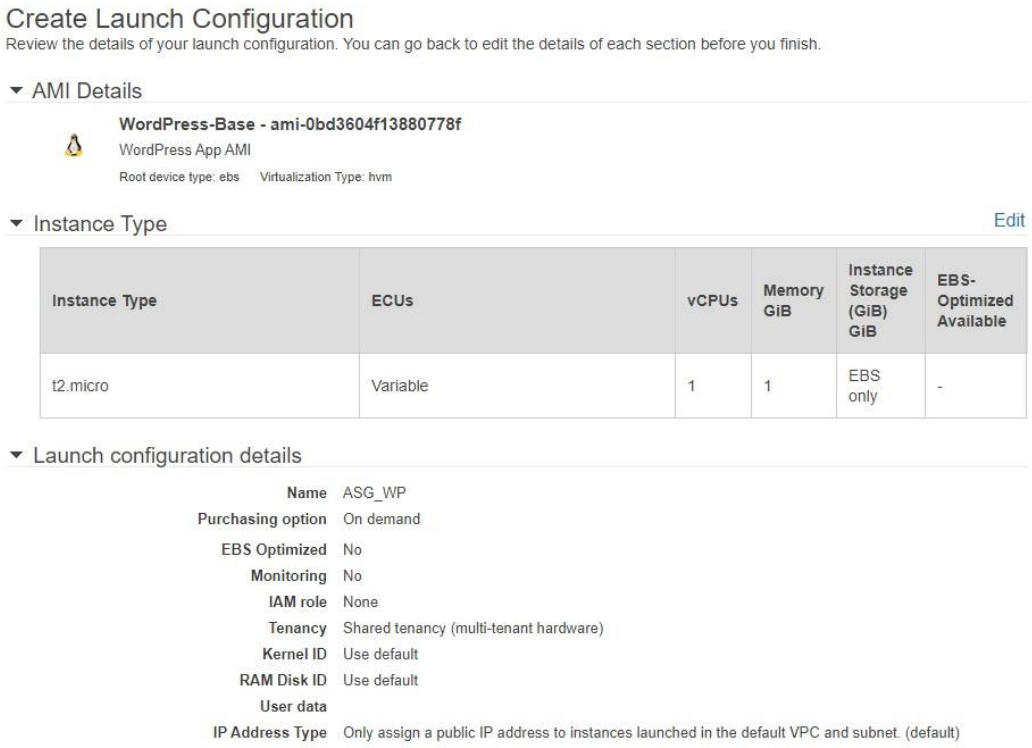


Рисунок 3.21 – Створення нової групи автоматичного масштабування

На рис. 3.21 можна побачити, що створюється нова група автоматичного масштабування з назвою ASG_WP. Обраний еталонний образ WordPress-Base, створений раніше, та тип екземплярів EC2, на які будуть встановлюватись образ, в даному випадку – t2.micro (1 vCPU, 1Gib RAM).

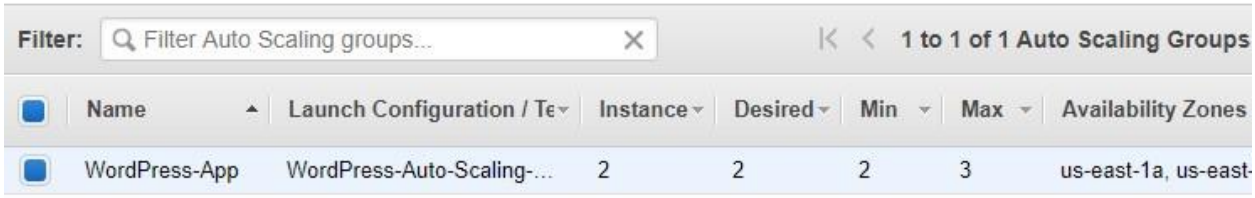


Рисунок 3.22 – Відомості про активну групу масштабування

Після створення групи автоматичного масштабування, можна задати політики за якими критеріями система буде вирішувати додавати новий вузол при надмірному навантажені, чи зупиняти в противному випадку. Такими критеріями може бути: відсоток завантаження центрального процесору, кількість запитів на вузол, кількість вхідних чи вихідних пакетів.

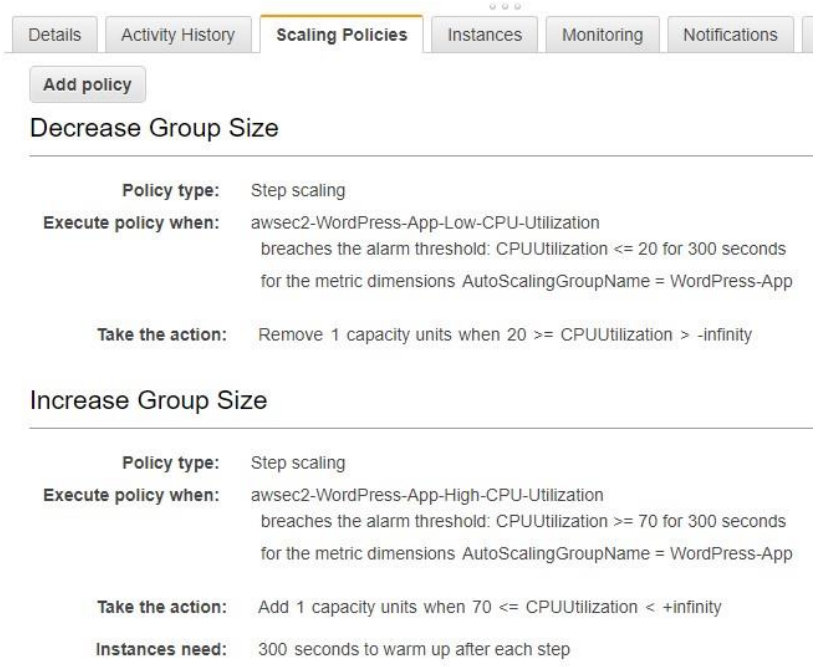


Рисунок 3.23 – Перелік політик масштабування

Рис. 3.23 зображує перелік створених політик масштабування при зростанні та зменшенні навантаження на ЦП (англ. CPU Utilization). При завантажені хоча б одного вузла більше 70% додається новий, а якщо менше 20%, то зупиняється один з активних. Додатково налаштовується автоматичне надсилання сповіщень розробникам додатку у разі масштабування системи.

3.2.5 Network Load Balancer

Додатково до Application Load Balancer, хмарна інфраструктура AWS підтримує балансуальник транспортного рівня L4 – Network Load Balancer (NLB). Трафік від клієнтів розподіляється по вузлам базуючись на IP-адресі та TCP-порту, що дає змогу обробляти мільйони запитів в секунду. Аналогічно до ALB, балансувальник транспортного рівня розподіляє навантаження між вузлами заданою цільової групи. Якщо розробник не потребує аналізу вхідних HTTP запитів, а просто приймає TCP/TLS пакети, як у випадку мережевих ігор чи відео-хостингу, тоді використовуючи NLB можна досягти більшої пропускної здатності каналу зв’язку. Недоліком даного балансувальника є обмеженість при перевірці працездатності вузлів, NLB має змогу перевіри доступність серверу в тому випадку, якщо той відповідає на ICMP ехо-запит. Application Load Balancer за рахунок можливості аналізу HTTP пакетів має змогу не просто перевірити успішність запиту, а також перевірити чи задовольняє вміст вихідних пакетів, параметрам надісланих користувачем.

Для TCP трафіку NLB розподіляє навантаження використовуючи алгоритм хешування потоку (англ. flow hash) на основі протоколу, IP-адреси джерела, порту джерела, IP-адреси призначення та порядкового номеру TCPпакета [27]. З’єднання TCP від клієнта мають різні вихідні порти та можуть бути направлені на різні вузли цільової групи. Кожне окреме TCP-з’єднання направляється на один сервер протягом всього сеансу.

На відміну від ALB, Network Load Balancer має можливість віддзеркалення трафіку (англ. Traffic mirroring), яка використовується для копіювання мережевого трафіку з балансувальника навантаження. Такий трафік може дублюватися на зовнішні пристрої безпеки чи моніторингу для: перевірки вмісту, моніторингу загроз, пошуку проблем. Дзеркальне відображення трафіку підтримує конфігуруємі фільтри та усічення пакетів, тому адміністратор може витягувати тільки необхідний йому трафік, використовуючи інструменті моніторингу за вибором. За рахунок таких можливостей, NLB є дієвим засобом боротьби з різного роду DDoS-атак [28].

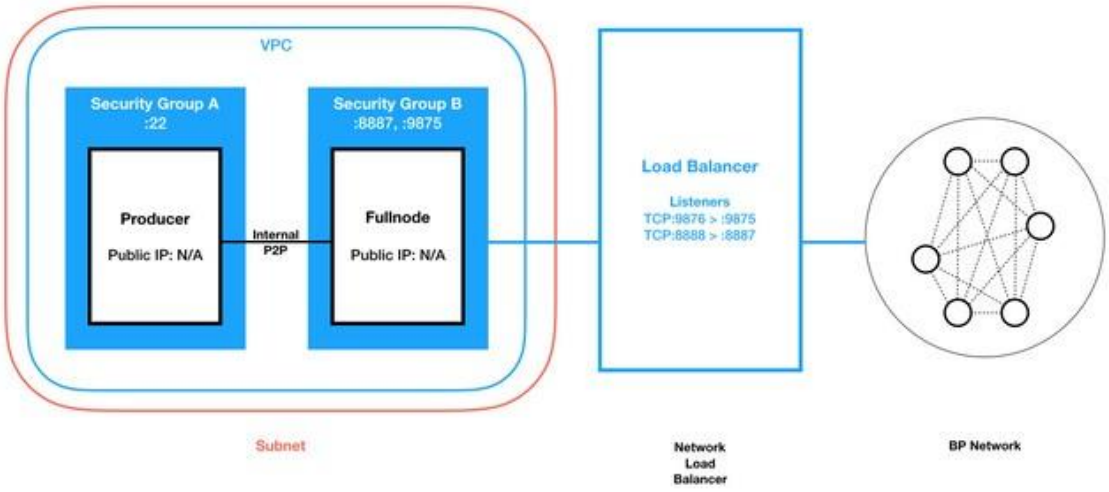


Рисунок 3.24 – Топологія системи з використанням NLB

4 ТЕСТУВАННЯ МЕТОДІВ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

Для подальшого аналізу продуктивності різних методів балансування навантаження, необхідно провести тестування систем, шляхом генерації трафіку на існуючий веб-додаток.

Тестування продуктивності – це практика тестування, яка виконується для визначення поведінки системи з точки зору відповідної реакції та стабільності при певному робочому навантажені. Додатково таке тестування може слугувати для дослідження, виміру, валідації та перевірки атрибутів

якості системи, таких як надійність, масштабованість та використання ресурсів [29]. Тестування продуктивності використовується для досягнення наступних цілей.

1. Оцінка готовності розробляємого програмного забезпечення до функціонування в робочій середі.
2. Перевірка відповідності критеріям ефективності.
3. Порівняння двох чи декількох платформ з однаковим ПЗ.
4. Визначення необхідної апаратної конфігурації.
5. Знайти вузькі місця системи.
6. Вияснити існуючі несправності та умови їх виникнення.

Базуючись на поставленій меті тестування та характеристик, які необхідно дослідити (час відгуку, пропускна здатність, використання ресурсів, максимальне навантаження), слід обирати необхідні види тестування продуктивності.

1. Навантажувальне тестування: основною метою якого є виявлення поведінки додатку при різних рівнях навантаження та при різних сценаріях використання. Досліджуваний параметр – час відгуку.
2. Стрес-тестування: основна мета – виявлення поведінки додатку при перевищенні стандартного навантаження. Головні параметри: час відгуку та пропускна здатність.
3. Тестування стабільності: перевірка відмовостійкості системи при нормальному навантажені в умовах довготривалої безперервної роботи. Головний досліджуваний параметр – використання оперативної пам’яті.
4. Тестування масштабування: основна мета даного тестування є визначення здатності системи до підвищення продуктивності за рахунок додавання обчислювальних вузлів та зміни архітектури, а також виявлення оптимальних системних налаштувань ПЗ.

Подальше тестування виконується для дослідження часу відгуку системи, перевірки коректності роботи автоматичного масштабування, рівня завантаженості вузлів та рівномірність розподілення запитів. В якості засобу навантажувального тестування був обраний хмарний сервіс BlazeMeter.

BlazeMeter – комерційна хмарна платформа для самостійного навантажувального тестування, що поширюється за моделлю PaaS. Сервіс пропонує інтегровану середу автоматизації тестування для всієї технічної групи протягом всього життєвого циклу розробки продукту. Можливе проведення неперервного тестування чи тестування за вимогою для API, мобільних додатків, веб-додатків і т. ін. Зручний веб-інтерфейс сервісу надає вичерпний перелік характеристик вимірювання, а також інтерактивні графіки.

4.1 Балансування з Application Load Balancer

В попередньому розділі була створена система для веб-додатку WordPress, що містить два вузли у режимі простою та балансувальник навантаження ALB? архітектура системи представлена на рис. 4.1. Так, як дана система передбачає збільшення кількості вузлів в разі надмірної завантаженості ЦП, тому слід перевірити коректність роботи групи автоматичного масштабування. Разом з цим перевіряється рівномірність балансування трафіку та час відгуку системи.

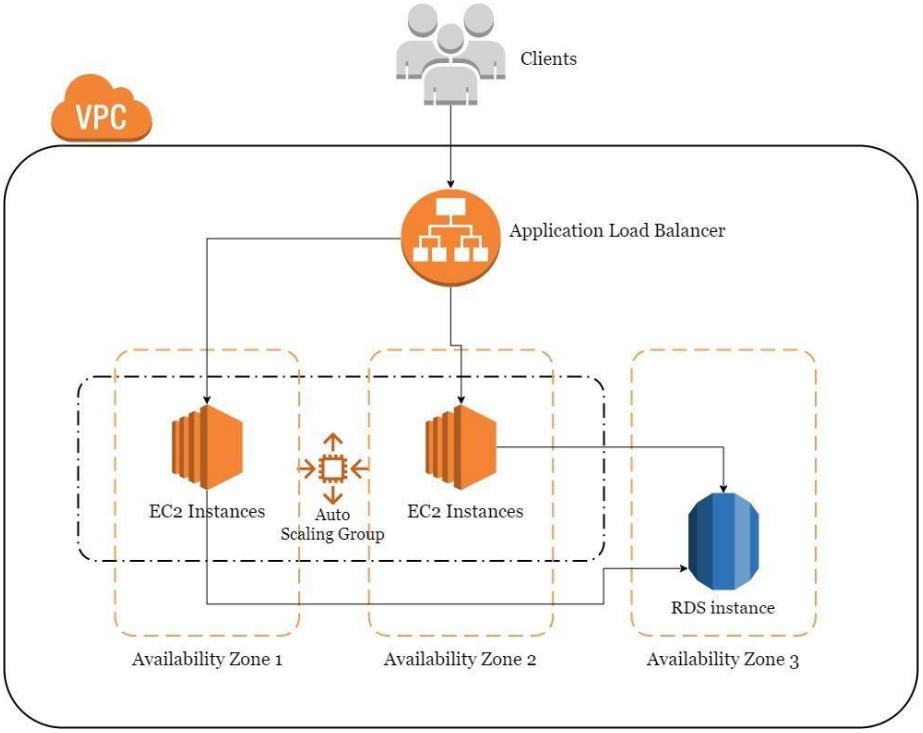


Рисунок 4.1 – Топологія системи з використанням ALB та RDS

Для того, щоб розпочати тестування в сервісі BlazeMeter, необхідно створити сценарії тестування, тобто перелік запитів які будуть виконуватися віртуальними користувачами. В даному випадку користувачі переходять на головну сторінку веб-додатку, потім на додаткові сторінки які містять текстові статті.

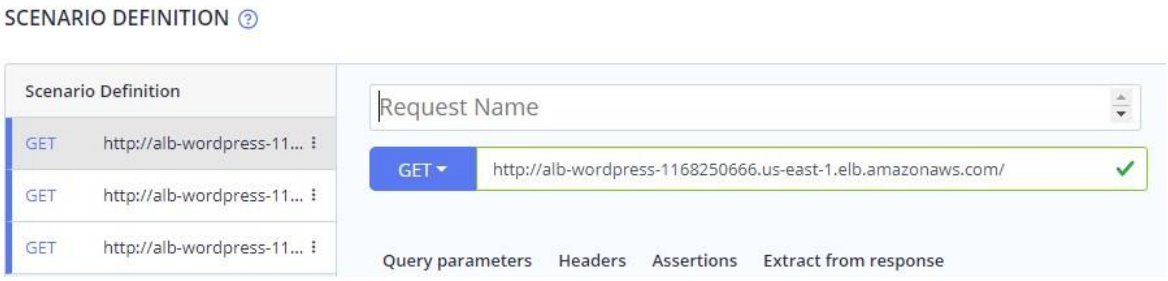


Рисунок 4.2 – Створення сценарію тестування

Обмеження пробної версії сервісу дозволяють виконати навантажувальне тестування із залученням 50 віртуальних користувачів протягом максимум 20 хвилин.

Виконавши тестування отримуємо такі результати сервісу BlazeMeter: кількість помилок – 0, середня пропускна здатність – 115.84 Запит./с, середній час відповіді – 421 мс, 90% час відповіді – 999 мс, середня ширина каналу зв'язку 6.25 МіБ/с. Слід пояснити значення 90% часу відповіді, який означає, що 90% від всіх запитів є меншими за це значення, тобто Більшість запитів за часом є меншими за 999 мс.

Рисунок 4.3 – Результати тестування (1)

Рисунок 4.4 – Результати тестування (2)

За графіком рис. 4.3 можна бачити, що з ростом кількості користувачів, зростає час відповіді, однак приблизно о 20:06 відбувається зниження середнього часу відповіді та підвищення кількості оброблених запитів. Така зміна обумовлена завантаженням ЦП вузлів, та активацією нового вузла, який розгрузив систему та забезпечив більшу загальну продуктивність. Завдяки тому, що система побудована на інфраструктурі AWS, є можливість застосувати сервіс CloudWatch, який збирає різноманітну інформацію з усіх сервісів AWS та відображає її у вигляді графіків. Отже можна пересвідчитися, що навантаження вузлів у цей час перевищило відмітку 70%, яка встановлена у налаштуваннях групи автоматичного масштабування, рис. 4.5.

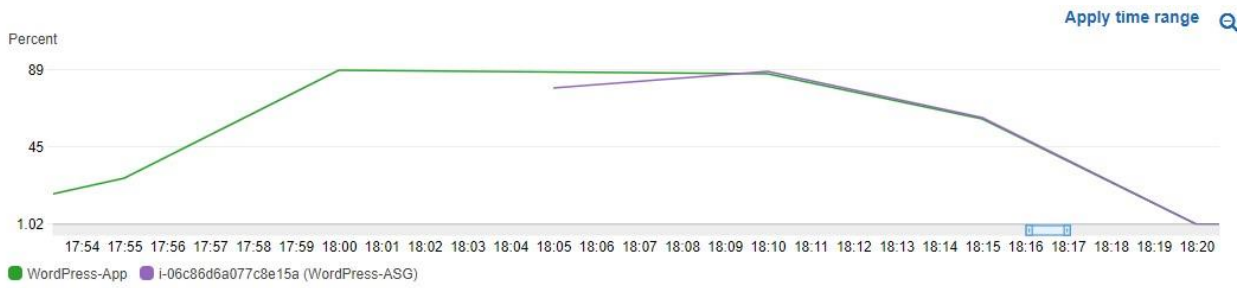


Рисунок 4.5 – Загальна завантаженість вузлів

З графіку видно, що загальне навантаження всіх ЦП групи масштабування WordPress-App перевищило поріг масштабування, тому згідно з графіку в 18:05 запускається новий вузол, який зменшує відсоток завантаження до відмітки 61% у момент 18:15. Слід пояснити розбіжності у часі в дві години, це обумовлено різними часовими поясами за якими працюють сервісі CloudWatch та BlazeMeter.

Рівномірність розподілу навантаження балансувальником можна перевірити за навантаженням кожного з вузла групи.

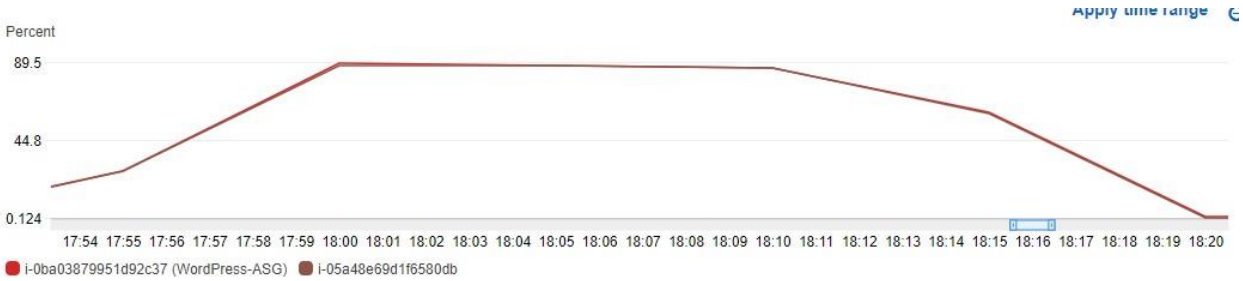


Рисунок 4.6 – Завантаженість вузлів групи автоматичного масштабування

Виходячи з графіку, зображеному на рис. 4.6, дві лінії, що характеризують завантаженість ЦП двох вузлів збігаються, тому робимо висновок, що Application Load Balancer виконує свою роботу задовільно.

Після завершення тестування та зменшенню навантаження вузлів, один з трьох вимикається та видаляється згодом, що свідчить про роботу політики завантаження ЦП менше 20%.

Рисунок 4.7 – Графік загальної кількості вузлів групи автоматичного масштабування

Наступне тестування перевіяє вплив на результати зміну алгоритму балансування з Round-Robin (циклічний перебір) на least outstanding requests (найменш невиконані запити). Тестовий сценарій залишається той самий, а отримані результати зображені на рис. 4.8-4.9.

Рисунок 4.8 – Результати тестування алгоритму least outstanding requests (1)

Відповідно вимірам значних змін не сталося, за винятком зменшення 90% часу відповіді – 563 мс. Даний показник свідчить, що кількість запитів які значно перевищують середній час відповіді знизилась. Це свідчить про уникнення аномальних зростань часу відповіді запиту.

Рисунок 4.9 – Результати тестування алгоритму least outstanding requests (2)

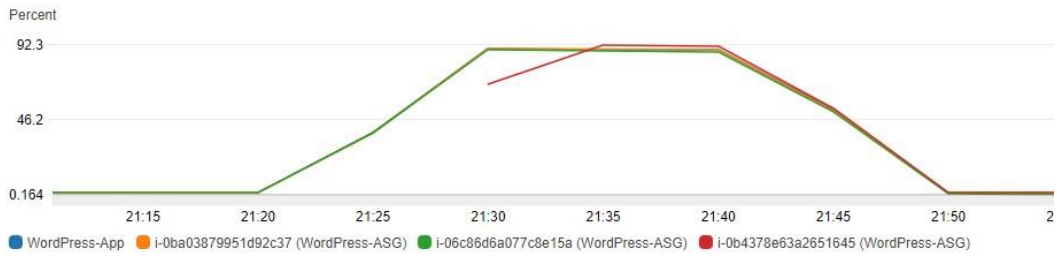


Рисунок 4.10 – Загальна завантаженість вузлів

Натомість, загальна завантаженість вузлів групи автоматичного масштабування не змінилась рис. 4.10. Можна припустити, що даний алгоритм більш ефективний у разі повного завантаження одного з вузлів, тому проведемо додаткове тестування з тими ж параметрами, але обмежимо групу масштабування до двох вузлів, що значить при надмірній завантаженості додаткових – виділятися не буде.

Рисунок 4.11 – Результати тестування алгоритму least outstanding requests без автоматичного масштабування (1)

Закономірно до зменшення загальної продуктивності системи до двох активних вузлів замість трьох, зменшилися середня пропускна здатність – 90.75 Запит./с, середня ширина каналу – 4.89 МіБ/с, та середній час відповіді – 537.65 мс.

Рисунок 4.12 – Результати тестування алгоритму least outstanding requests без автоматичного масштабування (2)

З графіку на рис. 4.12 видно деяке збільшення часу запиту у момент 01:00, але згідно рис. 4.13 можна сказати, що це похибка.

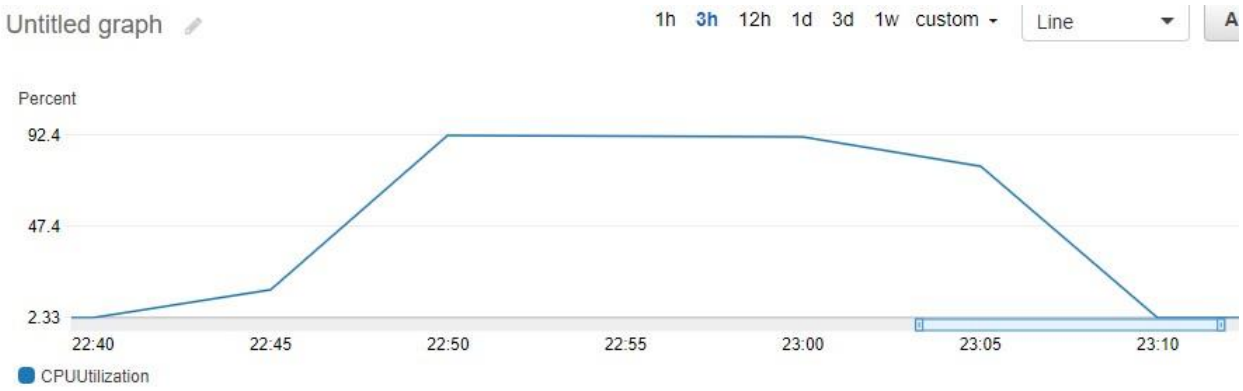


Рисунок 4.13 – Загальна завантаженість вузлів

Під час даного сеансу тестування загальна завантаженість вузлів не досягла відмітки 100%, що не дає в повній мірі оцінити можливості алгоритму least outstanding requests. Для підвищення рівня навантаження необхідно

збільшити кількість віртуальних користувачів, що неможливо у випадку користування базовою версією сервісу BlazeMeter.

4.2 Балансування з Network Load Balancer

Для того, щоб замінити Application Load Balancer на Network Load Balancer необхідно додати нову цільову групу, що поєднана з NLB до групи автоматичного масштабування. Тепер використовуючи DNS ім’я балансувальника NLB можна взаємодіяти з веб-додатком.

Проведемо аналогічне тестування за 50 віртуальними користувачами на 20 хвилин, результати представлені на рис. 4.14-4.15.

Рисунок 4.14 – Результати тестування Network Load Balancer (1)

Розглянувши результати тестування, можна помітити, що приросту пропускної здатності, як очікується для балансувальника навантаження транспортного рівня, не відбулося.

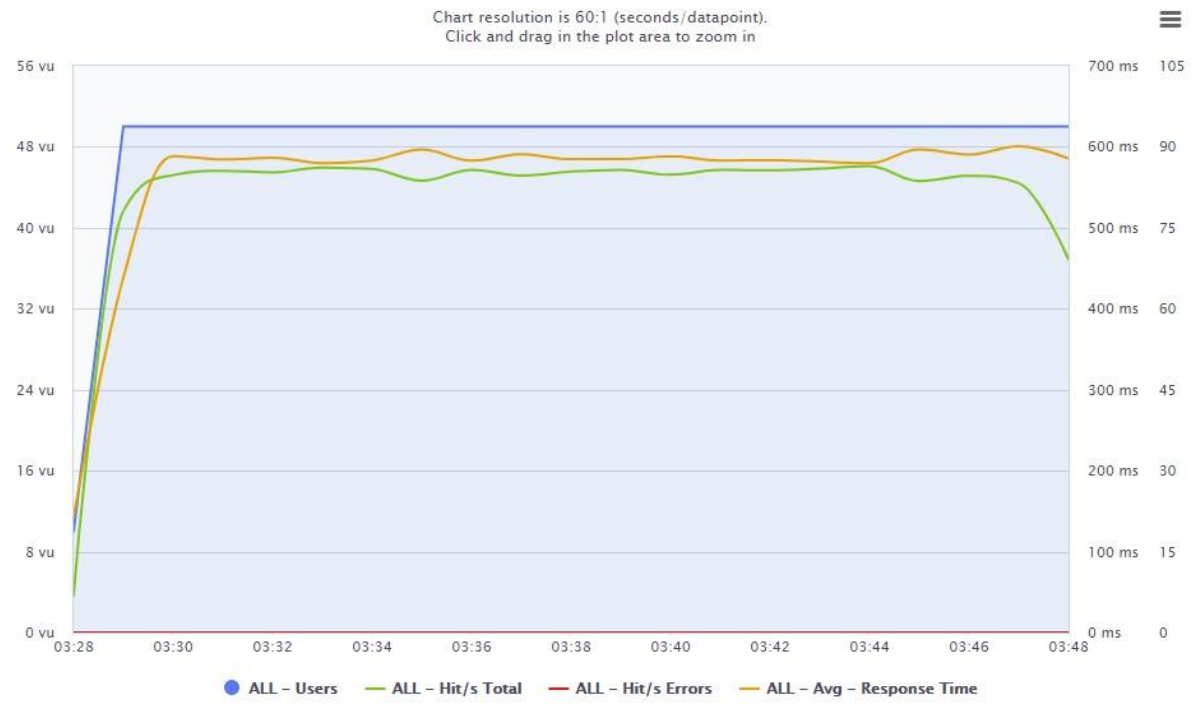


Рисунок 4.15 – Результати тестування Network Load Balancer (2)

Виходячи з графіку можна зробити припущення, що автоматичне масштабування вузлів не відбулося, тому загальна продуктивність не зросла, а залишилась приблизно на рівні застосування двох вузлів – 84 Запит./с. Щоб підтвердити дане припущення необхідно розглянути показання сервісу CloudWatch.

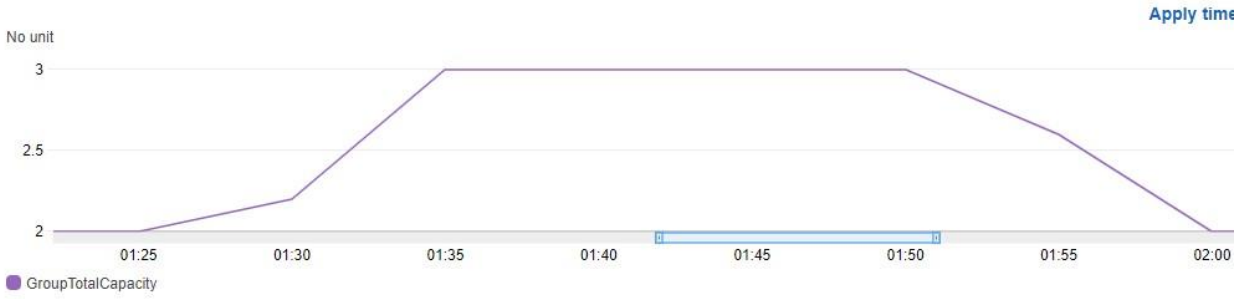


Рисунок 4.16 – Графік кількості активних вузлів

Графік кількості активних вузлів, рис. 4.16, демонструє, що політики групи автоматичного масштабування спрацювали, та масштабували систему

до трьох активних вузлів. Тому далі необхідно перевірити завантаженість цих вузлів.

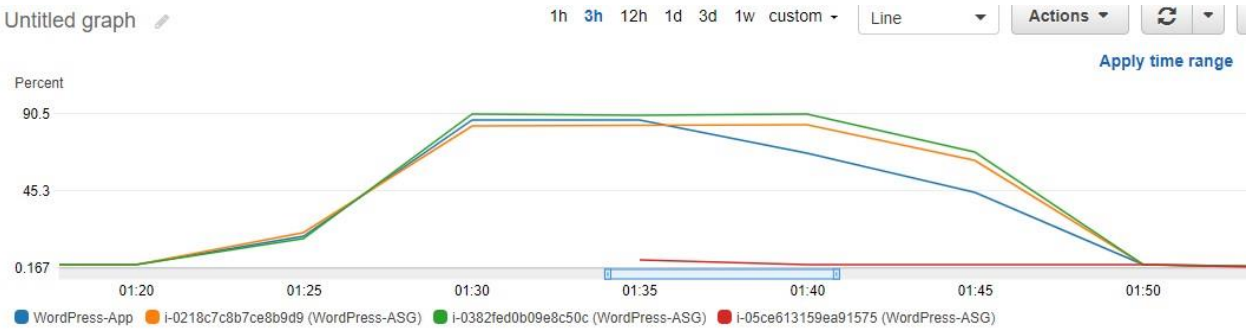


Рисунок 4.17 – Графік завантаженості вузлів групи автоматичного масштабування

В результаті аналізу графіка завантаженості вузлів на рис. 4.17, робимо висновок, що хоча система була масштабована правильно, балансувальник не виконав своєї функції та не розподілив запити на новий вузол, тому той простоював весь час тестування з моменту свого запуску.

Оскільки, NLB – балансувальник навантаження транспортного рівня L4 та працює на основі розподілу TCP-з’єднань, можна зробити висновок, що під час тестування створювалися безперервні TCP-сесії, які просто не могли бути перенаправлені на новостворений вузол. Напевно Network Load Balancer буде більш доцільніший для веб-додатків, що використовують специфічні протоколи на базі TCP, тоді як для передачі HTTP-запитів доцільніше використовувати Application Load Balancer.

ВИСНОВКИ

Стрімкий розвиток інформаційних технологій потребує від розробників програмного забезпечення створювати продуктивні та зручні продукти, адже висока конкуренція спонукає боротися за кожного користувача. Погана продуктивність веб-додатків може слугувати причиною відмови клієнтів від користування даною розробкою. Тому для забезпечення високої продуктивності та надійності при великій кількості запитів веб-додатку, виникає потреба у масштабуванні системних ресурсів. Вертикальне масштабування пов’язане з вдосконаленням існуючих апаратних засобів шляхом заміни комплектуючих на більш продуктивні, але таке покращення не завжди задовольняє потреби розробників, адже з часом стає недостатнім. Горизонтальне масштабування полягає у збільшенні кількості обчислювальних ресурсів (вузлів) аналогічними, що спонукає до використання додаткового компонента – балансувальника навантаження. Розподілення запитів між існуючими вузлами, що обслуговують веб-додаток, відбувається за допомогою алгоритмів балансування, таких як: Round-Robin, Least Connection, Hash-IP та ін. Окрім розподілення запитів, балансувальник навантаження забезпечує виконання додаткових функцій, наприклад, перевірка працездатності, підтримка сеансу з’єднання та забезпечення високої доступності вузлів шляхом надлишкових ресурсів.

Хмарні інфраструктури дають змогу розробниками використовувати обчислювальні засоби, та за потреби модифікувати чи масштабувати, без необхідності придбання власних апаратних засобів, а сплачуючи фінанси за погодинне використання хмарним провайдерам, які надають обчислення у виді сервісу. Існує три найбільш поширені моделі хмарних обчислень: програмне забезпечення як сервіс (SaaS), платформа як сервіс (PaaS), інфраструктура як сервіс (IaaS), кожна з моделей делегує відповідальність налаштування системи у певній мірі, між користувачем та постачальником.

Amazon Web Services – один з найпопулярніших постачальників хмарних інфраструктур, надає користувачам широкий перелік спеціалізованих сервісів, що дають можливість створення різноманітних систем та автоматизації налаштування. Сервіс Elastic Load Balancer пропонує три варіації балансувальників навантаження: Application Load Balancer, що працює на прикладному рівні L7, Network Load Balancer транспортного рівня L4 та Classic Load Balancer, який забезпечує зворотну сумісність з раніше створеними системами.

В даній роботі розглянутий приклад розгортання веб-додатку на базі хмарної інфраструктури Amazon Web Services. За основу веб-додатку була обрана WordPress вільна система керування контентом (CMS), що виконана за допомогою технологій PHP та MySQL. Створена система з використанням зовнішньої реляційної бази даних сервісу RDS, що дає можливість відділити обчислювальні вузли від зберігання стану системи для можливого подальшого масштабування. Група автоматичного масштабування забезпечує збільшення чи зменшення кількості вузлів у разі виконання встановлених політик.

Заключний розділ – “Тестування продуктивності методів балансування навантаження” описує проведене тестування продуктивності, що показало працездатність системи, коректну роботу балансувальника навантаження та автоматичного масштабування. Були порівняні два методи балансування шляхом тестування Application Load Balancer – прикладного рівня L7 та Network Load Balancer – транспортного рівня L4. На основі отриманих результатів ALB виявися найбільш доцільним варіантом розподілення HTTPзапитів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Digital 2019: global internet use accelerates [Електроний ресурс] – Режим доступу: https://wearesocial.com/blog/2019/01/digital-2019-globalinternet-use-accelerates

2. What is Cloud Computing? [Електроний ресурс] – Режим доступу: https://aws.amazon.com/what-is-cloud-computing/

3. Load balancing (computing) [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Load_balancing_%28computing%29

4. The NIST Definition of Cloud Computing [Електроний ресурс] – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-145/final#pubs-abstractheader

5. Cloud Computing in Business [Електроний ресурс] – Режим доступу: https://medium.com/@outrightsystems/cloud-computing-in-businessab19f308221d

6. Cloud computing [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Cloud_computing

7. Different Types of Cloud Computing Service Models [Електроний ресурс] – Режим доступу: https://www.bluepiit.com/blog/different-types-ofcloud-computing-service-models/

8. Scalability [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Scalability

9. Основы масштабирования [Електроний ресурс] – Режим доступу: https://habr.com/ru/post/15362/

10. Amazon Web Services [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Amazon_Web_Services

11. Microsoft Azure [Електроний ресурс] – Режим доступу:
https://en.wikipedia.org/wiki/Microsoft_Azure

12. Google Cloud Platform [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Google_Cloud_Platform

13. AWS vs Azure vs Google Cloud Market Share 2019: What the Latest Data
Shows [Електроний ресурс] – Режим доступу: https://www.parkmycloud.com/blog/aws-vs-azure-vs-google-cloud-market-share/

14. Public Cloud Services Comparison [Електроний ресурс] – Режим доступу: http://comparecloud.in/

15. SYN flood [Електроний ресурс] – Режим доступу:
https://en.wikipedia.org/wiki/SYN_flood

16. Load Balancing 101: Nuts and Bolts [Електроний ресурс] – Режим доступу: https://f5.com/resources/white-papers/load-balancing-101-nuts-andbolts-31392

17. TLS termination proxy [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/TLS_termination_proxy

18. Introduction to modern network load balancing and proxying [Електроний ресурс] – Режим доступу: https://blog.envoyproxy.io/introductionto-modern-network-load-balancing-and-proxying-a57f6ff80236

19. S. Sankara Narayanan, M. Ramakrishnan A Comprehensive Study on Load Balancing Algorithms in Cloud Computing Environments [Текст] / S. Sankara Narayanan, M. Ramakrishnan // Research Journal of Applied Sciences, Engineering and Technology – 2016. – Т. 13, №10. – С. 794–799.

20. Rajwinder Kaur, Pawan Luthra Load Balancing in Cloud Computing [Текст] / Rajwinder Kaur, Pawan Luthra // Recent Trends in Information, Telecommunication and Computing, ITC – 2014, №5. – С. 374–379.

21. ZEN Load balancer часть №1. Теория и практика балансировки нагрузки [Електроний ресурс] – Режим доступу: https://ivirt-it.ru/zen-loadbalancer-theory/

22. Comparing Load Balancing Algorithms [Електроний ресурс] – Режим доступу: https://www.jscape.com/blog/load-balancing-algorithms

23. Zenon Chaczko, Venkatesh Mahadevan Availability and Load Balancing in Cloud Computing [Текст] / Zenon Chaczko, Venkatesh Mahadevan, Shahrzad Aslanzadeh, Christopher Mcdermid // International Conference on Computer and Software Modeling, IPCSIT – 2011, №14. – С. 134–140.

24. Michael Wittig, Andreas Wittig Amazon Web Services in Action. [Текст] / 1 вид.: Manning Publication, 2015. – 424 p.

25. Application Load Balancer [Електроний ресурс] – Режим доступу: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html

26. Classic Load Balancer [Електроний ресурс] – Режим доступу: https://docs.aws.amazon.com/elasticloadbalancing/latest/classic/introduction.html

27. Network Load Balancer [Електроний ресурс] – Режим доступу: https://docs.aws.amazon.com/elasticloadbalancing/latest/network/introduction.html

28. Prevent DDoS with AWS Network Load Balancer [Електроний ресурс] – Режим доступу: https://steemit.com/eos/@eosnodeone/prevent-ddos-with-awshttps://steemit.com/eos/@eosnodeone/prevent-ddos-with-aws-network-load-balancernetwork-load-balancer

29. Software performance testing [Електроний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Software_performance_testing

30. Шайхутдинова А. Ф. Тестирование производительности вебприложений: основные приемы генерации нагрузки и мониторинга // European science. 2015. №6 (7). URL: https://cyberleninka.ru/article/n/testirovaniehttps://cyberleninka.ru/article/n/testirovanie-proizvoditelnosti-veb-prilozheniy-osnovnye-priemy-generatsii-nagruzki-i-monitoringaproizvoditelnosti-veb-prilozheniy-osnovnye-priemy-generatsii-nagruzki-ihttps://cyberleninka.ru/article/n/testirovanie-proizvoditelnosti-veb-prilozheniy-osnovnye-priemy-generatsii-nagruzki-i-monitoringamonitoringa.

31. Test the Performance of Web Applications Under Load [Електроний ресурс] – Режим доступу: https://www.loadtestingtool.com/

32. BlazeMeter [Електроний ресурс] – Режим доступу: https://www.blazemeter.com/

Висновки

В даній роботі побудовано систему балансування навантаження на базі хмарної інфраструктури Amazon Web Services, розглянутий приклад розгортання веб-додатку, за основу якого була обрана WordPress система керування контентом.

Проведено тестування продуктивності, що показало працездатність системи, коректну роботу балансувальника навантаження та автоматичного масштабування. Були порівняні два методи балансування шляхом тестування Application Load Balancer – прикладного рівня L7 та Network Load Balancer – транспортного рівня L4. На основі отриманих результатів ALB виявися найбільш доцільним варіантом розподілення HTTP-запитів.

ПОРІВНЯННЯ МЕТОДІВ БАЛАНСУВАННЯ НАВАНТАЖЕННЯ

Ковріжний О.В.
Науковий керівник – д.т.н., проф. Кривуля Г.Ф.
Харківський національний університет радіоелектроніки
(61166, Харків, пр. Науки,14, каф. АПОТ, тел. (057) 702-13-26) e-mail: oleksii.kovrizhnyi@nure.ua

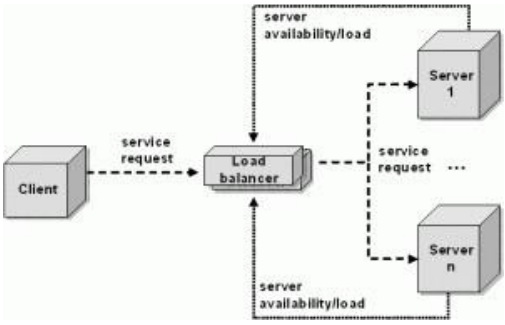
This paper focuses on the problem of load balancing network traffic. Review of existing methods of load balancing, identifying their strength and weaknesses.

Вступ. З проблемою балансування навантаження зустрічається будьякий веб-проект, бо відмова роботи серверу через велику кількість вхідних запитів може відбутися несподівано та призвести до небажаних матеріальних втрат. Стійкість серверу до навантаження можна вирішувати різними шляхами: будь-то зміна апаратної складової чи оптимізацією використовуємих алгоритмів та програмних кодів. Великі веб-проекти вирішують цю проблему шляхом кластеризації декількох серверів, навантаження між якими розподіляється за допомогою спеціальних методик з використання, як апаратних, так і програмних інструментів. *Мета*

дослідження – визначити можливі переваги та недоліки балансування навантаження на різних рівнях мережевої моделі OSI. *Задача* – полягає у порівнянні існуючих методів балансування навантаження, можливості їх комбінування та визначення області використання.

Зміст дослідження. Балансування навантаження - це метод для розподілу роботи між декількома обчислювальними ресурсами, такими як комп'ютери, кластери, мережі, центральні процесори та диски (носії інформації). Мета балансування полягає у оптимізації використання ресурсів, максимізація пропускної здатності, мінімізація часу відповіді та запобігання перенавантаження.

Рішення з балансування навантаження часто поділяють на дві категорії: L4 та L7, які відповідають транспортному та прикладному рівням мережевої моделі OSI. Стосовно категорії L4 балансування відбувається шляхом TCPз'єднання клієнта з балансувальником, який в свою чергу завершає з'єднання, обирає найбільш доступний сервер та встановлює з ним зв'язок. Таким чином всі маніпуляції відбуваються на рівні з'єднання/сеансу протоколів TCP/UDP.



Класична архітектура балансування навантаження

Слід зазначити, що балансувальник на рівні L4 робить одне вихідне TCP-з'єднання для кожного вхідного, приводячи до двох вхідних и двох вихідних з'єднань у ланцюгу клієнт-сервер. Розглянемо ситуацію, коли клієнт А відправляє 1 запит в хвилину, а клієнт Б – 50 запитів у секунду по вже встановленому з'єднанню. В результаті сервер, який був обраний для клієнта А, буде оброблювати приблизно в 3000 разів менше, ніж сервер обраний для клієнта Б. Тому безперервне TCP-з'єднання порушує перш за все мету балансування навантаження і є недоліком.

Балансування категорії L7 працює на прикладному рівні моделі OSI, орієнтовані на роботу з високорівневими протоколами такими як HTTP/HTTPS. В процесі балансування відбувається аналіз запитів клієнта та перенаправлення на різні сервери в залежності від характеру контенту (за такими принципами працює модуль Upstream веб-серверу Nginx). Оскільки L7 балансувальник виконує значно складніший аналіз, перетворення та маршрутизацію трафіку веб-додатку, тому навантаження на апаратну складову значно збільшується, що приводить до зменшення продуктивності

обробки навантаження (яка вимірюється у оброблених пакетах за секунду), порівняно з оптимізованими L4 балансувальниками. Слід зазначити про значну складність розробки алгоритмів аналізу трафіку, серед яких вірогідність програмної помилки значно більша.

Висновки. Підводячи підсумки можна зазначити, що важливу роль для сучасних мережевих протоколів відіграє балансувальники категорії L7. Хоча порівнюючи функціонально категорія L7 може повністю замінити L4, але кожний метод балансування виконує свою задачу і займає відповідне місце. Балансування категорії L4 використовується не менше, бо майже всі розподілені архітектури використовують дворівневу систему балансування навантаження L4/L7 для інтернет-трафіку. Тому для найбільш продуктивного балансування навантаження на сервери слід поєднувати можливості різних методів.

Список джерел:

1. Bourke T. Server Load Balancing // O'Reilly & Associates. – 2001. – V.126/ – N 2. – P. 182.

2. Introduction to modern network load balancing and proxying

[Електронний ресурс] – Режим доступу: www / URL: https://blog.envoyproxy.io/introduction-to-modern-network-load-balancing-andproxying-a57f6ff80236

Позначення	Найменування	Дод. відомості
	Текстові документи	
1 ГЮІК.505ХХХ.024 ПЗ	Пояснювальна записка	96 с
	Графічні документи	
2 ГЮІК. 505ХХХ.024 Д1	Графічна частина магістерської роботи	21 слайд

	Інші документи	
	Копія тез доповіді на конференції	1

					ГЮІК.505ХХХ.024 Д3			
Змін.	Арк.	Номер докум.	Підп.	Дата				
Розроб.		Ковріжний О.В.			Система балансування навантаження веб-застосунків на базі хмарної інфраструктури Відомість атестаційної роботи	Літ	Аркуш	Аркушів
Перевір.		Кривуля Г.Ф.				У	1	1
Н.контр.		Філіпенко І.В.				ХНУРЕ кафедра АПОТ		
Затв.		Чумаченко С.В.						