ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «СИСТЕМА УПРАВЛІННЯ МЕРЕЖЕВОЮ ІНФРАСТРУКТУРОЮ ДЛЯ ПІДТРИМКИ ВИСОКИХ НАВАНТАЖЕНЬ»

на здобуття освітнього ступеня магістра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні системи та мережі
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Антон САРАНЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КСДМ-61
Антон САРАНЕНКО
Ім'я, ПРІЗВИЩЕ
Керівник: Людмила АСЄЄВА
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Комп'ютерної інженерії
Ступінь вищої освіти - «Магістр»
Напрям підготовки - 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
"___"_____20__ року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Сараненко Антон Дмитрович

(прізвище, ім'я, по батькові)

1. Тема роботи: «Система управління мережевою інфраструктурою для підтримки високих навантажень»

Керівник роботи Асєєва Людмила Анатоліївна – phD (доктор філософії), доцент кафедри комп'ютерної інженерії

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від 15.10.2024 року № 320 .

2. Строк подання студентом роботи 22.12.2024 року

3. Вхідні дані до роботи:

3.1. Вимоги до кваліфікаційної роботи магістра з актуальних завдань.
3.2 Нормативні матеріали (стандарти).
3.3 Технічні вимоги.
3.4 Науково-технічна література з питань, пов'язаних з темою роботи.

. Зміст кваліфікаційної роботи (перелік питань, які потрібно розробити):

4.1. Аналіз мережевих інфраструктур та її оптимізації;
4.2 Методи та технології оптимізації мережевої інфраструктури;
4.3 Практична частина;
4.4 Висновки

5. Графічні матеріали.
1. Презентація

6. Дата видачі завдання _01.09.2024__

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми		Виконано
2	Підбір науково-технічної літератури		Виконано
3	Аналіз джерел		Виконано
4	Аналіз мережевих інфраструктур та їх оптимізації		Виконано
5	Методи та технології оптимізації мережевої інфраструктури		Виконано
6	Практична частина		Виконано
7	Підготовка вступу, висновків та реферату		Виконано
8	Попередній захист роботи		Виконано
9	Подання роботи в деканат		Виконано

Студент _____ _____Сараненко А.Д_____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____Лащевська Н. О.__
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи: 92 с., 41 рис., 13 табл., 1 дод., 25 джерел.

Мета роботи: дослідження методів та технологій оптимізації мережевої інфраструктури для забезпечення високої продуктивності та надійності роботи системи навіть у найскладніших умовах високих навантажень.

Об'єкт дослідження: процес оптимізації мережевої інфраструктури для підтримки високих навантажень.

Предмет дослідження: методи та технології балансування навантаження, управління трафіком, масштабованості, кешування та використання сучасних рішень, таких як SDN, DPI та алгоритми машинного навчання.

У представленій роботі розглянуто сучасні методи оптимізації мережевої інфраструктури, наведено аналіз актуальних підходів до управління трафіком та балансування навантаження, включаючи традиційні алгоритми (Round Robin, Least Connections) та новітні технології, такі як SDN і DPI. Показано актуальність застосування хмарних технологій та алгоритмів машинного навчання (ML) для підвищення продуктивності мережевих систем. Розроблено експериментальну модель, що демонструє ефективність впроваджених рішень.

Ключові слова: оптимізація, мережеві інфраструктури, балансування навантаження, управління трафіком, хмарні технології, кешування, Software-Defined Networking (SDN), Network Function Virtualization (NFV), масштабованість, Deep Packet Inspection (DPI), машинне навчання (ML).

ABSTRACT

Purpose: To research methods and technologies for optimizing network infrastructure to ensure high performance and reliability of the system even under the most challenging high-load conditions.

Object of Research: The process of optimizing network infrastructure to support high loads.

Subject of Research: Methods and technologies for load balancing, traffic management, scalability, caching, and the application of modern solutions such as SDN, DPI, and machine learning algorithms.

This study examines modern methods for optimizing network infrastructure, analyzes current approaches to traffic management and load balancing, including traditional algorithms (Round Robin, Least Connections) and cutting-edge technologies such as SDN and DPI. The relevance of cloud technologies and machine learning (ML) algorithms for enhancing the performance of network systems is demonstrated. An experimental model illustrating the effectiveness of the implemented solutions has been developed, and recommendations for their practical application are provided.

Keywords: optimization, network infrastructures, load balancing, traffic management, cloud technologies, caching, Software-Defined Networking (SDN), Network Function Virtualization (NFV), scalability, Deep Packet Inspection (DPI), machine learning (ML).

ЗМІСТ

ВСТУП

Сучасні інформаційно-комунікаційні системи переживають стрімке зростання обсягу переданих даних, що зумовлено швидким розвитком технологій, збільшенням кількості користувачів та обсягів трафіку. Це створює серйозні виклики для мережевої інфраструктури, яка повинна забезпечувати надійну і безперебійну роботу навіть за умов високих навантажень. Висока пропускна здатність та низька затримка є ключовими факторами для ефективної роботи таких мереж, особливо в умовах значного трафіку, який генерується бізнес-сервісами, інтернет-платформами та технологіями інтернету речей (IoT).

Актуальність теми дослідження обумовлена критичною важливістю оптимізації мережевої інфраструктури для забезпечення стабільної роботи різних бізнес-процесів, комерційних і державних структур, які залежні від швидкості та надійності передачі даних. Неоптимізована мережа може призводити до значних втрат через простої систем, порушення комунікацій та зниження якості надання послуг. Це є особливо актуальним для підприємств, які працюють з великими масивами даних у реальному часі, таких як хмарні сервіси, дата-центри, фінансові установи та онлайн-платформи.

У сучасному світі існує багато підходів до оптимізації мережевих інфраструктур, таких як застосування балансування навантаження, управління трафіком, хмарних технологій, кешування та програмно-визначених мереж (SDN). Такі технології дозволяють підвищити продуктивність, надійність та ефективність мереж. Зокрема, балансування навантаження є одним із ключових методів оптимізації, який розподіляє запити між серверами для забезпечення стабільної роботи мережі навіть під час високих навантажень.

Таким чином, оптимізація мережевої інфраструктури для підтримки високих навантажень є надзвичайно актуальним завданням. Це дозволяє не лише підвищити ефективність роботи мережі, але й забезпечити її стабільність, надійність та

швидкість передачі даних, що є ключовими вимогами для сучасних інформаційних систем.

Метою даного дослідження є оптимізація мережевої інфраструктури для забезпечення стабільної роботи під високими навантаженнями. У рамках дослідження було реалізовано тестове середовище на основі хмарних платформ із використанням Load Balancer та CloudFront. Проведено аналіз затримок, часу відповіді серверів та стабільності мережі під різними умовами.

Практична частина роботи включала розробку архітектурних моделей з балансуванням навантаження та оцінку їх ефективності за допомогою інструментів Apache JMeter. Результати дослідження підтвердили перевагу використання Load Balancer, кешування та інших технологій для забезпечення надійності системи навіть у складних умовах.

Дослідження також пропонує рекомендації щодо впровадження сучасних технологій, таких як SDN, хмарні сервіси та алгоритми машинного навчання (ML), для подальшої оптимізації мережевої інфраструктури.

1 АНАЛІЗ МЕРЕЖЕВИХ ІНФРАСТРУКТУР ТА ЇХ ОПТИМІЗАЦІЇ

1.1 Теоретичні основи побудови мережевих інфраструктур

Мережева інфраструктура — це сукупність апаратних і програмних засобів, які забезпечують обмін даними між комп’ютерами, пристроями та сервісами. Вона включає елементи фізичного рівня, такі як маршрутизатори, комутатори, кабелі, точки доступу, а також мережеві протоколи, що регулюють передачу даних. Основними принципами побудови мережевої інфраструктури є її масштабованість, надійність, продуктивність та безпека. Сучасні мережеві рішення повинні забезпечувати не лише високу пропускну здатність, але й відповідати потребам надійного керування трафіком і підвищеної безпеки.

Класична модель побудови мережі базується на багаторівневій архітектурі, яка включає три основні рівні: [1]

- **Користувацький рівень (Access Layer):** Забезпечує підключення кінцевих пристроїв, таких як комп'ютери, смартфони, сервери тощо, до мережі. Основні елементи цього рівня — комутатори та бездротові точки доступу. Важливими аспектами цього рівня є забезпечення надійного доступу кінцевих користувачів і керування локальним трафіком.
- **Рівень агрегації або розподілу (Distribution Layer):** Відповідає за об'єднання кількох підмереж і забезпечення маршрутизації між ними. Тут також можуть виконуватися функції управління політиками безпеки та балансування навантаження. Цей рівень оптимізує обробку міжсегментного трафіку та забезпечує комутацію між різними рівнями доступу.
- **Магістральний рівень (Core Layer):** Забезпечує високу швидкість передачі даних між різними частинами мережі, підтримуючи великі обсяги трафіку.

Магістральний рівень з'єднує інші рівні інфраструктури та забезпечує доступ до зовнішніх мереж, таких як Інтернет. Це особливо важливо для великих організацій, які вимагають надійної передачі даних на великі відстані з мінімальною затримкою.

1.1.1 Мережеві топології

Ключовою особливістю сучасних мережевих інфраструктур є підтримка різноманітних топологій, що дозволяє забезпечити гнучкість і надійність роботи мережі в умовах високих навантажень. Найбільш поширеними топологіями є [2]:

- **Топологія зірка:** Всі пристрої підключені до центрального вузла, який може бути комутатором або маршрутизатором. Така топологія спрощує керування мережею, оскільки всі з'єднання контролюються через єдиний центр. Проте ця топологія вразлива до відмови центрального вузла, що може призвести до втрати доступу до всієї мережі.
- **Кільцева топологія:** Кожен пристрій з’єднаний з двома сусідніми, утворюючи кільце. Перевага цієї топології полягає у відсутності потреби в центральному вузлі, що знижує ризик єдиної точки відмови. Однак у випадку порушення з'єднання між пристроями може виникати повна втрата зв’язку, якщо не передбачені додаткові механізми.
- **Сітчаста топологія (Mesh):** Кожен пристрій може бути з’єднаний з кількома іншими, що забезпечує високий рівень надійності. У разі відмови одного з вузлів трафік може бути перенаправлений іншими шляхами, що дозволяє уникати перебоїв у роботі. Це особливо важливо для критичних систем і великих інфраструктур.

1.1.2 Технології та протоколи передачі даних

Окрім фізичної архітектури, важливу роль у побудові мереж відіграють технології та протоколи передачі даних, які визначають правила взаємодії між пристроями в мережі. Найпоширенішими є такі:

- **Ethernet:** Це технологія дротових локальних мереж, яка визначає фізичний та канальний рівні передачі даних. Ethernet підтримує швидкість передачі даних від 10 Мбіт/с до 400 Гбіт/с і є основною технологією для побудови локальних мереж (LAN). Вона забезпечує високу продуктивність і стабільність мережі. Сучасні версії Ethernet підтримують використання як мідних, так і оптичних кабелів, що дозволяє досягати максимальних швидкостей передачі.
- **TCP/IP:** Це набір мережевих протоколів, який є основою роботи Інтернету та сучасних мереж. TCP/IP забезпечує маршрутизацію, поділ даних на пакети, їх доставку та перевірку цілісності. Завдяки своїй універсальності та масштабованості, TCP/IP використовується у мережах будь-якого масштабу — від локальних до глобальних.
- **Wi-Fi:** Це бездротова технологія передачі даних, яка використовує стандарти сімейства IEEE 802.11. Вона дозволяє пристроям підключатися до мережі без використання кабелів. Останні версії Wi-Fi (зокрема Wi-Fi 6) забезпечують швидкість до 9,6 Гбіт/с, що робить її однією з найзручніших технологій для забезпечення мобільності користувачів у локальних мережах.

Такі технології, як Ethernet та Wi-Fi, визначають способи фізичної передачі даних, тоді як протоколи, такі як TCP/IP, регулюють логіку обміну даними, забезпечуючи їхню доставку між пристроями.

1.1.3 Забезпечення безпеки мереж

Крім апаратної та програмної складової, мережі повинні підтримувати механізми безпеки для захисту переданих даних і ресурсів. Основні механізми включають:

— **Міжмережеві екрани (Firewall):** Захищають мережу від небажаного трафіку шляхом фільтрації пакетів за різними критеріями, такими як IP-адреси, порти та типи протоколів. Міжмережеві екрани запобігають несанкціонованим доступам до мережевих ресурсів.

— **Віртуальні приватні мережі (VPN):** Забезпечують безпечний доступ до ресурсів мережі через Інтернет, шифруючи передані дані. VPN-технології часто використовуються для захищеного доступу до корпоративних мереж з віддалених локацій.

1.2 Типи та характеристики мережевих інфраструктур

Мережеві інфраструктури можуть бути різних типів, залежно від їх масштабу, функціональності та потреб, зокрема LAN, WAN та VPN. Кожен з цих типів забезпечує різні рівні продуктивності, надійності та безпеки, що особливо важливо для організацій, які працюють з високими навантаженнями. Далі розглянемо основні типи та їх характеристики [3].

1.2.1 Локальні мережі (LAN)

LAN (Local Area Network) забезпечують зв'язок між пристроями в межах невеликої географічної території, зазвичай одного офісу або будівлі. Ці мережі використовують Ethernet або Wi-Fi для швидкої передачі даних. Вони є ідеальними для забезпечення високої пропускної здатності та низьких затримок у середовищах з інтенсивним обміном даних.

Характеристики LAN:

— Швидка передача даних (від 1 Гбіт/с і вище).

— Легкість у розгортанні та обслуговуванні.

— Залежність від фізичних кабелів або Wi-Fi для підключення.

— Оптимізація LAN важлива для зменшення затримок при обробці трафіку в реальному часі, особливо у випадку великого навантаження на сервери.

1.2.2 Глобальні мережі (WAN)

WAN (Wide Area Network) охоплюють великі відстані, з'єднуючи декілька локальних мереж. Основним завданням WAN є забезпечення зв'язку між офісами в різних містах або країнах. WAN використовують провайдерів телекомунікацій для передачі даних, що часто пов'язано з великими витратами і потребою в додаткових рішеннях для захисту.

Характеристики WAN:

— Велика географічна масштабованість.

— Низька швидкість передачі даних порівняно з LAN.

— Складність у забезпеченні безпеки та управління мережею.

— Оптимізація WAN включає балансування навантаження між віддаленими локаціями для мінімізації затримок і підвищення продуктивності за допомогою програмно-визначених мереж (SDN).

1.2.3 Віртуальні приватні мережі (VPN)

VPN (Virtual Private Network) забезпечує захищений канал передачі даних через загальнодоступні мережі, такі як Інтернет. VPN активно використовується для віддаленого доступу до корпоративних ресурсів, забезпечуючи безпеку шляхом шифрування даних.

Характеристики VPN:

— Захищений доступ до мережі через шифрування.

— Висока мобільність користувачів та можливість працювати з віддалених місць.

— Деяке зниження продуктивності через оверхед шифрування.

1.3 Проблеми, що виникають при високих навантаженнях

Сучасні мережі стикаються з багатьма викликами, коли працюють під високими навантаженнями. При зростанні кількості користувачів і обсягу переданих даних зростає ймовірність виникнення низки проблем, таких як затримки в передачі даних, перевантаження мережевих пристроїв і втрата пакетів. Ці проблеми можуть негативно впливати на продуктивність і стабільність мережевої інфраструктури. У цьому розділі буде розглянуто основні проблеми, що виникають під час високих навантажень, а також методи їхньої оптимізації для забезпечення стабільної роботи мережі.

1.3.1 Затримки в мережі

Затримки в мережі є однією з найпоширеніших проблем, що виникають при високих навантаженнях. Вони можуть бути викликані збільшенням обсягу трафіку, недостатньою пропускною здатністю каналів або неправильним управлінням трафіком. Затримка (latency) визначається як час, який потрібен пакету для досягнення кінцевої точки [4]. Високі затримки можуть призводити до зниження якості обслуговування (QoS), особливо для додатків, що потребують реального часу, таких як відеоконференції або онлайн-ігри.

Для зменшення затримок використовуються різні методи управління трафіком, такі як Quality of Service (QoS) і програмно-визначені мережі (SDN). QoS дозволяє пріоритизувати трафік, забезпечуючи більшу пропускну здатність для критичних сервісів, що допомагає знизити затримки для таких додатків, як VoIP або потокове відео.

1.3.2 Втрата пакетів

Втрати пакетів (packet loss) є серйозною проблемою, що виникає при перевантаженні мережі. Коли пристрої не в змозі обробити великий обсяг трафіку, пакети можуть бути втрачені, що призводить до зниження якості обслуговування, зокрема до проблем з передачею відео та аудіо. Втрата пакетів може виникати через недостатню пропускну здатність, перевантаження маршрутизаторів або комутаторів, а також через помилки у фізичних каналах [5].

Для мінімізації втрат пакетів широко використовуються технології балансування навантаження (load balancing) та маршрутизації з оптимізацією трафіку. Балансування навантаження дозволяє рівномірно розподіляти трафік між різними серверами або маршрутизаторами, зменшуючи ймовірність перевантаження будь-якого окремого елементу мережі.

1.3.3 Перевантаження мережевих пристроїв

Мережеві пристрої, такі як маршрутизатори та комутатори, можуть бути перевантажені великим обсягом трафіку. Це перевантаження може призводити до зниження їхньої продуктивності та збільшення часу обробки пакетів. Однією з причин перевантаження є недостатня масштабованість мережевої інфраструктури, що робить її вразливою до зростання кількості користувачів або додатків.

Одним із сучасних рішень цієї проблеми є впровадження програмно-визначених мереж (SDN), які дозволяють централізовано управляти всіма елементами мережі. SDN забезпечує більш ефективне управління трафіком і дозволяє динамічно перенаправляти його між різними елементами мережі залежно від їхньої поточної завантаженості.

1.3.4 Масштабованість

Проблеми з масштабованістю виникають, коли мережа не може впоратися зі збільшенням кількості користувачів або сервісів. Це може бути викликано обмеженими ресурсами апаратного забезпечення або застарілою інфраструктурою, яка не підтримує сучасні протоколи передачі даних.

Для вирішення цієї проблеми використовуються різні підходи до масштабування, такі як горизонтальне та вертикальне масштабування мереж. Горизонтальне масштабування полягає в додаванні нових пристроїв до мережі, що дозволяє розподіляти навантаження на більшу кількість ресурсів. Вертикальне масштабування передбачає покращення продуктивності існуючих пристроїв шляхом їхнього оновлення або модернізації.

1.3.5 Вирішення проблем за допомогою Deep Packet Inspection (DPI)

Одним з ключових методів покращення ефективності мережевої інфраструктури під високими навантаженнями є використання технології Deep Packet Inspection (DPI) [6]. DPI дозволяє аналізувати вміст кожного пакета даних, що проходить через мережу, на більш глибокому рівні, що дозволяє ідентифікувати та фільтрувати небажані пакети. Це особливо корисно для захисту мережі від кіберзагроз та оптимізації трафіку.

DPI також може бути використано для поліпшення балансування навантаження. Замість того, щоб просто перенаправляти трафік, система може аналізувати вміст пакетів і визначати, які з них потребують негайної обробки, а які можуть бути відкладені. Це дозволяє значно зменшити затримки та втрати пакетів у критичних додатках.

1.3.6 Ефективність використання пропускної здатності

Оптимізація використання пропускної здатності є критичною для підтримки високої продуктивності мережі. Недостатнє використання пропускної здатності може призводити до перевантаження каналів зв'язку та підвищення затримок, тоді як надмірне використання може створювати вузькі місця та знижувати загальну продуктивність мережі. Методи оптимізації включають впровадження механізмів компресії даних, пріоритизації трафіку та використання технологій мультиплексування.

Стратегії оптимізації:

— Використання протоколів компресії даних для зменшення обсягу переданих даних.
— Пріоритизація важливих типів трафіку за допомогою QoS.
— Використання мультиплексування для одночасної передачі кількох потоків даних через один канал.

1.3.7 Управління якістю обслуговування (QoS)

Якість обслуговування (Quality of Service, QoS) є ключовим аспектом управління мережею, особливо в умовах високих навантажень. QoS дозволяє визначати пріоритети для різних типів трафіку, забезпечуючи належний рівень продуктивності для критичних додатків, таких як VoIP, відеоконференції та потокові сервіси. Це досягається шляхом управління пропускною здатністю, затримками та втратами пакетів для забезпечення стабільної роботи мережі навіть під високими навантаженнями [7].

Принципи роботи QoS:

— Класифікація трафіку: QoS здійснює класифікацію вхідного трафіку, визначаючи його пріоритет на основі встановлених правил. Дані можуть бути класифіковані на основі вмісту заголовка IP-пакетів, адрес, типу протоколу (TCP, UDP), призначених портів або спеціальних міток (Tagging) — наприклад, Differentiated Services Code Point (DSCP) або VLAN.

— Маркування трафіку: Для відстеження пріоритетності, деякі пакети можуть бути позначені спеціальними полями, наприклад, DSCP у заголовку IP-пакета. Це дозволяє мережевим пристроям, як маршрутизаторам і комутаторам, ідентифікувати цей трафік і обробляти його відповідно до встановлених пріоритетів.

— Черги (Queuing): QoS використовує механізми черг для обробки трафіку з різними рівнями пріоритету. Для найвищих пріоритетів (голос, відео) трафік може отримувати перший доступ до ресурсів мережі. Для менш пріоритетного трафіку, як-от електронна пошта чи веб-трафік, можуть бути використані черги з нижчим пріоритетом.

— Traffic Shaping: Це процес обмеження швидкості передачі даних для окремих потоків, щоб уникнути перевантаження мережі. Наприклад, якщо інтерфейс підтримує передачу лише до 100 Мбіт/с, полірування може обмежити передачу відео на рівні 10 Мбіт/с, залишаючи інші ресурси для важливіших сервісів.

— Congestion Avoidance: QoS застосовує активне управління заторами для виявлення можливих перевантажень і запобігання їм. Це досягається за допомогою алгоритмів, таких як Random Early Detection (RED), які виявляють пікові навантаження і блокують деякі пакети до того, як мережа буде перевантажена.
— Traffic Policing: Випадки, коли затримка є критичною (як у відеоконференціях), можуть бути оброблені за допомогою обмежень для менш важливих потоків, з метою виділення ресурсу для високопріоритетного трафіку.

Ключові аспекти управління QoS:

— Класифікація трафіку на основі типу додатків та їхніх вимог.
— Визначення політик пріоритетності для різних класів трафіку.
— Впровадження механізмів маркування та чергування для забезпечення належного обслуговування.

1.4 Сучасні тенденції в мережевій інфраструктурі

1.4.1 Software-Defined Networking (SDN)

Software-Defined Networking (SDN) є однією з найважливіших сучасних тенденцій у розвитку мережевих інфраструктур. SDN розділяє контрольну площину (control plane) та площину передачі даних (data plane), дозволяючи централізовано керувати мережею через програмне забезпечення [8]. Це забезпечує гнучкість, масштабованість та ефективність управління мережевими ресурсами.

Роль SDN у сучасних мережах:

— Централізоване управління: SDN-контролери мають глобальне бачення мережі, що дозволяє приймати оптимальні рішення щодо маршрутизації та балансування навантаження.
— Динамічне налаштування: Мережа може бути швидко налаштована відповідно до змінних умов, забезпечуючи адаптивність та гнучкість у реальному часі.
— Інтеграція з ML: SDN дозволяє інтегрувати алгоритми машинного навчання для прогнозування та оптимізації трафіку, підвищуючи ефективність мережевих операцій.

1.4.2 Network Function Virtualization (NFV)

Network Function Virtualization (NFV) є ще однією важливою технологією, яка сприяє оптимізації мережевої інфраструктури. NFV дозволяє віртуалізувати мережеві функції, такі як брандмауери, балансувальники навантаження та системи виявлення загроз, що дозволяє запускати їх на стандартному обладнанні без необхідності використання спеціалізованих апаратних пристроїв [9].

Роль NFV у сучасних мережах:

— Зменшення витрат: Віртуалізація мережевих функцій дозволяє зменшити витрати на обладнання та його обслуговування.
— Масштабованість: NFV забезпечує легке масштабування мережевих функцій відповідно до потреб, що особливо важливо у великих мережах з високими навантаженнями.
— Гнучкість: NFV дозволяє швидко впроваджувати нові мережеві сервіси та функції без необхідності оновлення апаратного забезпечення.

2 МЕТОДИ ТА ТЕХНОЛОГІЇ ОПТИМІЗАЦІЇ МЕРЕЖЕВОЇ ІНФРАСТРУКТУРИ

У цьому розділі досліджуються ключові методи та технології, спрямовані на оптимізацію мережевої інфраструктури для забезпечення стабільної роботи під високими навантаженнями. Розглядаються підходи до управління трафіком, балансування навантаження, а також методи забезпечення ефективності та надійності мережевих систем.

Розділ також включає аналіз сучасних підходів до масштабованості мереж, інтеграції хмарних технологій та використання програмно-визначених мереж (SDN). Висвітлюються переваги впровадження цих рішень для досягнення високої продуктивності та гнучкості мережевої інфраструктури.

2.1 Кешування (Caching)

Кешування на ряду з QoS є одним з ключових механізмів для оптимізації передачі даних в мережах і забезпечення продуктивності в умовах високих навантажень. Основна ідея кешування полягає у тимчасовому збереженні часто запитуваних даних в проміжних сховищах (кешах) для швидкого доступу. Завдяки цьому можна значно зменшити кількість звернень до оригінальних джерел даних, знизити затримки та збільшити пропускну здатність [11].

2.1.1 Принципи роботи кешування:

— Тимчасове збереження: Кеші зберігають дані, що часто запитуються, у пам'яті з високою швидкістю доступу (наприклад, в оперативній пам'яті або SSD).

— Пріоритизація важливих даних: Дані з високим пріоритетом, такі як часто запитувані веб-сторінки, зберігаються у кешах, що дозволяє уникнути затримок.
— Евакуаційні алгоритми (Eviction Policies): Коли кеш заповнений, старі або менш важливі дані витісняються за допомогою алгоритмів, таких як Least Recently Used (LRU) або Least Frequently Used (LFU), для звільнення місця для нових даних.

2.1.2 Види кешування:

Локальне кешування: Локальні кеші на рівні браузера зберігають статичний контент (HTML, CSS, JavaScript) для зниження затримок при повторному завантаженні. Це ефективно для зменшення навантаження на сервери, особливо в умовах великих обсягів однотипних запитів.

Переваги локального кешування:

— Зменшення затримки до кількох мілісекунд.
— Зниження навантаження на сервери.

Серверне кешування: Кеші на рівні серверів часто використовуються для збереження результатів складних запитів до баз даних. Наприклад, кеші, які зберігають результати запитів до SQL-бази даних у пам’яті (RAM), дозволяють значно знизити час відповіді, оскільки запити до кешованих даних можуть бути оброблені в мікросекундах, тоді як доступ до бази даних може займати секунди.

Розподілене кешування: Розподілені кеші, такі як Redis або Amazon S3, забезпечують кешування даних у великих масштабах, розподіляючи збережені дані між кількома серверами або навіть дата-центрами. Це дозволяє забезпечити надійність, масштабованість та доступність даних у великих мережевих інфраструктурах.

Приклади розподіленого кешування:

— Redis: Забезпечує зберігання ключів-значень та складних структур даних, таких як списки та множини, з мінімальною затримкою.

Підтримує TTL (Time to Live) для автоматичного видалення застарілих даних.

— Amazon S3: Використовується для зберігання великих об'єктів, таких як медіа-файли, з можливістю масштабованого доступу до них.

Таблиця 2.1: Показники ефективності кешування

Показник	**Опис**
Cache Hit Ratio	Відсоток запитів, що обробляються з кешу без необхідності звернення до основного джерела
Затримка доступу (Latency)	Час, необхідний для отримання кешованих даних. Оптимально цей показник повинен бути нижчим за 10 мс
Пропускна здатність (Throughput)	Обсяг даних, який може бути оброблений системою з використанням кешування. Пропускна здатність підвищується завдяки зниженню кількості звернень до оригінальних серверів

Таблиця 2.2: Порівняння Cache Hit Rate для різних моделей кешування при різних розмірах кешу

Модель кешування	**0.1% Cache Size**	**0.5% Cache Size**	**5% Cache Size**	**25% Cache Size**
Proxy	60%	65%	70%	72%
Client	47.3%	55%	60%	70.26%
Hierarchy	62%	68%	75%	76%
Hierarchy-P2P	68%	75%	80%	82%
Single	70%	78%	85%	85%

Ця таблиця представляє Cache Hit Rate для різних моделей кешування при різних розмірах кешу. Дані отримані на основі симуляцій, описаних у статті "Exploiting Client Cache: A Scalable and Efficient Approach to Build Large Web Cache" авторів Zhiyong Xu, Yiming Hu та Laxmi Bhuyan [12]

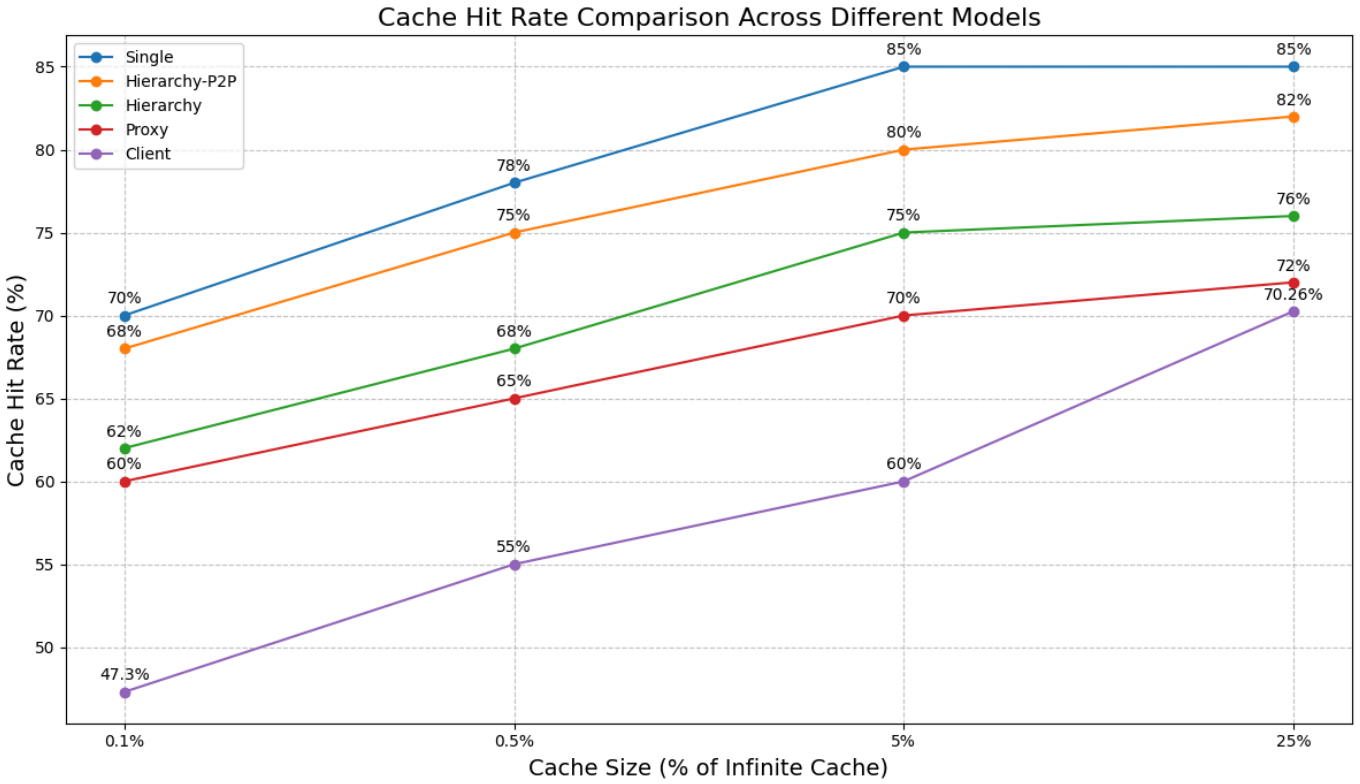


Рисунок 2.1 - Порівняння Cache Hit Rate для різних моделей кешування

2.1.3 Моделі кешування

Proxy Model

У Proxy Model кеші розміщуються на рівні серверів-проксі, які обробляють запити клієнтів. Ця модель ефективна для організацій з великим числом клієнтів, оскільки дозволяє централізовано управляти кешем, знижуючи навантаження на оригінальні веб-сервери та підвищуючи швидкість доступу до даних.

Таблиця 2.3: Переваги та недоліки Proxy Model

Переваги Proxy Model	**Недоліки Proxy Model**
Централізоване управління кешем	Може стати вузьким місцем при високих навантаженнях
Зменшення навантаження на оригінальні сервери	Вимагає наявності потужних серверів-проксі
Підвищення швидкості доступу до часто запитуваних даних	

Client Model

Client Model передбачає використання локальних кешів на клієнтських комп'ютерах для зберігання часто запитуваного контенту. Це дозволяє знизити затримки при повторних запитах до тих самих ресурсів.

Таблиця 2.4: Переваги та недоліки Client Model

Переваги Client Model	**Недоліки Client Model**
Зменшення затримок завдяки локальному збереженню даних	Обмежена ємність кешу на кожному клієнті
Зниження навантаження на сервери	Відсутність кооперації між клієнтами може призвести до дублювання даних
Підвищення швидкості обробки запитів	

Hierarchy Model

Hierarchy Model створює ієрархічну структуру кешів, де кеші організовані в кластери з суперклієнтами, які відповідають за управління кешем у своїх кластерах. Це дозволяє розподіляти навантаження та покращувати масштабованість системи.

Таблиця 2.5: Переваги та недоліки Hierarchy Model

Переваги Hierarchy Model	Недоліки Hierarchy Model
Розподіл навантаження між суперклієнтами: Забезпечує рівномірний розподіл ресурсів.	**Можливі проблеми з відмовами суперклієнтів:** При відмові одного з суперклієнтів можуть виникнути збої в роботі.
Зменшення навантаження на центральний проксі-сервер: Ефективно знижує навантаження.	**Дублювання файлів у кластері**: Через відсутність ефективного управління кешем збільшується витрата ресурсів.
Покращена масштабованість системи: Дозволяє легко додавати нові рівні або вузли в ієрархію.	**Складність адміністрування**: Чим більше рівнів у моделі, тим важче керувати системою.
Ефективна організація ресурсів: Ієрархічна структура сприяє швидкому доступу до даних. адміністрування: Чим більше рівнів у моделі, тим важче керувати системою.	**Збільшення затримок**: Ієрархія може створювати додаткову затримку при обробці запитів.

Hierarchy-P2P Model

Hierarchy-P2P Model поєднує ієрархічну структуру з технологіями peer-to-peer, де суперклієнти організовують маршрутизацію та управління кешем у своїх кластерах за допомогою P2P-протоколів. Це дозволяє покращити кооперацію між клієнтами та знизити навантаження на суперклієнтів.

Таблиця 2.6: Переваги та недоліки Hierarchy-P2P Model

Переваги Hierarchy-P2P Model	Недоліки Hierarchy-P2P Model
Ефективна кооперація між клієнтами	Складність реалізації та управління
Зменшення навантаження на суперклієнтів	Можливі проблеми з балансуванням навантаження при збільшенні кількості клієнтів
Покращення масштабованості та надійності системи	

Single Model передбачає єдиний великий кеш, який управляється суперклієнтами без дублювання файлів у кластерах. Це дозволяє максимально ефективно використовувати кеш-простір та підвищити Cache Hit Rate.

Таблиця 2.7: Переваги та недоліки Single Model

Переваги Single Model	Недоліки Single Model
Ефективна кооперація між клієнтами	Складність реалізації та управління
Зменшення навантаження на суперклієнтів	Можливі проблеми з балансуванням навантаження при збільшенні кількості клієнтів
Покращення масштабованості та надійності системи	

2.2 Балансування навантаження

Балансування навантаження є важливою технологією в мережевій інфраструктурі, яка дозволяє рівномірно розподіляти трафік і обчислювальні ресурси між різними серверами або мережевими пристроями. Цей підхід забезпечує підвищену надійність, стабільність і продуктивність мережі, запобігаючи перенавантаженням окремих компонентів системи. У сучасних умовах, коли обсяги даних зростають експоненційно, правильне балансування трафіку стає критичним для підтримки високої якості обслуговування (QoS), зниження затримок та запобігання втратам пакетів.

2.2.1 Традиційні методи балансування навантаження

Традиційні алгоритми балансування навантаження, такі як Round Robin та Least Connections, використовуються для розподілу запитів між серверами на основі

різних критеріїв. Ці алгоритми є фундаментальними у багатьох мережах і кластерних обчислювальних системах.

Алгоритм Round Robin

Round Robin — це один із найбільш базових і поширених алгоритмів для розподілу навантаження. Його концепція була розроблена на основі класичного механізму черги з рівномірним розподілом завдань, що походить ще з операційних систем. Алгоритм по черзі направляє кожен новий запит до наступного сервера у списку, не враховуючи поточне навантаження на сервери або складність запитів [14].

Принцип роботи:

— Статичний цикл: Сервери організовані в циклічний список. Кожен сервер отримує запит по черзі. Після останнього сервера цикл починається знову.

— Рівномірний розподіл: Запити розподіляються рівномірно без урахування поточної продуктивності або стану серверів.

— Простота впровадження: Алгоритм не вимагає складного моніторингу стану системи та легко реалізується в програмному забезпеченні для балансування навантаження.

Основні характеристики:

— Підходить для гомогенних систем: Round Robin найкраще працює в умовах, де всі сервери мають однакові апаратні характеристики та обробляють запити приблизно з однаковою швидкістю.

— Легке впровадження: Завдяки простій логіці, алгоритм використовується у багатьох стандартних системах балансування, таких як NGINX, HAProxy та інші рішення для кластерів і веб-додатків.

Приклад використання: Round Robin є основою багатьох мережевих рішень для вебсерверів і додатків, де є великий потік запитів, таких як статичні вебсайти або розподілені обчислювальні платформи. Наприклад, балансувальники навантаження у рішеннях від NGINX часто використовують цей алгоритм як один з

базових варіантів для простих середовищ, де всі сервери мають однакову потужність.

Недоліки:

— Ігнорування поточного стану серверів: Алгоритм не враховує навантаження на кожен окремий сервер, тому в середовищах з різними типами запитів або неоднорідними ресурсами він може призвести до перевантаження деяких серверів.

— Недоліки для динамічних середовищ: У середовищах, де навантаження та продуктивність серверів можуть змінюватися, Round Robin стає менш ефективним, оскільки не враховує поточну завантаженість кожного вузла.

Алгоритм Least Connections

Алгоритм Least Connections був розроблений як рішення для більш динамічних систем, де запити можуть суттєво відрізнятися за часом обробки, а сервери можуть мати різну потужність. Основний принцип полягає у тому, щоб нові запити направляти до сервера, який має найменшу кількість активних з'єднань. Цей алгоритм дозволяє краще балансувати навантаження, оскільки бере до уваги поточний стан мережевих вузлів [14].

Принцип роботи:

— Моніторинг активних з'єднань: Балансувальник навантаження постійно відстежує кількість активних з'єднань на кожному сервері.

— Динамічне розподілення запитів: Запити спрямовуються до серверів, які мають найменшу кількість активних з'єднань у момент надходження нового запиту.

— Підтримка неоднорідних середовищ: Least Connections підходить для середовищ, де сервери можуть мати різні потужності, або коли запити відрізняються за часом виконання.

Основні характеристики:

— Гнучкість: Цей метод дозволяє ефективно використовувати ресурси навіть у системах з різними характеристиками серверів.

— Збалансованість навантаження: За рахунок динамічного моніторингу кількості з'єднань алгоритм гарантує рівномірний розподіл навантаження.

Приклад використання: Алгоритм Least Connections часто використовується у великих мережевих середовищах, таких як дата-центри та хмарні платформи. AWS Elastic Load Balancer використовує цей підхід у поєднанні з іншими стратегіями для оптимального управління запитами у хмарних сервісах. Інші приклади впровадження можна знайти у рішеннях від F5 Networks, які використовують Least Connections для великих корпоративних мереж з неоднорідними ресурсами.

Недоліки:

— Потреба в обчислювальних ресурсах: Вимога постійного моніторингу з'єднань може збільшувати навантаження на систему управління трафіком.

— Затримки у розподілі: Може виникати затримка у випадку, якщо сервери швидко змінюють свій стан (від активного до пасивного або навпаки).

Порівняння Round Robin та Least Connections

Таблиця 2.8: Порівняння Round Robin та Least Connections

Характеристика	Round Robin	Least Connections
Простота	Дуже простий у реалізації	Потребує додаткового моніторингу
Розподіл навантаження	Рівномірний	Динамічний, базується на поточному стані
Продуктивність	Підходить для однорідних систем	Ефективний у неоднорідних середовищах
Приклад використання	NGINX, HAProxy	AWS ELB, F5 Networks

Висновки щодо традиційних методів

Традиційні алгоритми балансування навантаження, такі як Round Robin та Least Connections, заклали основу для управління трафіком у сучасних мережах. Хоча вони й ефективні у багатьох ситуаціях, зокрема в однорідних системах або невеликих середовищах, вони можуть бути недостатніми для складних мереж з різними типами трафіку та серверів. У зв'язку з цим, на сучасному етапі розвитку інфраструктури такі методи часто поєднуються або замінюються інноваційними рішеннями, такими як програмно-визначені мережі (SDN) та алгоритми, що використовують машинне навчання.

2.2.2 Балансування навантаження для програмно-визначених мереж (SDN)

Інновації в управлінні трафіком через SDN

Завдяки централізованому підходу, SDN пропонує значні переваги у процесі балансування навантаження. У традиційних мережах кожен маршрутизатор або комутатор працює автономно, що ускладнює динамічне управління потоками трафіку. У SDN же вся логіка управління винесена до окремого SDN-контролера, який має глобальне бачення мережі і може приймати більш обґрунтовані рішення щодо розподілу ресурсів і перенаправлення трафіку в реальному часі.

SDN забезпечує динамічне балансування навантаження шляхом використання інформації про поточний стан мережі, що дозволяє не лише ефективніше використовувати ресурси, але й зменшувати затримки, перевантаження та втрати пакетів. Наприклад, контролер може перенаправляти трафік через менш завантажені маршрути або вузли, забезпечуючи рівномірний розподіл трафіку між доступними ресурсами.

Протоколи для реалізації SDN

Одним з найпоширеніших протоколів для реалізації SDN є OpenFlow, який дозволяє контролеру безпосередньо взаємодіяти з комутаторами та маршрутизаторами, керуючи їхньою поведінкою. OpenFlow підтримує точкові

рішення щодо обробки кожного пакета: контролер може встановлювати правила для маршрутизації трафіку, змінювати шляхи або змінювати пріоритети, що значно підвищує гнучкість управління мережею.

У OpenFlow-мережах контролер отримує інформацію про стан кожного мережевого елемента і використовує її для оптимізації трафіку, наприклад, перенаправляючи потоки до менш завантажених вузлів. Цей механізм також підтримує використання різних алгоритмів балансування навантаження, таких як Least Connections або алгоритми на основі машинного навчання.

Таблиця 2.9: Порівняльна таблиця протоколів SDN

Характеристика	**OpenFlow**	**NETCONF**	**OVSDB**	**P4**
Основні Функції	Управління потоками, встановлення правил маршрутизації	Налаштування пристроїв мережі	Управління конфігураціями мережевих елементів	Програмування поведінки мережевих елементів
Підтримка	Широко підтримується різними виробниками	Використовується для управління конфігураціями	Використовуєтьс я для управління базами даних комутаторів	Дозволяє високий рівень кастомізації
Переваги	Простота інтеграції, гнучкість	Стандартизований підхід до управління	Ефективне управління конфігураціями	Висока програмованіст ь
Недоліки	Обмежена функціональні сть поза управління потоками	Менш оптимізований для високошвидкісни х операцій	Обмежена функціональніст ь для складних сценаріїв	Висока складність розробки

Впровадження у реальних системах

Багато сучасних компаній інтегрують SDN у свої рішення для управління мережами та балансування навантаження. Приклади використання SDN можна знайти в продуктах таких компаній, як:

— Cisco: рішення Cisco ACI (Application Centric Infrastructure) підтримує інтеграцію SDN для забезпечення динамічного балансування навантаження та оптимізації мережевого трафіку. Завдяки централізованому контролеру, Cisco ACI забезпечує автоматизацію управління мережею та ефективне розподілення ресурсів залежно від поточного стану інфраструктури.

— Fortigate: рішення SD-WAN від Fortigate використовує концепцію SDN для оптимізації використання різних каналів зв’язку та балансування навантаження між ними. Fortigate SD-WAN дозволяє динамічно керувати мережею, забезпечуючи надійне та швидке передавання даних з мінімальними затримками, що особливо важливо для корпоративних VPN-мереж і хмарних сервісів.

Серед найпопулярніших контролерів є POX і RYU, які часто використовуються для розробки, тестування і прототипування SDN-рішень. Розуміння відмінностей між цими контролерами та їх порівняння з комерційними рішеннями, такими як Cisco ACI та Fortigate, є важливим для вибору відповідної архітектури для мережі.

POX і RYU

POX і RYU є open-source контролерами, які підтримують стандартний протокол OpenFlow. Вони розроблені для тестування та створення мережевих додатків, що працюють в SDN-середовищі. Основна їхня мета — це забезпечити швидку розробку SDN-рішень для академічних або дослідницьких середовищ, а також малих або середніх мережевих інфраструктур.

— **POX:** Python-версія більш старого контролера NOX, орієнтована на прототипування мережевих рішень і використовує просту інтерфейсну модель для управління потоками в мережі. Підтримує Linux, Windows та Mac OS, що

робить його гнучким для використання в різних операційних середовищах. Ідеальний для досліджень та навчальних проєктів [15].

— **RYU:** Python-контролер, розроблений NTT Labs, який пропонує ширший набір функцій для роботи з OpenFlow. Однією з ключових переваг RYU є його висока продуктивність, що робить його ідеальним для більш складних SDN-середовищ, які вимагають низької затримки і високої пропускної здатності. Підтримує лише Linux, але завдяки своїм API дозволяє створювати більш гнучкі мережеві програми [16].

Таблиця 2.10: Порівняльна таблиця протоколів POX, RYU, Cisco ACI, Fortigate SD-WAN

Параметр	**POX**	**RYU**	**Cisco ACI**	**Fortigate SD-WAN**
Основна платформа	Linux, Windows, Mac OS	Linux	Приватна (Cisco Nexus)	Приватна
Мова програмування	Python	Python	Приватна (APIC)	Приватна
Продуктивність	Середня	Висока	Висока	Висока
Підтримка OpenFlow	Так	Так	Частково	Частково
Інтеграція з ML	Обмежена	Розширена	Вбудована підтримка	Вбудована підтримка
Призначення	Дослідження, прототипи	Продуктивні системи	Корпоративні мережі	Корпоративні мережі

Дослідження продуктивності контролерів POX і RYU показало, що контролер RYU забезпечує кращу продуктивність шляхом вищої пропускної здатності та меншого часу відгуку. У тестах на пропускну здатність (через TCP-трафік)

контролер RYU перевершив POX на 1.24% до 5.35%, демонструючи здатність обробляти більшу кількість даних за одиницю часу. Крім того, контролер RYU також продемонстрував менші затримки, зменшуючи час відгуку на 0.5-1 мілісекунду у порівнянні з POX, а також забезпечив кращий показник джиттеру — на 0.02 мс нижчий ніж у POX [17, с. 11].

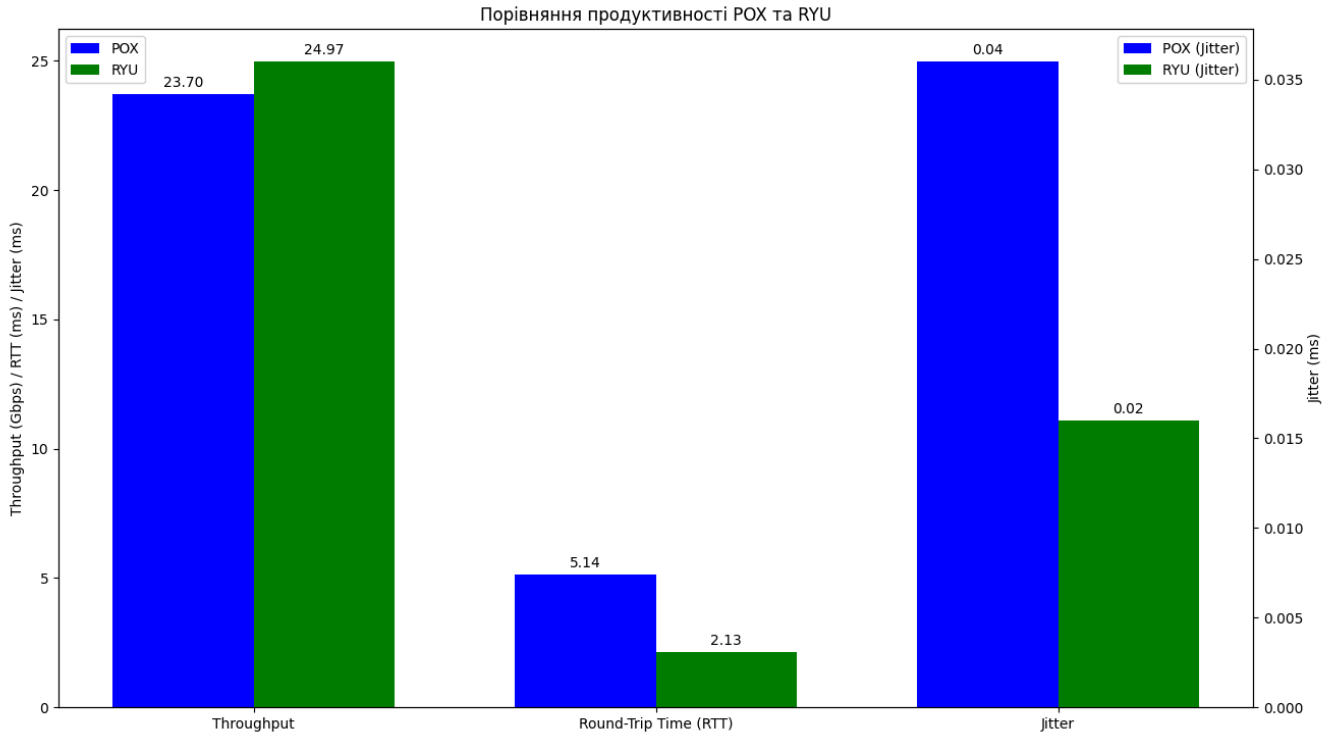


Рисунок 2.2 - Порівняння продуктивності POX та RYU

Це робить RYU перспективним вибором для систем, де важлива висока пропускна здатність та низькі затримки, таких як системи реального часу або хмарні обчислення. Контролер POX, з іншого боку, може бути більш гнучким у плані підтримки різних платформ, оскільки він підтримує Linux, Windows та Mac OS, що робить його корисним для тестових середовищ або невеликих SDN-рішень.

Це порівняння підкреслює важливість вибору відповідного SDN-контролера залежно від специфіки застосування, оскільки різні контролери можуть забезпечити різний рівень якості обслуговування (QoS).

Переваги та недоліки SDN для балансування навантаження:

Переваги:

— Глобальне бачення мережі: SDN-контролер має повну інформацію про стан мережі та може приймати рішення на основі загальної картини, що робить управління ефективнішим.

— Динамічне перенаправлення трафіку: Контролер може перенаправляти потоки залежно від навантаження на конкретні вузли, що запобігає їх перевантаженню.

— Інтеграція з алгоритмами машинного навчання: SDN-системи можуть використовувати ML-алгоритми для прогнозування пікових навантажень та автоматичного коригування параметрів мережі.

Недоліки:

— Залежність від контролера: Якщо контролер виходить з ладу, уся мережа може зазнати збоїв.

— Складність впровадження: Впровадження SDN у традиційні мережі може вимагати значних інвестицій і технічних ресурсів.

Таким чином, SDN пропонує значні переваги для балансування навантаження у сучасних мережах, особливо в умовах високих навантажень і великої кількості користувачів. Впровадження таких рішень, як OpenFlow, Cisco ACI та Fortigate SD-WAN, дозволяє забезпечити більш надійну та продуктивну мережеву інфраструктуру.

2.2.3 Використання Deep Packet Inspection (DPI) для балансування навантаження

Історія та розвиток DPI

Deep Packet Inspection (DPI) — це технологія аналізу мережевого трафіку, яка зародилася у відповідь на зростаючі потреби у безпеці та ефективному управлінні мережею. Спочатку вона використовувалася в системах мережевої безпеки, таких як брандмауери та системи виявлення атак (IDS), для глибшого аналізу трафіку. З часом, у міру розвитку мереж та збільшення складності трафіку, DPI знайшла

застосування у різних аспектах мережевої інфраструктури, зокрема для балансування навантаження та оптимізації мереж.

Поява DPI стала можливою завдяки збільшенню обчислювальних потужностей пристроїв, що дозволило аналізувати кожен пакет даних у реальному часі, не завдаючи значного впливу на продуктивність мережі. Сьогодні DPI є ключовою технологією, що використовується в багатьох продуктах, таких як Fortinet, Cisco, Palo Alto Networks та інших постачальників рішень для кібербезпеки та мережевої оптимізації.

Принципи роботи DPI

Основний принцип роботи DPI полягає в аналізі даних на рівнях, які виходять за межі заголовків пакетів (основні адресні та протокольні дані). Технологія дозволяє виявляти конкретний вміст пакетів, включаючи дані, що передаються додатками, інформацію про користувачів, типи файлів та навіть специфічні сигнатури шкідливих програм.

DPI функціонує через декілька основних етапів:

— Збір трафіку: DPI захоплює трафік, що проходить через мережу, на певному вузлі або пристрої (наприклад, маршрутизатори чи брандмауері).

— Аналіз заголовків та корисного навантаження: Після захоплення пакету, технологія аналізує не лише заголовки протоколів, а й корисне навантаження, що дає можливість отримати інформацію про вміст пакета.

— Ідентифікація протоколів та додатків: DPI може ідентифікувати конкретні протоколи або додатки, навіть якщо вони використовують нестандартні порти чи обфускацію для приховування своєї активності.

— Реалізація політик: На основі отриманої інформації, DPI приймає рішення щодо подальшої обробки пакету. Це може бути блокування, перенаправлення або пріоритизація трафіку.

Ця можливість глибокого аналізу робить DPI ефективним інструментом для різних завдань, таких як виявлення шкідливих програм, фільтрація контенту та оптимізація мережевого трафіку.

Методи реалізації DPI

DPI використовує кілька підходів для аналізу трафіку та виявлення аномалій. Основними методами є:

— Аналіз сигнатур: Система DPI порівнює вміст пакетів з базою даних сигнатур шкідливих програм або порушень політик безпеки. Це ефективний метод виявлення відомих загроз.

— Аномальний аналіз: DPI аналізує мережевий трафік для виявлення відхилень від нормальної поведінки, що може свідчити про присутність нових загроз або аномалій у роботі мережі.

— Інспекція на основі патернів: Цей метод дозволяє виявляти конкретні шаблони або патерни у трафіку для класифікації типів додатків або протоколів.

— Аналіз глибини пакетів: Це дозволяє ідентифікувати та контролювати специфічні типи даних (наприклад, відео, аудіо або текстові повідомлення) і, відповідно, виконувати необхідні дії для оптимізації трафіку або блокування небажаних елементів.

Використання DPI для балансування навантаження

DPI відіграє вирішальну роль у сучасних системах балансування навантаження, оскільки забезпечує набагато глибшу видимість трафіку, що дозволяє приймати інтелектуальні рішення щодо маршрутизації даних. Традиційні алгоритми балансування, такі як Round Robin або Least Connections, обмежені у здатності аналізувати типи трафіку. DPI, на відміну від них, дозволяє аналізувати кожен пакет на предмет типу даних та його пріоритету, що дає змогу збалансувати навантаження більш ефективно [18].

DPI також дозволяє використовувати методи інтелектуального балансування, при яких важливі дані (наприклад, VoIP або відеоконференції) можуть отримувати

більш високий пріоритет і передаватися з мінімальними затримками. Це досягається завдяки можливості DPI ідентифікувати і категоризувати трафік на основі його вмісту, а не лише за основними заголовками протоколів.

Реальні приклади впровадження DPI

— **Fortinet:** Використовує DPI у своїх рішеннях FortiGate для аналізу мережевого трафіку в реальному часі. Це дозволяє забезпечувати не лише балансування навантаження, але й захист від загроз на глибокому рівні, блокуючи шкідливі пакети до їх проникнення в мережу.

— **Cisco:** Використовує DPI у своїй архітектурі Cisco ACI (Application Centric Infrastructure). DPI допомагає контролювати додатки на рівні їх вмісту, а не тільки заголовків, забезпечуючи кращу оптимізацію використання ресурсів.

Інтелектуальне балансування з використанням DPI

Однією з головних переваг DPI є можливість адаптивного управління ресурсами та інтелектуального балансування навантаження. Завдяки доступу до глибоких даних про трафік, DPI дозволяє розподіляти ресурси на основі типу трафіку або додатка. Наприклад, при високому навантаженні мережі, система DPI може перенаправляти некритичний трафік через менш завантажені канали, залишаючи більше ресурсів для критичних додатків, таких як відеоконференції або транзакції.

Іншим важливим аспектом є можливість протоколу OpenFlow, який підтримує динамічну маршрутизацію на основі аналізу трафіку. DPI може інтегруватися з OpenFlow для забезпечення точного управління трафіком, що підвищує ефективність використання мережевих ресурсів і знижує затримки.

Переваги та недоліки DPI

Переваги:

— Глибокий контроль над трафіком: DPI надає детальну інформацію про кожен пакет, що дозволяє управляти трафіком на глибшому рівні.

— Поліпшення безпеки: Можливість виявляти загрози всередині пакетів (наприклад, шкідливе програмне забезпечення або атаки на рівні прикладних протоколів).
— Оптимізація трафіку: Інтелектуальні рішення щодо маршрутизації, що дозволяє уникнути перевантажень і покращити якість обслуговування (QoS).
— Зниження затримок: Завдяки пріоритизації критичного трафіку та ефективному розподілу ресурсів.

Недоліки:

— Висока вартість: Впровадження DPI вимагає значних інвестицій у мережеві пристрої та програмне забезпечення.
— Затримки в обробці: Незважаючи на ефективність DPI, процес глибокої інспекції пакетів може впливати на продуктивність мережі, особливо в умовах високого навантаження.
— Проблеми з конфіденційністю: Глибока інспекція трафіку може викликати занепокоєння щодо конфіденційності користувачів, особливо у випадку доступу до приватних даних.

2.2.4 Алгоритми машинного навчання для балансування навантаження

Історія та розвиток технології

Балансування навантаження є однією з ключових проблем для сучасних мережевих і хмарних інфраструктур, оскільки зростаюча кількість користувачів і пристроїв вимагає від мережевих систем обробки величезної кількості даних і запитів. Традиційні методи балансування навантаження, такі як алгоритми Round Robin або Least Connections, зіткнулися з обмеженнями в умовах динамічних і масштабованих систем. Це стимулювало розвиток більш інтелектуальних методів, таких як алгоритми на основі машинного навчання (ML), здатних передбачати навантаження і адаптувати ресурси у реальному часі. Перші спроби інтеграції ML для балансування навантаження були направлені на оптимізацію обчислювальних

ресурсів у хмарних середовищах і підвищення продуктивності мережевої інфраструктури.

Принципи роботи алгоритмів машинного навчання для балансування навантаження

Основною перевагою ML-алгоритмів є здатність адаптуватися до мінливих умов мережі та прогнозувати поведінку трафіку. Алгоритми ML здатні аналізувати великі обсяги даних у реальному часі, враховуючи метрики, такі як пропускна здатність, затримка, завантаженість серверів і час відгуку. Найпоширенішими методами, що використовуються для балансування навантаження, є:

— Регресія та рішення на основі дерев (Random Forest, Decision Trees): ці методи допомагають передбачати навантаження на основі історичних даних і показників, таких як кількість користувачів та обсяг трафіку. Вони визначають, які ресурси будуть оптимально використані для обслуговування поточних запитів.

— Нейронні мережі та глибоке навчання: завдяки здатності виявляти приховані закономірності в даних, нейронні мережі дозволяють оптимізувати балансування на основі різноманітних факторів у складних мережах. Наприклад, у хмарних середовищах це може допомогти передбачати пікові навантаження та автоматично розподіляти ресурси між серверами або навіть хмарними провайдерами.

— Підкріплювальне навчання: цей підхід використовується для динамічного прийняття рішень на основі винагороди та покарань. Він дозволяє навчити агента, який самостійно оптимізує балансування на основі даних про поточний стан мережі.

У мережах SDN (програмно-визначених мережах) підхід на основі машинного навчання може значно підвищити ефективність балансування, забезпечуючи швидку адаптацію до змін у трафіку або доступності серверів. Наприклад, Google Cloud

використовує глибокі нейронні мережі для передбачення завантаженості хмарних серверів і оптимізації розподілу трафіку між різними регіонами та дата-центрами.

Труднощі та виклики

Попри значні переваги використання ML для балансування навантаження, існує ряд викликів, пов'язаних із впровадженням таких рішень:

— Невизначеність і мінливість середовища: Мережеве середовище, особливо в хмарних обчисленнях і гетерогенних мережах, змінюється надзвичайно швидко. Алгоритми повинні бути достатньо гнучкими для того, щоб адаптуватися до постійних змін, але водночас уникати надмірного навантаження на обчислювальні ресурси.

— Велика кількість параметрів: Алгоритми машинного навчання повинні враховувати безліч факторів: пропускну здатність, час затримки, показники продуктивності серверів, типи трафіку тощо. Це може ускладнити процес навчання та підвищити ризик неправильного прийняття рішень, що може призвести до перевантаження серверів або значного збільшення затримки.

— Безпека та конфіденційність: Балансування навантаження з використанням ML вимагає постійного моніторингу та аналізу трафіку. Це може створити потенційні загрози для безпеки та конфіденційності даних, особливо коли йдеться про глибоке інспектування трафіку.

Реальні приклади та рішення:

— **Google Cloud:** Використовує інтелектуальні алгоритми для адаптивного управління трафіком. За допомогою ML-алгоритмів Google Cloud може прогнозувати попит на ресурси і відповідно адаптувати свою інфраструктуру для підтримки стабільної продуктивності.

— **AWS Elastic Load Balancing (ELB):** Впроваджує машинне навчання для керування своїми сервісами Route 53, що відповідає за маршрутизацію трафіку

на основі реальних умов завантаженості серверів. Це допомагає забезпечувати ефективний розподіл трафіку на рівні глобальних мережевих операцій.

— **Microsoft Azure:** Також пропонує ML-рішення для автоматичного балансування навантаження в хмарних середовищах, що дозволяє зменшити витрати на інфраструктуру та підвищити ефективність використання ресурсів.

Open-source рішення:

— **Kubernetes:** Платформа для управління контейнеризованими додатками, яка використовує машинне навчання для автоматичного масштабування ресурсів та балансування навантаження між вузлами.

— **TensorFlow Serving:** Фреймворк від Google для розгортання моделей машинного навчання у виробничих середовищах, що інтегрується з різними системами для розподілу навантаження на основі аналізу трафіку.

— **ONNX (Open Neural Network Exchange):** Відкрита екосистема для обміну моделями нейронних мереж, яка дозволяє інтегрувати інтелектуальне балансування навантаження у додатки на основі штучного інтелекту.

Прориви в технології балансування навантаження на основі ML

Нещодавні прориви в технології балансування навантаження на основі ML включають використання підкріплювального навчання для самонавчальних агентів, здатних динамічно приймати рішення в режимі реального часу. Алгоритми Reinforcement Learning, зокрема DQN (Deep Q-Networks), дозволяють агентам навчатися на основі проб і помилок, коригуючи свої дії для досягнення оптимального результату. Ці підходи застосовуються в SD-WAN рішеннях для забезпечення стабільності та продуктивності мереж навіть за мінливих умов.

2.2.5 Високодоступне балансування навантаження у хмарних інфраструктурах

Нещодавні прориви в технології балансування навантаження на основі ML включають використання підкріплювального навчання для самонавчальних агентів, здатних динамічно приймати рішення в режимі реального часу. Алгоритми

Reinforcement Learning, зокрема DQN (Deep Q-Networks), дозволяють агентам навчатися на основі проб і помилок, коригуючи свої дії для досягнення оптимального результату. Ці підходи застосовуються в SD-WAN рішеннях для забезпечення стабільності та продуктивності мереж навіть за мінливих умов.

Принципи балансування навантаження для високої доступності

Основна задача балансування навантаження у хмарних інфраструктурах — це рівномірний розподіл трафіку між наявними ресурсами, такими як сервери, віртуальні машини або контейнери, з урахуванням їхнього стану та завантаженості. Це дозволяє уникнути перевантаження окремих вузлів і забезпечити безперервність надання послуг. До ключових принципів балансування навантаження належать:

— Висока доступність **(High Availability)**: У хмарних інфраструктурах доступність сервісів визначається не лише потужністю окремих серверів, а й можливістю балансування запитів між кількома дата-центрами або навіть регіонами. Це забезпечує стійкість до відмов та мінімізує ризик простоїв під час аварій.

— Масштабованість **(Scalability)**: Хмарні провайдери використовують автоматизовані механізми масштабування для забезпечення безперебійної роботи під час пікових навантажень. Наприклад, під час різкого збільшення кількості запитів, балансувальники можуть динамічно додавати нові ресурси, такі як додаткові сервери або контейнери, для обробки додаткового навантаження.

— стійкість до збоїв **(Fault Tolerance)**: Балансувальники можуть автоматично виявляти несправні або перевантажені ресурси і перенаправляти трафік на інші доступні вузли. Це забезпечує стійкість до відмов окремих серверів або навіть цілих дата-центрів без негативного впливу на користувацький досвід.

Технології та механізми високодоступного балансування

Хмарні провайдери використовують різноманітні технології для реалізації високодоступного балансування навантаження:

— DNS-based Load Balancing (балансування на основі DNS): Цей механізм використовує систему доменних імен (DNS) для розподілу запитів між різними дата-центрами або серверами. Користувацькі запити автоматично перенаправляються на найближчі або найменш завантажені ресурси на основі географічного місцезнаходження або стану серверів.

— Anycast Routing: Технологія, що дозволяє декільком серверам у різних географічних місцях відповідати на один і той самий IP-адресний запит. Це значно зменшує затримки і забезпечує рівномірний розподіл навантаження, особливо для глобальних сервісів.

— Автоматичне масштабування (Auto-scaling): У хмарних інфраструктурах, таких як Amazon Web Services (AWS) та Microsoft Azure, автоматичне масштабування є важливою складовою високодоступного балансування навантаження. Auto-scaling дозволяє автоматично збільшувати або зменшувати кількість ресурсів (віртуальних машин, контейнерів тощо) на основі поточного навантаження, що допомагає уникнути простоїв під час пікових навантажень або економити ресурси під час низької активності.

— Health Checks (перевірка стану): Для забезпечення надійного перенаправлення трафіку балансувальники навантаження виконують регулярні перевірки стану серверів та інших компонентів системи. Якщо ресурс не відповідає або його продуктивність знижується, балансувальник автоматично виключає його з потоку запитів, поки проблема не буде усунена.

— Geo-redundancy (географічне резервування): Використання декількох дата-центрів у різних регіонах дозволяє хмарним провайдерам забезпечувати стійкість до відмов у випадку природних катастроф або технічних збоїв. Якщо один регіон стає недоступним, трафік автоматично перенаправляється на інші доступні центри обробки даних.

Приклади з реальних рішень

Microsoft Azure Load Balancer: Це хмарний сервіс, що забезпечує балансування навантаження на рівнях 4 (L4) та 7 (L7) мережевої моделі OSI. Azure використовує механізми автоматичного масштабування для обслуговування як внутрішніх, так і зовнішніх запитів. Один із важливих аспектів Azure Load Balancer — підтримка failover між різними регіонами та дата-центрами, що забезпечує стійкість до регіональних збоїв. Крім того, Azure Traffic Manager дозволяє балансувати трафік між кількома регіонами, використовуючи перевірки стану, що забезпечує глобальну високо доступність сервісів.

AWS Elastic Load Balancing (ELB): Це масштабована і гнучка система балансування навантаження, яка підтримує кілька типів балансувальників: Classic Load Balancer, Application Load Balancer (для L7), Network Load Balancer (для L4) і Gateway Load Balancer. AWS ELB використовує механізми перевірки стану та автоматичного перенаправлення трафіку у випадку збоїв на окремих вузлах або серверах. ELB також тісно інтегрується з іншими сервісами AWS, такими як Auto Scaling, для динамічного додавання ресурсів під час пікових навантажень.

Переваги та виклики високодоступного балансування

Переваги:

— Підвищена надійність: Використання географічно розподілених дата-центрів та автоматичних механізмів масштабування зменшує ймовірність простоїв і забезпечує стабільну роботу навіть під час аварій.

— Масштабованість: Хмарні рішення дозволяють компаніям швидко реагувати на зміни у навантаженні, автоматично додаючи ресурси для обробки зростаючого трафіку, що особливо важливо для додатків, таких як електронна комерція або потокові сервіси.

— Економічна ефективність: Системи балансування навантаження в хмарі дозволяють оптимізувати використання ресурсів, зменшуючи витрати на інфраструктуру в умовах мінливого навантаження.

Недоліки:

— Складність впровадження: Високо доступне балансування вимагає ретельного налаштування та інтеграції з іншими системами, що може бути складним для організацій з обмеженими технічними ресурсами.
— Затримки та продуктивність: Незважаючи на всі переваги, балансувальники навантаження можуть додавати невеликі затримки у процес обробки запитів, особливо у глобальних мережах. Це потребує оптимізації алгоритмів для зменшення цих затримок.

2.2.6 Масштабованість мереж та використання хмарних технологій

Масштабованість мережевої інфраструктури є одним із ключових факторів успішної роботи сучасних цифрових систем. Зі зростанням кількості користувачів, обсягу трафіку та складності додатків, мережі повинні мати можливість ефективно адаптуватися до змінних умов, забезпечуючи стабільну продуктивність та високу доступність сервісів. Хмарні технології відіграють вирішальну роль у забезпеченні цієї масштабованості, надаючи гнучкі, потужні та економічно ефективні рішення для управління мережевими ресурсами.

Поняття масштабованості мережі

Масштабованість мережі визначається здатністю мережевої інфраструктури розширювати свої можливості для обробки збільшеного навантаження без значних змін у її архітектурі або додаткових витрат. Існують два основні типи масштабованості:

— Вертикальна масштабованість (Scale-Up): Підвищення потужності окремих мережевих пристроїв шляхом додавання більш потужних процесорів, оперативної пам'яті або збільшення пропускної здатності [19].
— Горизонтальна масштабованість (Scale-Out): Додавання нових мережевих пристроїв або серверів до існуючої інфраструктури для розподілу навантаження та підвищення загальної пропускної здатності мережі [19].

Горизонтальна масштабованість зазвичай вважається більш ефективною та гнучкою, оскільки дозволяє легко адаптувати мережу до змінних умов та зростаючих потреб без значних перебоїв у роботі.

Використання хмарних технологій для масштабованості

Хмарні технології забезпечують необхідну гнучкість та адаптивність для масштабування мережевих інфраструктур. Вони дозволяють організаціям швидко реагувати на зміни у навантаженні, зменшуючи необхідність у великих капітальних інвестиціях у фізичну інфраструктуру. Основні компоненти хмарних технологій, що сприяють масштабованості, включають:

— Віртуалізація: Технологія, яка дозволяє створювати віртуальні екземпляри серверів, зберігання даних та мережевих ресурсів на фізичних пристроях. Це забезпечує гнучкість у розподілі ресурсів та ефективне використання апаратних засобів.

— Контейнеризація: Використання контейнерів (наприклад, Docker) дозволяє запускати додатки ізольовано від основної системи, що спрощує їх розгортання, масштабування та управління.

— Оркестрація контейнерів: Інструменти, такі як Kubernetes, забезпечують автоматизоване розгортання, масштабування та управління контейнеризованими додатками, що значно полегшує процес масштабування мережевих сервісів.

— Автоматичне масштабування (Auto-Scaling): Механізми, які автоматично додають або видаляють ресурси (сервери, контейнери) відповідно до поточного навантаження, забезпечуючи оптимальну продуктивність та економію витрат.

Хмарні платформи та їх можливості для масштабованості

Провідні хмарні провайдери, такі як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP), пропонують широкий спектр сервісів, що

сприяють масштабованості мережевих інфраструктур. Розглянемо детальніше функціонал та технології цих платформ:

Amazon Web Services (AWS)

AWS є однією з найпопулярніших та найрозвиненіших хмарних платформ, що пропонує різноманітні сервіси для забезпечення масштабованості [20]:

— Elastic Compute Cloud (EC2): Дозволяє запускати та масштабувати віртуальні машини відповідно до потреб. EC2 пропонує різні типи інстансів, оптимізованих для різних завдань, таких як обчислювальні, пам'яті-інтенсивні або сховищні операції.

— Auto Scaling Groups (ASG): Автоматично масштабують кількість EC2-інстансів на основі заданих правил та показників продуктивності. Це забезпечує підтримку стабільної продуктивності додатків навіть під час пікових навантажень.

— Elastic Load Balancing (ELB): Забезпечує автоматичне розподілення трафіку між кількома серверами, підвищуючи доступність та надійність додатків. ELB підтримує різні типи балансувальників, включаючи Application Load Balancer (ALB) для рівня 7 та Network Load Balancer (NLB) для рівня 4.

— Amazon Virtual Private Cloud (VPC): Надає можливість створювати ізольовані мережеві середовища з гнучким управлінням мережевими ресурсами. VPC дозволяє налаштовувати підмережі, маршрутизацію, мережеві ACL та безпеку.

— AWS Lambda: Функції без серверів, які дозволяють запускати код у відповідь на події без необхідності керувати серверною інфраструктурою. Lambda автоматично масштабує ресурси відповідно до кількості запитів.

— Amazon S3 (Simple Storage Service): Масштабоване сховище об'єктів для зберігання великих обсягів даних з високою доступністю та надійністю. S3 дозволяє зберігати та отримувати дані з будь-якої точки світу з мінімальними затримками.

— Amazon RDS (Relational Database Service) та DynamoDB: Масштабовані рішення для управління реляційними та NoSQL базами даних відповідно. RDS підтримує автоматичне масштабування зберігання та обчислювальних ресурсів, тоді як DynamoDB забезпечує безперервну масштабованість та низьку латентність.

Microsoft Azure

Azure пропонує комплексні сервіси для забезпечення масштабованості мережевих інфраструктур, орієнтуючись на інтеграцію з існуючими системами та підтримку різних типів робочих навантажень:

— Azure Virtual Machines (VM): Дозволяє створювати та керувати віртуальними машинами різних конфігурацій, оптимізованих для різних завдань. Azure VM підтримує автоматичне масштабування за допомогою Scale Sets.

— Azure Scale Sets: Дозволяє автоматично масштабувати кількість віртуальних машин відповідно до поточного навантаження. Scale Sets інтегруються з Azure Load Balancer та Azure Application Gateway для забезпечення балансування навантаження.

— Azure Load Balancer: Пропонує балансування навантаження на рівнях 4 (TCP/UDP) та 7 (HTTP/HTTPS), забезпечуючи високу доступність додатків. Azure Load Balancer підтримує автоматичне масштабування та інтегрується з іншими сервісами Azure для динамічного управління трафіком.

— Azure Kubernetes Service (AKS): Платформа для управління контейнеризованими додатками, яка забезпечує автоматизоване розгортання, масштабування та управління Kubernetes-кластерами. AKS інтегрується з Azure Monitor та Azure Policy для забезпечення моніторингу та дотримання політик безпеки.

— Azure Functions: Сервіс без серверів, що дозволяє запускати функції у відповідь на події. Azure Functions автоматично масштабує ресурси відповідно

до кількості викликів, забезпечуючи високий рівень гнучкості та економії витрат.

— Azure Blob Storage: Масштабоване сховище об'єктів для зберігання великих обсягів неструктурованих даних. Blob Storage підтримує автоматичне масштабування зберігання та забезпечує високу доступність даних через географічну реплікацію.

— Azure Cosmos DB: Глобально розподілена NoSQL база даних, що забезпечує безперервну масштабованість та низьку латентність

Google Cloud Platform (GCP)

Google Cloud Platform надає широкий спектр сервісів, які сприяють масштабованості та гнучкості мережевих інфраструктур:

— Google Compute Engine (GCE): Дозволяє створювати та керувати віртуальними машинами з високою гнучкістю та масштабованістю. GCE підтримує автоматичне масштабування груп інстансів за допомогою Managed Instance Groups.

— Google Kubernetes Engine (GKE): Платформа для управління контейнеризованими додатками, яка забезпечує автоматизоване розгортання, масштабування та управління Kubernetes-кластерами. GKE інтегрується з іншими сервісами Google Cloud для забезпечення моніторингу, логування та безпеки.

— Google Cloud Load Balancing: Забезпечує глобальне балансування навантаження з високою пропускною здатністю та низькими затримками. Cloud Load Balancing підтримує балансування на рівнях 4 (TCP/UDP) та 7 (HTTP/HTTPS), а також інтегрується з Cloud CDN для оптимізації доставки контенту.

— Google Cloud Functions: Сервіс без серверів, що дозволяє запускати функції у відповідь на події. Cloud Functions автоматично масштабує ресурси відповідно до кількості викликів, забезпечуючи високу гнучкість та економію витрат.

— Google Cloud Storage: Масштабоване сховище об'єктів для зберігання великих обсягів даних з високою доступністю та надійністю. Cloud Storage підтримує різні класи зберігання, що дозволяє оптимізувати витрати відповідно до потреб додатків.

— Google Cloud Spanner: Глобально розподілена реляційна база даних, яка забезпечує безперервну масштабованість та високу доступність. Cloud Spanner поєднує гнучкість NoSQL з консистентністю SQL-баз даних, що робить його ідеальним для масштабованих додатків з високими вимогами до транзакцій.

— Google BigQuery: Масштабована аналітична база даних для обробки великих обсягів даних. BigQuery дозволяє виконувати високопродуктивні SQL-запити та інтегрується з іншими сервісами Google Cloud для забезпечення ефективної обробки та аналізу даних.

Технології, що сприяють масштабованості

Крім основних хмарних сервісів, існують додаткові технології, які значно підвищують масштабованість мережевих інфраструктур:

— Мікросервісна архітектура: Розбиття додатків на невеликі, незалежні сервіси, що дозволяє масштабувати окремі компоненти додатка без впливу на інші частини системи. Це також сприяє більш ефективному управлінню життєвим циклом додатків та їхніх компонентів.

— Service Mesh: Технології, такі як Istio, забезпечують управління мережевим трафіком між мікросервісами, забезпечуючи високий рівень контролю, безпеки та надійності. Service Mesh дозволяє здійснювати деталізований моніторинг, управління трафіком та забезпечення безпеки на рівні мережевих зв'язків між сервісами.

— Edge Computing: Розміщення обчислювальних ресурсів ближче до кінцевих користувачів дозволяє знизити затримки та зменшити навантаження на центральні сервери, що сприяє більш ефективному масштабуванню мережі.

Edge Computing особливо важливий для додатків реального часу, таких як IoT, відео-потоки та онлайн-ігри.

— Network Functions Virtualization (NFV): Віртуалізація мережевих функцій, таких як маршрутизатори, брандмауери та балансувальники навантаження, що дозволяє динамічно розгортати та масштабувати мережеві сервіси. NFV сприяє гнучкості мережевої інфраструктури та знижує витрати на фізичне обладнання.

Переваги використання хмарних технологій для масштабованості

Використання хмарних технологій для забезпечення масштабованості мережевих інфраструктур має низку суттєвих переваг:

Гнучкість та адаптивність: Можливість швидко масштабувати ресурси відповідно до змінних вимог дозволяє ефективно реагувати на пікові навантаження та зменшувати витрати під час періодів низької активності.

Висока доступність та надійність: Хмарні провайдери забезпечують резервування ресурсів та географічну розподіленість, що мінімізує ризик простоїв та забезпечує безперебійну роботу додатків навіть у випадку аварій.

Оптимізація витрат: Модель оплати за використання дозволяє компаніям ефективно управляти витратами, платячи лише за ті ресурси, які дійсно використовуються. Це особливо важливо для стартапів та малих підприємств, які не мають великих капіталовкладень у фізичну інфраструктуру.

Автоматизація: Інтеграція з інструментами автоматичного масштабування та управління ресурсами дозволяє зменшити ручну роботу та підвищити ефективність управління мережею.

Безпека: Хмарні провайдери пропонують розширені механізми безпеки, включаючи шифрування даних, управління доступом та моніторинг безпеки, що забезпечує захист мережевих ресурсів від несанкціонованого доступу та атак.

Виклики та обмеження

Попри численні переваги, використання хмарних технологій для масштабованості мережевих інфраструктур супроводжується певними викликами та обмеженнями:

Залежність від провайдера: Використання хмарних сервісів може призвести до високої залежності від одного або кількох провайдерів, що створює ризики у випадку їхніх збоїв або змін у політиках.

Безпека та конфіденційність: Хоча хмарні провайдери пропонують потужні механізми безпеки, передача даних до хмари може викликати занепокоєння щодо їх конфіденційності та захисту від несанкціонованого доступу. Організаціям необхідно ретельно оцінювати політики безпеки та відповідність вимогам щодо конфіденційності.

Виклики інтеграції: Інтеграція хмарних сервісів з існуючою інфраструктурою може бути складною, особливо у випадках гібридних або мульти-хмарних середовищ. Це вимагає спеціалізованих знань та досвіду у сфері управління хмарними ресурсами.

Затримки: Хоча хмарні сервіси забезпечують високу пропускну здатність, віддаленість дата-центрів може спричиняти додаткові затримки, особливо для додатків, що потребують низької латентності, таких як фінансові системи чи додатки реального часу.

Управління витратами: Хоча модель оплати за використання дозволяє оптимізувати витрати, неконтрольоване масштабування може призвести до значних витрат. Організаціям необхідно впроваджувати стратегії моніторингу та управління витратами для уникнення перевитрат.

Практичні реалізації та кейси

Розглянемо декілька реальних прикладів впровадження масштабованості мережевих інфраструктур за допомогою хмарних технологій:

Google Cloud

Google Kubernetes Engine (GKE): GKE дозволяє автоматизувати розгортання, масштабування та управління контейнеризованими додатками. Використовуючи GKE, організації можуть легко масштабувати свої додатки в залежності від навантаження, забезпечуючи високу доступність та продуктивність. GKE інтегрується з іншими сервісами Google Cloud, такими як Cloud Monitoring та Cloud Logging, що дозволяє ефективно моніторити стан кластерів та додатків [21].

Google Cloud Load Balancing: Забезпечує глобальне балансування навантаження з низькими затримками та високою пропускною здатністю. Це дозволяє розподіляти трафік між кількома дата-центрами по всьому світу, забезпечуючи високу доступність та оптимізацію використання ресурсів.

Amazon Web Services (AWS)

Amazon Elastic Kubernetes Service (EKS): EKS спрощує управління Kubernetes-кластерами, забезпечуючи автоматичне масштабування та інтеграцію з іншими сервісами AWS, такими як Amazon RDS та Amazon S3. EKS дозволяє організаціям швидко реагувати на зміни навантаження та забезпечувати високу доступність додатків [22].

AWS Global Accelerator: Цей сервіс покращує доступність та продуктивність додатків за рахунок використання мережі глобальних точок присутності AWS. Global Accelerator оптимізує маршрутизацію трафіку до найближчих та найменш завантажених ресурсів, знижуючи затримки та підвищуючи загальну продуктивність.

Microsoft Azure

Azure Kubernetes Service (AKS): AKS надає повністю керований Kubernetes, що дозволяє організаціям легко розгортати, масштабувати та управляти контейнеризованими додатками. AKS інтегрується з Azure Monitor та Azure Policy, що забезпечує ефективний моніторинг та дотримання політик безпеки.

Azure Front Door: Це глобальний балансувальник навантаження та служба оптимізації доставки контенту, що забезпечує високу доступність та швидкість

доставки веб-додатків. Front Door використовує маршрутизацію на основі найближчого географічного розташування та оптимізує маршрутизацію трафіку для зниження затримок.

Інші приклади

Spotify: Використовує Google Kubernetes Engine для управління своїми мікросервісами, забезпечуючи високий рівень масштабованості та гнучкості. Spotify інтегрує GKE з іншими сервісами Google Cloud для оптимізації розподілу трафіку та забезпечення безперебійної роботи своїх стрімінгових сервісів [23].

Netflix: Використовує Amazon Web Services для забезпечення масштабованості своїх потокових сервісів. Netflix активно використовує Auto Scaling Groups та Elastic Load Balancing для динамічного масштабування своїх серверів відповідно до навантаження, забезпечуючи високу доступність та стабільну продуктивність.

3. ПРАКТИЧНА ЧАСТИНА

Цей розділ присвячений практичному дослідженню можливостей масштабованості мережевих інфраструктур з використанням сучасних хмарних платформ. Основна увага приділяється аналізу та тестуванню таких провідних хмарних сервісів, як Amazon Web Services (AWS), Cloudflare та інших. Враховуючи обмеженість ресурсів для локального тестування, практична частина зосереджена виключно на хмарних рішеннях, доповнених теоретичним описом традиційних локальних методів масштабування.

Метою цього розділу є не лише налаштування та тестування хмарних сервісів для оцінки їх ефективності у забезпеченні масштабованості, але й інтеграція власного домену, зареєстрованого через GoDaddy, для демонстрації реальних кейсів застосування хмарних рішень. Практична частина включає створення тестового середовища на обраних хмарних платформах, налаштування балансувальників навантаження, генерацію навантаження для тестування продуктивності, збір та аналіз отриманих даних, а також формулювання висновків та рекомендацій щодо вибору оптимальних хмарних сервісів для різних сценаріїв використання.

Крім практичних завдань, розділ містить теоретичний огляд локальних рішень для масштабованості мереж, що дозволяє порівняти традиційні методи з сучасними хмарними підходами. Це забезпечує всебічне розуміння переваг та недоліків різних підходів до масштабування, сприяючи обґрунтованому вибору найбільш відповідних рішень для конкретних потреб організацій.

3.1 Мета практичної роботи

Основною метою цієї практичної роботи є аналіз та оптимізація мережевої інфраструктури для підтримки високих навантажень за допомогою хмарних платформ і сучасних технологій управління трафіком. Практична частина роботи спрямована на тестування і порівняння ефективності хмарних сервісів, таких як Amazon Web Services (AWS) та Cloudflare, у контексті масштабованості, управління трафіком і балансування навантаження.

Дослідження включає налаштування Load Balancer для розподілу трафіку, використання CDN для зниження затримок і оптимізації доступу до ресурсів, а також аналіз впливу таких рішень на час відповіді, продуктивність і стабільність роботи мережі. Крім того, робота передбачає інтеграцію домену, зареєстрованого через GoDaddy, для моделювання реальних сценаріїв використання хмарних технологій.

Основна увага приділяється порівнянню різних архітектур мереж: односерверної (Single Server Architecture) та з балансуванням навантаження (Load Balancing Architecture). Тестування проводиться з використанням інструментів Apache JMeter для оцінки продуктивності та визначення ефективності впроваджених рішень. Це дозволяє зробити висновки про оптимальні методи побудови мережевої інфраструктури, здатної забезпечувати високу продуктивність та надійність у складних умовах.

3.2 Налаштування тестового середовища на хмарних платформах

Для ефективного проведення тестування масштабованості на обраних хмарних платформах необхідно виконати кілька підготовчих кроків. Ці кроки включають налаштування тестових облікових записів, забезпечення безпеки доступу через системи управління ідентифікацією та доступом (IAM), а також встановлення та налаштування необхідних інструментів командного рядка (CLI) для управління хмарними сервісами.

3.2.1 Створення та налаштування тестових облікових записів

Для проведення тестування буде використано тестові облікові записи на кожній з обраних хмарних платформ: Amazon Web Services (AWS) та Cloudflare. Ці акаунти мають безкоштовний баланс або пробний період, що дозволяє проводити необхідні експерименти без значних витрат.

— **AWS**: Використовується AWS Free Tier, який надає обмежений доступ до основних сервісів протягом 12 місяців.

— **Cloudflare**: Безкоштовний обліковий запис, який надає доступ до базових функцій Cloudflare безкоштовно.

3.2.2 Встановлення необхідних інструментів

Для ефективного управління хмарними сервісами через командний рядок було встановлено відповідні інструменти CLI та SDK. Ці інструменти забезпечують автоматизацію процесів розгортання, управління та моніторингу ресурсів, що є ключовим для проведення масштабованих тестів.

AWS CLI:

Для встановлення AWS CLI на операційну систему Windows необхідно виконати ряд дій, зокрема скористатися Windows Terminal для запуску відповідної команди. Спочатку відкриваємо Windows Terminal. Та виконуємо команду **msiexec.exe /i https://awcli.amazonaws.com/AWSCLIV2.msi /qn.** Дана команда

завантажує і встановлює пакет AWS CLI версії 2 у безшумному режимі (без взаємодії з користувачем, що позначається параметром /qn). Після запуску цієї команди автоматично розпочнеться процес встановлення програмного забезпечення, який виконається у фоновому режимі, не потребуючи додаткових дій зі сторони користувача.

Рисунок 3.1 - Виконання команди msiexec.exe /i https://awcli.amazonaws.com/AWSCLIV2.msi /qn

Для перевірки коректності встановлення AWS CLI слід виконати команду:

aws --version

Ця команда відображає інформацію про версію встановленого програмного забезпечення, що дозволяє впевнитись у його успішному інсталяційному процесі.

Рисунок 3.2 - Виконання команди aws --version

Для початку налаштування підключення до AWS через CLI необхідно виконати команду **aws configure**.

Після цього система запитає кілька параметрів, які слід ввести для коректного підключення до облікового запису AWS. Розглянемо кожен з цих параметрів:

— AWS Access Key ID [None]: Це ідентифікатор ключа доступу, який використовується для зовнішнього керування ресурсами AWS. Його можна згенерувати на сторінці керування обліковими даними за посиланням: https://console.aws.amazon.com/iam/home#/security_credentials. Даний ключ є необхідним для автентифікації запитів до AWS сервісів через CLI.

— AWS Secret Access Key [None]: Це секретна частина ключа доступу, яка генерується разом із Access Key ID. Важливо зберігати цей ключ у безпечному

місці, оскільки він не відображається повторно після генерації. Його також можна отримати через консоль керування обліковими даними AWS.

— Default region name [None]: Цей параметр вказує на регіон, у якому розміщені ваші ресурси AWS. Регіон є частиною URL-адреси, коли ви заходите до свого облікового запису через веб-інтерфейс AWS (наприклад, us-west-2 або eu-central-1). Вказання правильного регіону забезпечує доступ до ваших ресурсів у конкретному географічному розташуванні.

— Default output format [None]: Формат виводу даних, який буде використовуватися за замовчуванням під час виконання команд AWS CLI. Найбільш поширеним і зручним форматом є JSON, проте підтримуються також інші формати, такі як YAML, текстовий або табличний формат.

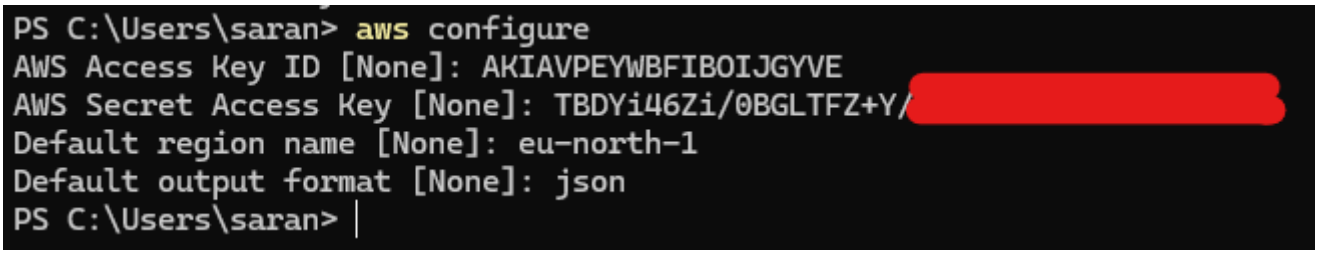


Рисунок 3.3 - Виконання команди aws configure та приклад налаштувань

Для перевірки того, чи прийняті ключі доступу і правильність налаштувань підключення, слід виконати команду **aws sts get-caller-identity**.

Ця команда надсилає запит до сервісу AWS Security Token Service (STS) і повертає інформацію про поточного користувача або роль, яку представляєте. У відповідь отримаємо такі дані:

— UserId: Унікальний ідентифікатор користувача або ролі в AWS.

— Account: Ідентифікатор облікового запису AWS, до якого ви підключені.

— ARN: Amazon Resource Name, який визначає вашу особу або роль у AWS.

Отримання цих даних підтверджує, що ключ доступу та секретний ключ коректно налаштовані, а ваше підключення до облікового запису AWS успішно встановлено.

3.2.3 Переніс управління DNS доменом до Cloudflare

Також перенесемо тестовий домен на платформу Cloudflare. Для цього виконуємо наступні кроки:

— Увійшовши в панель керування Cloudflare, переходимо до розділу Domain Registration.

— Далі обираємо Transfer Domain, що дозволить нам перенести вже існуючий домен на платформу Cloudflare.

— Натискаємо кнопку Add a Domain, після чого система запитає інформацію про домен, який ми хочемо перенести.

— Після введення необхідних даних і підтвердження, розпочнеться процес переносу домену на Cloudflare.

Цей процес дозволить використовувати можливості Cloudflare для оптимізації трафіку та балансування навантаження, що є важливим етапом у тестуванні функціоналу хмарних рішень для мережевої інфраструктури.

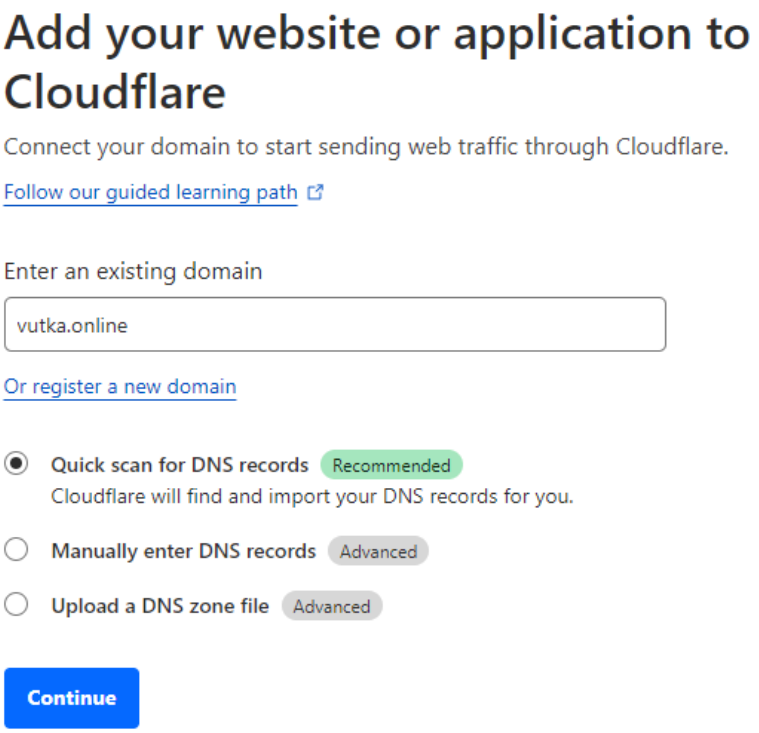


Рисунок 3.4 - Автоматичне перенесення домену до Cloudflare

3.2.4 Розгортання тестового веб-додатку з використанням Nginx та Flask

Для проведення тестування масштабованості необхідно мати веб-сайт або веб-додаток, який слугуватиме об'єктом тестування. Вибір технологій для тестового веб-додатку базується на цілях дослідження та вимогах до навантаження. У рамках даного дослідження вирішено використовувати комбінацію веб-сервера Nginx, веб-фреймворка Flask та сервера додатків Gunicorn. Ця архітектура дозволяє створити реалістичне середовище розгортання та оптимізувати продуктивність додатку.

Етапи розгортання:

— Встановлення Flask: Flask — це легкий веб-фреймворк на основі Python, що дозволяє швидко створювати прості веб-додатки. Він буде використовуватись як бекенд для нашого тестового додатку.

Для встановлення Flask виконуємо такі кроки:

— Встановлюємо pip (якщо ще не встановлений).

```
ubuntu@ip-172-31-19-238:~$ sudo apt install python3-pip
```

Рисунок 3.5 - Встановлення pip

— Створюємо віртуальне середовище і використовуємо команду для встановлення Flask, Nginx та Gunicorn:

```
ubuntu@ip-172-31-19-238:~$ python3 -m venv venv
ubuntu@ip-172-31-19-238:~$ source venv/bin/activate
(venv) ubuntu@ip-172-31-19-238:~$ pip install Flask gunicorn
```

Рисунок 3.6 - Створення вірт. середовища та встановлення Flask Gunicorn

— Створюємо простий тестовий веб-додаток на Flask наступним чином:

```python
from flask import Flask, render_template, request, jsonify, abort
from flask_cors import CORS
import sqlite3
import os

app = Flask(__name__)
CORS(app)

@app.after_request
def add_header(response):
    response.headers["Cache-Control"] =
"no-store, no-cache, must-revalidate, max-age=0"
    response.headers["Pragma"] = "no-cache"
    response.headers["Expires"] = "0"
    return response

# Підключення до бази даних
def get_db_connection():
    conn = sqlite3.connect(db_path)
    conn.row_factory = sqlite3.Row
    return conn

BASE_DIR = os.path.dirname(os.path.abspath(__file__))
db_path = os.path.join(BASE_DIR, 'resources', 'test_db.sqlite')

# Сторінка з HTML-контентом
@app.route('/')
def home():
    return render_template('index.html')

@app.route('/about')
def about():
    return render_template('about.html')
```

Рисунок 3.7 - Код app.py - основний код веб-сайту - частина 1

```
    if category:
        query += " WHERE category = ?"
        params.append(category)

    if sort == 'price_desc':
        query += " ORDER BY price DESC"
    elif sort == 'price_asc':
        query += " ORDER BY price ASC"
    elif sort == 'categories_asc':
        query += " ORDER BY category ASC"
    elif sort == 'categories_desc':
        query += " ORDER BY category DESC"

    query += " LIMIT ? OFFSET ?"
    params.extend([per_page, (page - 1) * per_page])

    products = conn.execute(query, params).fetchall()
    conn.close()
    results = [dict(product) for product in products]
    return jsonify(results), 200

# API для отримання статистики продуктів

@app.route('/api/products/complex_analytics', methods=['GET'])
def complex_analytics():
    conn = get_db_connection()

    # Середня ціна для кожної категорії
    category_avg_query =
"SELECT category, AVG(price) AS avg_price FROM products GROUP BY categor
y"
    category_avg = {row['category']: row['avg_price'] for row in conn.
execute(category_avg_query)}

    # Топ-10 товарів за ціною в кожній категорії
    top_products = {}
    categories = ['electronics', 'clothing', 'home', 'toys', 'books']
    for category in categories:
        query =
"SELECT name, price FROM products WHERE category = ? ORDER BY price DESC
LIMIT 10"
        top_products[category] = [dict(row) for row in conn.execute(
query, (category,))]

    conn.close()
    return jsonify({
        'category_avg_price': category_avg,
        'top_products': top_products
    }), 200


@app.route('/api/products/stats', methods=['GET'])
def product_stats():
    category = request.args.get('category')
    conn = get_db_connection()

    # Обчислення середньої ціни для конкретної категорії
    query_avg_price =
"SELECT AVG(price) AS avg_price FROM products WHERE category = ?"
    avg_price = conn.execute(query_avg_price, (category,)).fetchone()[
'avg_price'] if category else None

    # Отримання кількості продуктів за категоріями
    query_count =
"SELECT category, COUNT(*) as count FROM products GROUP BY category"
    category_counts = conn.execute(query_count).fetchall()
    conn.close()

    return jsonify({
        'average_price': avg_price,
        'category_counts': {row['category']: row['count'] for row in
category_counts}
    }), 200

if __name__ == '__main__':
    app.run()
```

Рисунок 3.8 - Код app.py - основний код веб-сайту - частина 2

Створення systemd-сервісу для Gunicorn

Gunicorn — сервер додатків WSGI, що служить проміжною ланкою між Nginx та Flask. Він ефективно обробляє запити до Flask-додатку, створюючи декілька робочих процесів (воркерів), що дозволяє збільшити кількість одночасно оброблюваних запитів і забезпечує високу продуктивність навіть під навантаженням [24].

Для налаштування автоматичного запуску Flask-додатку через Gunicorn створюємо systemd-сервіс, який забезпечить стабільну роботу додатку як системного процесу. Це дозволить керувати додатком, використовуючи команди systemd, а також забезпечить його автоматичний запуск після перезавантаження сервера.

```
  GNU nano 7.2                                                    /etc/systemd/s
[Unit]
Description=Gunicorn instance to serve my Flask app
After=network.target

[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/myflaskapp
Environment="PATH=/home/ubuntu/myflaskapp/myenv/bin"
ExecStart=/home/ubuntu/myflaskapp/myenv/bin/gunicorn --workers 3 --bind 127.0.0.1:8000 app:app

[Install]
WantedBy=multi-user.target
```

Рисунок 3.9 - Файл конфігурації Gunicorn

Роз'яснення параметрів:

— [Unit]:
  — Description: Описує сервіс — у цьому випадку це Gunicorn-сервіс для нашого Flask-додатку.
  — After=network.target: Гарантує, що сервіс запускається після встановлення мережевого підключення.
— [Service]:
  — User та Group: Вказують користувача та групу, під якими буде виконуватись сервіс. У прикладі встановлено User=ubuntu та

Group=www-data, але вони можуть змінюватись відповідно до вашого середовища.

— WorkingDirectory: Вказує робочу директорію, де знаходиться Flask-додаток.

— Environment: Шлях до віртуального середовища Python, що містить Gunicorn і Flask.

— ExecStart: Команда для запуску Gunicorn. У цьому випадку Gunicorn запускається з трьома робочими процесами (--workers 3) і прив'язується до 127.0.0.1:8000, обробляючи додаток app з файлу app.py.

— [Install]:

— WantedBy=multi-user.target: Вказує, що сервіс повинен запускатись у стандартному багатокористувацькому режимі.

Конфігурація Nginx як зворотного проксі-сервера:

Починаємо з налаштування веб-сервера Nginx для роботи з нашим веб-додатком. Створюємо окремий файл конфігурації, який визначатиме правила проксирування запитів до нашого додатку, забезпечуючи балансування навантаження й оптимізацію обробки трафіку.

Створення та відкриття файлу конфігурації: У директорії /etc/nginx/sites-available/ створюємо новий конфігураційний файл, наприклад, myflaskapp, де будуть зберігатися налаштування для нашого додатку:

```
server {
    listen 80;
    server_name www.vutka.online;

    location / {
        include proxy_params;
        proxy_pass http://unix:/home/username/myflaskapp/myflaskapp.sock;
    }
}
```

Рисунок 3.10 - Файл конфігурації Nginx

Активуємо конфігурацію та перевіряємо коректність налаштувань:

Для того щоб Nginx застосував нову конфігурацію, створюємо символічне посилання на наш конфігураційний файл у директорії sites-enabled. Це дозволяє Nginx використовувати налаштування, визначені для нашого додатку:

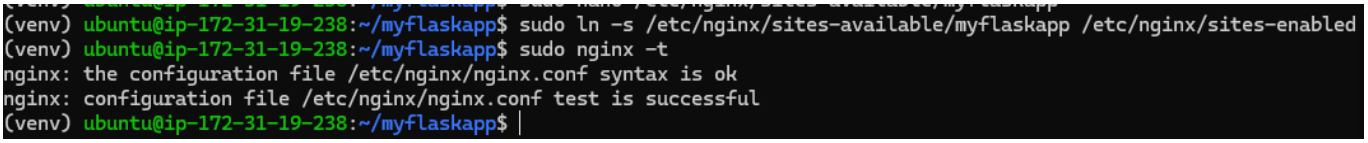


Рисунок 3.11 - Виконання команд для активації конфігурації Nginx

Налаштування SSL/TLS сертифікатів:

Щоб забезпечити безпечне з'єднання для нашого веб-додатку, налаштовуємо SSL/TLS сертифікати, які шифрують передані дані. Для цього використовуємо Certbot — зручний інструмент для автоматичного отримання та оновлення безкоштовних SSL-сертифікатів від Let’s Encrypt [25].

Рисунок 3.12 - Встановлення Certbot

Після встановлення Certbot запускаємо його для отримання сертифікатів. Виконуємо команду, яка автоматично налаштує Nginx на використання SSL

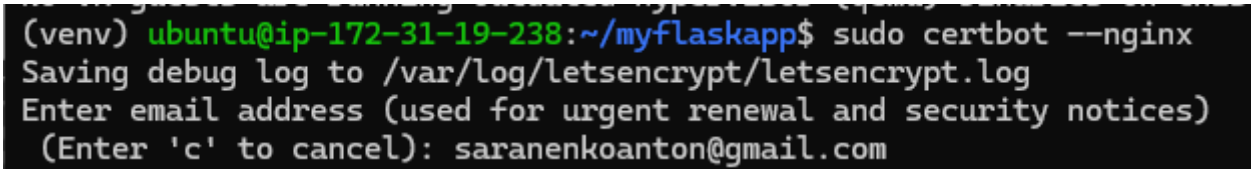


Рисунок 3.13 - Запуск процесу отримання сертифікатів

Certbot запитає наш домен (або IP-адресу) та електронну пошту для повідомлень про оновлення сертифіката. Далі він автоматично додасть необхідні SSL-налаштування в конфігурацію Nginx і створить сертифікат.

```
Which names would you like to activate HTTPS for?
We recommend selecting either all domains, or all domains in a VirtualHost/server block.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1: www.vutka.online
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Select the appropriate numbers separated by commas and/or spaces, or leave input
blank to select all options shown (Enter 'c' to cancel): 1
```

Рисунок 3.14 - Підтвердження отримання сертифікату

Після завершення налаштування Certbot автоматично перезапустить Nginx з оновленими параметрами. Щоб перевірити, чи сертифікати коректно застосовано, переходимо на веб-сайт і перевіряємо наявність захищеного з'єднання

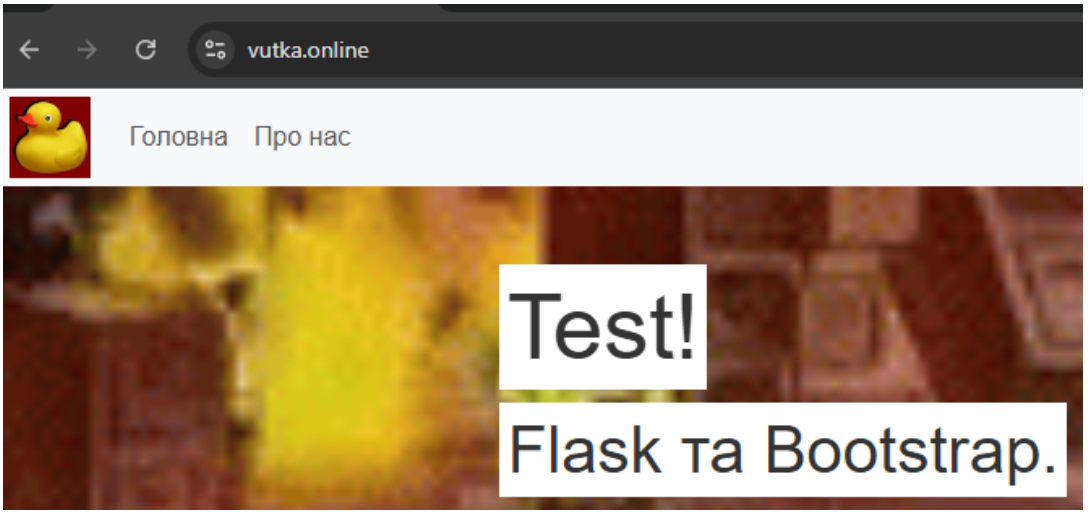


Рисунок 3.15 - Сторінка веб-додатку з активованим SSL/TLS шифруванням

Налаштування та доповнення веб-додатку

Для більш реалістичного моделювання роботи сайту та підвищення навантаження на сервер було розширено функціональність базового веб-додатку, додавши кілька нових елементів і сторінок:

— Створення статичних сторінок:

До додатку було додано дві статичні сторінки:

— Головна сторінка (/) — відображає основну інформацію та базову навігацію.

- Сторінка "Про нас" (/about) — додаткова сторінка, яка демонструє можливість переходу між сторінками та навантажує сервер запитами до статичних ресурсів.

— Додатковий мультимедійний контент:

Для тестування роботи з великими обсягами статичних ресурсів було інтегровано анімації у форматі GIF, а також високоякісні зображення. Це дозволяє змоделювати реальну роботу сайту, що обробляє значний обсяг статичного контенту, та оцінити продуктивність у таких умовах.

— API для роботи з базою даних:

Додано інтерактивні функції через API:

- Фільтрація, сортування та сегментація продуктів: Реалізовано через маршрут /api/products, який дозволяє користувачам отримувати дані з бази даних із зазначеними параметрами. Це моделює реальні кейси, коли сервер обробляє складні запити.
- Статистика продуктів: Додано маршрут /api/products/complex_analytics для отримання розширеної аналітики, наприклад, середньої ціни товарів за категоріями та топ-10 продуктів у кожній категорії.
- Оцінка середніх показників та підрахунків: Через маршрут /api/products/stats можна отримати середню ціну та кількість товарів за категоріями.

Таким чином, додаток відображає роботу з мультимедійним контентом та запитами до бази даних і дозволяє ефективніше перевірити продуктивність і масштабованість налаштувань сервера та балансування навантаження.

3.3 Планування тестів масштабованості

Планування тестів масштабованості є критично важливим етапом у процесі оцінки здатності системи ефективно обробляти збільшене навантаження. Це забезпечує структурований підхід до проведення тестів, дозволяючи отримати достовірні та репрезентативні результати. У цьому підрозділі розглянемо ключові аспекти планування тестів масштабованості, включаючи визначення цілей тестування, вибір відповідних інструментів, встановлення критеріїв успішності та розробку сценаріїв навантаження.

3.3.1 Визначення цілей тестування

Цілі тестування масштабованості спрямовані на зосередження на ключових аспектах продуктивності системи та визначення метрик, які слід контролювати. Відповідно до поставлених завдань, ми прагнемо оцінити, чи забезпечують обрані хмарні сервіси стабільність і високу продуктивність додатку при різних рівнях навантаження. Основні цілі тестування включають:

— Оцінка продуктивності: Визначення середнього та максимального часу відповіді на запити та пропускної здатності системи (кількість запитів на секунду). Ці показники дозволяють виявити, як швидко сервер обробляє запити за нормальних та пікових умов.

— Виявлення вузьких місць: Ідентифікація компонентів, які можуть стати обмежувальними факторами при збільшенні навантаження. Наприклад, це можуть бути база даних, мережеві з'єднання або серверні ресурси. Виявлення таких компонентів є важливим для подальшої оптимізації системи.

— Перевірка стабільності: Оцінка здатності системи підтримувати стабільну роботу протягом тривалого часу під постійним або змінним навантаженням.

3.3.2 Вибір інструментів для навантажувального тестування

Для досягнення цілей тестування обираються інструменти, що відповідають вимогам проєкту, зручні у використанні та забезпечують необхідні можливості для створення навантаження і моніторингу системи. Основними інструментами, які використовуються у цьому дослідженні, є Apache JMeter. Потужний інструмент з відкритим вихідним кодом, який надає гнучкі можливості для налаштування сценаріїв навантаження та підтримує різні протоколи (HTTP, HTTPS, FTP тощо). JMeter дозволяє детально налаштовувати сценарії та вимірювати основні показники продуктивності.

Вибір саме цього інструменту забезпечує можливість створення реалістичних сценаріїв навантаження та отримання точних даних про продуктивність системи.

3.3.3 Встановлення критеріїв успішності

Для оцінки результатів тестування визначено два ключові критерії успішності, які дозволяють об'єктивно інтерпретувати ефективність роботи системи під навантаженням:

— **Час відгуку (Response Time)**: Середній час відповіді на запит має залишатися в межах 2 секунд під стандартним навантаженням і не перевищувати 5 секунд за умов пікового навантаження. Це забезпечує задовільний рівень користувацького досвіду навіть у стресових ситуаціях.

— **Кількість помилок**: Відсоток помилкових запитів не повинен перевищувати 1% від загальної кількості запитів. Це гарантує стабільність і надійність роботи додатку навіть за високих навантажень.

Ці критерії є основними орієнтирами для оцінки продуктивності системи та допомагають провести порівняльний аналіз результатів тестування в різних умовах.

3.3.4 Розробка сценаріїв навантаження

Для оцінки роботи системи було розроблено сценарії тестування, які імітують сталі умови експлуатації додатку. Основна увага приділялася моделюванню стабільного навантаження з різною кількістю одночасних користувачів та різною

швидкістю їхнього підключення до системи. Це дозволяє оцінити поведінку веб-додатку в умовах постійного навантаження, яке є характерним для багатьох сучасних сервісів.

У тестуванні було реалізовано два основних сценарії:

— Сценарій зі 200 користувачами: У цьому сценарії імітується одночасна робота 200 користувачів, які підключаються до системи впродовж 20. Це дозволяє визначити, як система справляється з помірним навантаженням і чи є відмінності у її поведінці залежно від швидкості підключення користувачів.

— Сценарій із 500 користувачами: Тут моделюється робота 500 користувачів, які підключаються до системи за аналогічних умов — протягом 60 секунд. Цей сценарій дозволяє оцінити роботу системи за більш інтенсивного навантаження, яке може виникати в реальних умовах експлуатації.

На відміну від типових тестів із піковим чи динамічним зростанням навантаження, у цих сценаріях симулювалося сталий стан системи, де кількість активних користувачів залишається приблизно постійною після завершення етапу підключення. Це підхід дозволяє зосередитися на аналізі стабільності та продуктивності веб-додатку у тривалих умовах використання.

Зібрані результати дають змогу оцінити такі показники, як час відповіді та кількість помилкових запитів за умов різного рівня навантаження, а також ідентифікувати потенційні точки для подальшої оптимізації системи.

3.4 Підготовка інструментів тестування

Для проведення навантажувального тестування було обрано Apache JMeter, один із найпопулярніших інструментів для моделювання навантаження, аналізу продуктивності веб-додатків та API. JMeter надає два режими роботи: CLI для

продуктивних тестів і GUI для зручного проєктування сценаріїв. У тестувальницькій спільноті склалася практика, за якої GUI використовується на початкових етапах створення сценаріїв, а для запуску великих проєктів під високим навантаженням використовується CLI, що дозволяє зменшити споживання ресурсів системи.

3.4.1 Огляд структури проекту Apache JMeter

Для дослідження продуктивності обраного середовища було створено проєкт зі специфічною структурою, яка охоплює різні аспекти запитів до сервера, обробку відповідей і логування.

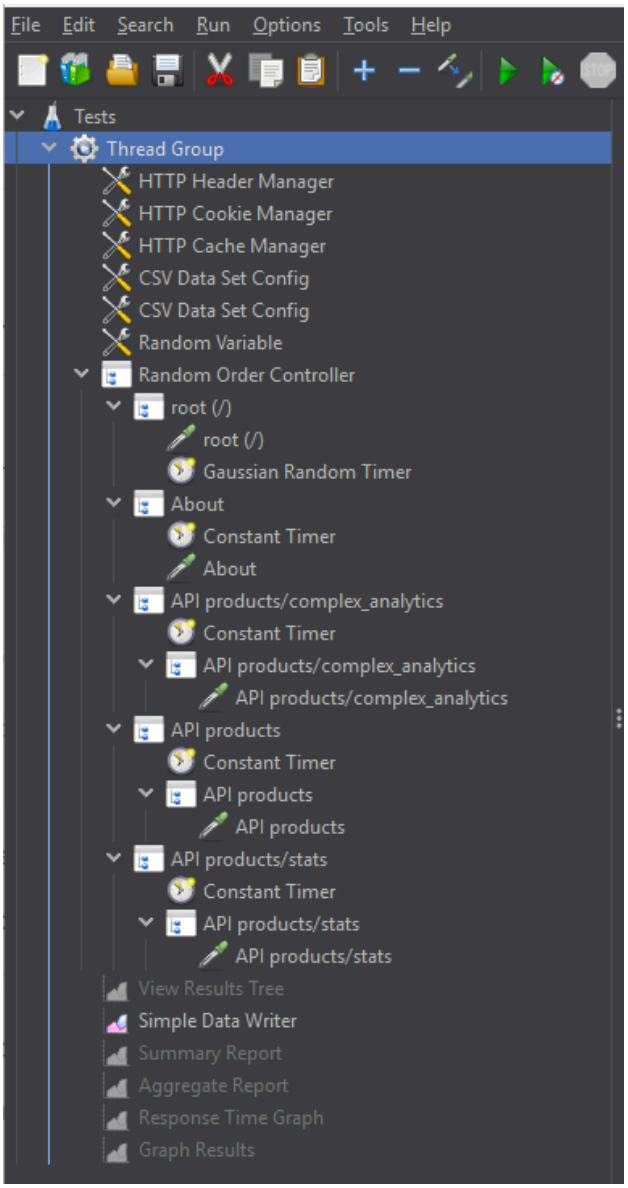


Рисунок 3.16 - Структура тестового проєкту

Основою тестування є **Thread Group**, що включає ключові компоненти:

— HTTP Header Manager – для налаштування HTTP-заголовків, що додаються до кожного запиту. Це дозволяє емулювати поведінку справжнього браузера та уникати блокування з боку серверів.

— HTTP Cookie Manager – для керування куками. Завдяки цьому компоненту забезпечується відключення збереження сесій між запитами.

— HTTP Cache Manager – для управління кешем на рівні клієнта, що дозволяє протестувати сценарії, які передбачають використання збережених даних.

— Random Variable – для генерації випадкових значень, які застосовуються в параметрах запитів. Це забезпечує унікальність кожного запиту, що зменшує ризик кешування відповідей сервером.

— CSV Data Set Config – для підвантаження параметрів із файлів CSV. Це використовується для тестування із великою кількістю змінних, наприклад, для передачі різних категорій або типів запитів.

3.4.2 Основні компоненти контролера

Центральним елементом тесту є **Random Order Controller**, який дозволяє виконувати запити до окремих контролерів у випадковому порядку. Цей компонент створює реалістичне навантаження на сервер, імітуючи непередбачувану поведінку користувачів.

Контролери, які входять до складу Random Order Controller:

— **root (/)**: виконує базовий запит до головної сторінки сайту. Це GET-запит, який завантажує HTML, CSS, зображення та інші ресурси сторінки.

— **/api/products/complex_analytics**: запит до API з передачею параметрів category та random. Параметри генеруються динамічно, використовуючи Random Variable і CSV Data Set Config, що дозволяє тестувати сценарії з доступом до різних записів у базі даних.

— **/api/products**: запит для отримання списку продуктів із передачею параметрів категорії, типу сортування та сторінки. Цей запит імітує стандартний пошук користувачем у каталозі.

— **/api/products/stats**: запит для отримання статистичних даних із бази даних. Це SELECT-запит із базовою обробкою на сервері.

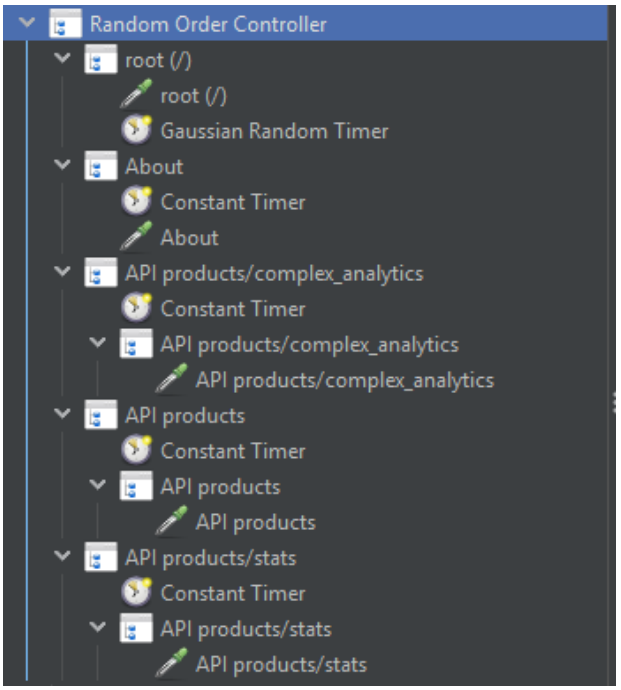


Рисунок 3.17 - Random Order Controller

3.4.3 Параметри роботи потоків

Кожен Request у Random Order Controller підтримує багатопотокове виконання в 5 потоків. Всі запити здійснюються через HTTP Client 4, який дозволяє обробляти великі обсяги даних і підтримує багатозадачність. Завантаження контенту виконується повністю, включаючи всі ресурси сторінки (HTML, CSS, зображення, відео).

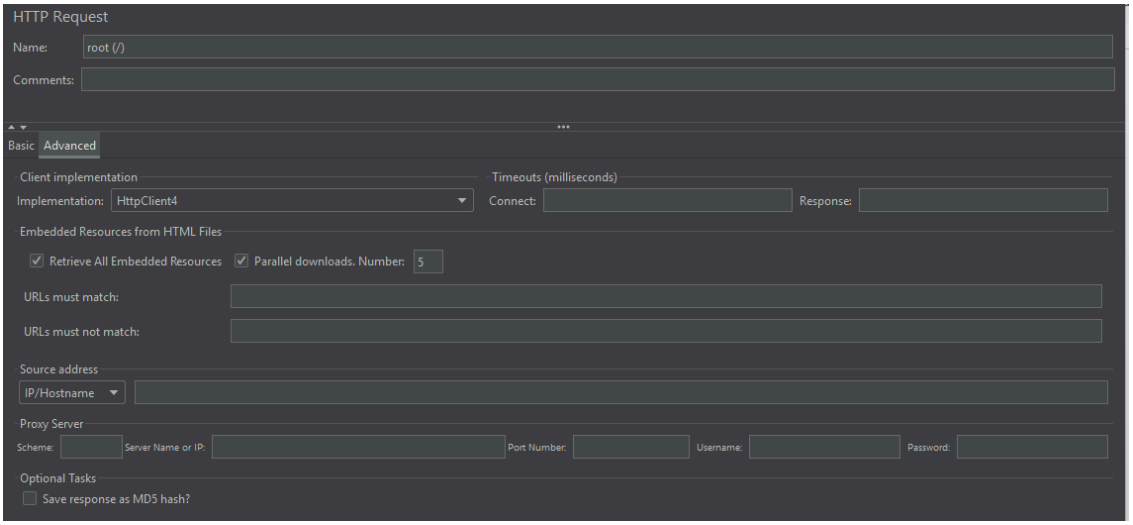


Рисунок 3.18 - Налаштування HTTP Request

3.4.4 Логування та обробка результатів

Рисунок 3.19 - Налаштування Data Writer

Для запису результатів використовується **Simple Data Writer**, який зберігає логи в CSV-файл. Це забезпечує можливість подальшого аналізу результатів, включаючи час відгуку сервера, статус кодів відповідей і час завантаження контенту.

3.4.5 Додаткові компоненти для імітації затримок

Для реалістичної імітації запитів використовується **Gaussian Random Timer**, який генерує випадкові затримки між запитами, імітуючи поведінку справжніх користувачів. Це допомагає уникнути надто синхронного навантаження на сервер, що може спотворювати результати тестування.

Крім цього, **Constant Timer** додає постійні затримки між запитами, дозволяючи краще контролювати розподіл навантаження.

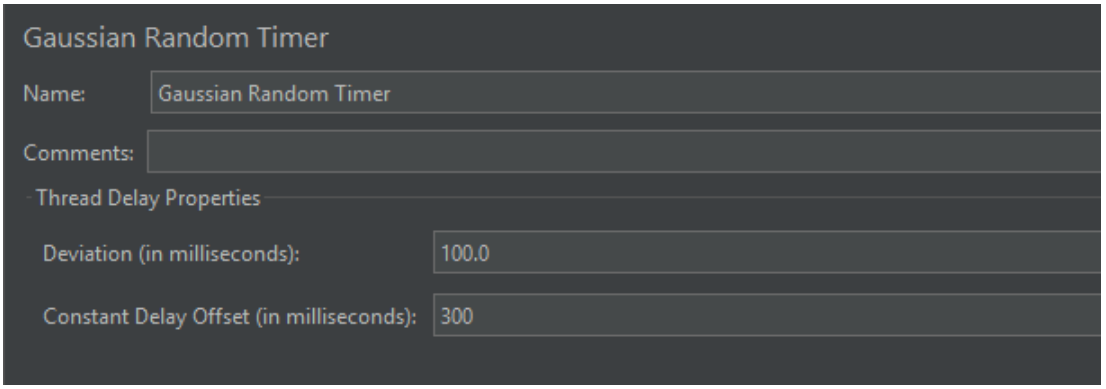


Рисунок 3.20 - Налаштування Gaussian Random Timer

3.5 Проведення тестування без методів оптимізації

Для оцінки базової продуктивності системи було проведено серію тестів, які дозволили визначити обмеження початкової конфігурації веб-додатку. У цьому розділі розглянуто результати двох експериментів, проведених без використання методів оптимізації, таких як Load Balancer, CloudFront чи Auto Scaling. Основною метою цих тестів було виявлення слабких місць системи, які могли б суттєво впливати на продуктивність за умов високого навантаження.

Під час тестування веб-додаток був розгорнутий на одному сервері у хмарній платформі AWS, без будь-якого додаткового балансування чи кешування. Навантаження на систему створювалося за допомогою інструменту JMeter, який генерував одночасні запити до серверу для симуляції реального навантаження.

3.5.1 Експерименти 1: 200 користувачів без оптимізації

Для першого етапу тестування використовувалась група із 200 віртуальних користувачів, які поступово збільшувались із 0 до 200 протягом 20 секунд (10 користувачів за секунду).

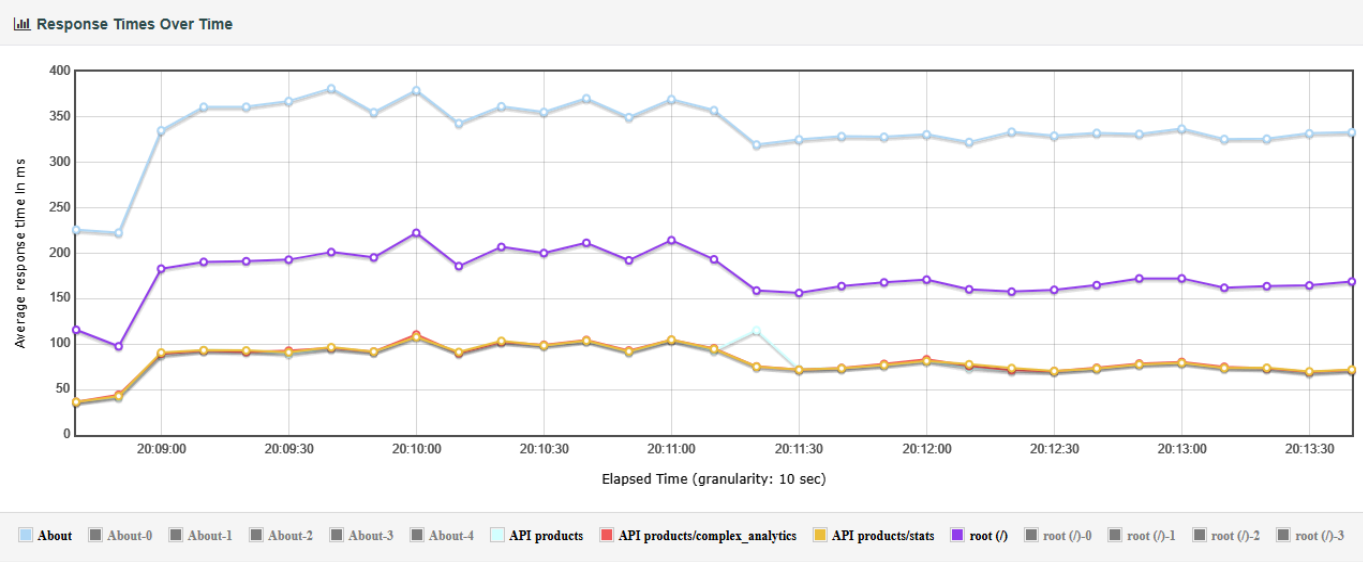


Рисунок 3.21 - Графік Response Time без оптимізації - 200 користувачів

У контрольному експерименті без використання оптимізаційних методів час відповіді залишався в межах прийнятного для більшості сторінок, проте вже на початкових етапах навантаження для сторінки root (/) спостерігалось збільшення часу відповіді. Це було пов’язано з тим, що сервер бази даних, розташований у Європі (Стокгольм), обробляв усі запити без допоміжних механізмів розподілу навантаження або кешування. Навантаження викликало поступове зростання часу відповіді, хоча система не зазнала серйозних відмов (помилки 503 або перевищення часу очікування практично не спостерігались). Цей тест є базовим для оцінки результатів інших експериментів.

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	**554243**	**1**	**0.00%**
About	100790	0	0.00%
About-0	50387	0	0.00%
About-1	50387	0	0.00%
About-2	50387	0	0.00%
About-3	50387	0	0.00%
About-4	25128	0	0.00%
API products	30190	2	0.01%
API products/complex_analytics	50271	0	0.00%
API products/stats	20142	0	0.00%
root (/)	100752	0	0.00%
root (/)-0	50350	0	0.00%
root (/)-1	50350	0	0.00%
root (/)-2	50350	0	0.00%
root (/)-3	25291	0	0.00%

Рисунок 3.22 - Таблиця помилок в ході Експерименту 1

3.5.2 Експерименти 2: 500 користувачів без оптимізації

У другій серії тестів кількість віртуальних користувачів збільшилась до 500. Навантаження зростало поступово: від 0 до 500 протягом 60 секунд (по 10 користувачів щосекунди). Основною метою було оцінити стійкість системи за умов значного навантаження.

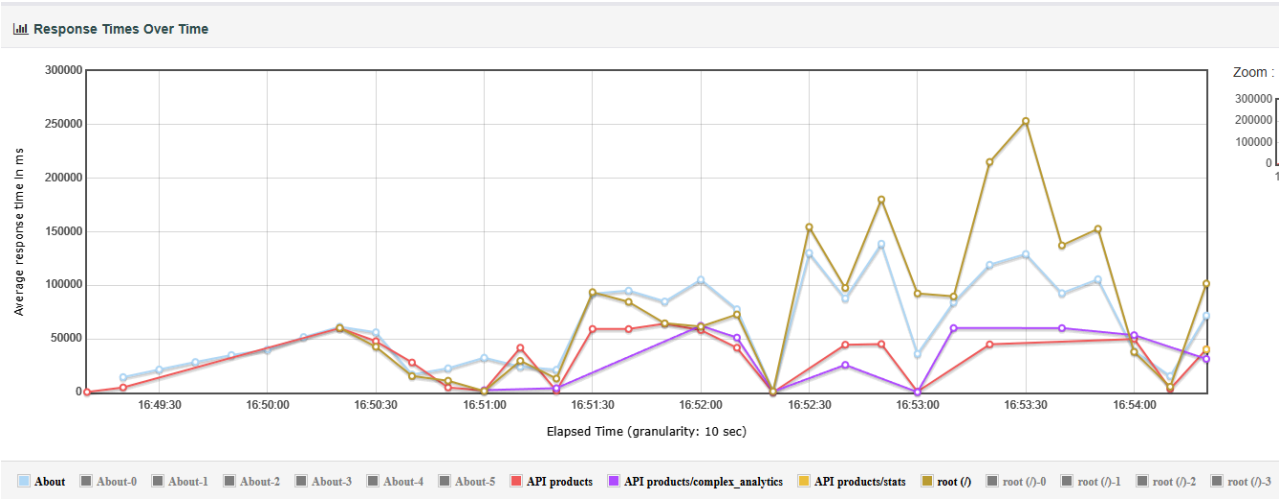


Рисунок 3.23 - Графік Response Time без оптимізації - 500 користувачів

Графік без оптимізації показує суттєве збільшення часу відповіді для всіх сторінок, включаючи **API**. Спостерігалось перевищення часу очікування та численні помилки 503. Відсоток невдалих запитів у цьому випадку досягав понад 50%, особливо для сторінок із медіаконтентом. Це свідчить про недостатню масштабованість системи без використання сучасних методів оптимізації.

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	**5335**	**2946**	**55.22%**
About	1178	1028	87.27%
About-0	338	0	0.00%
About-1	338	49	14.50%
About-2	338	234	69.23%
About-3	338	235	69.53%
About-4	338	240	71.01%
About-5	285	195	68.42%
API products	1010	114	11.29%
API products/complex_analytics	62	40	64.52%
API products/stats	2	0	0.00%
root (/)	1232	1138	92.37%
root (/)-0	325	0	0.00%
root (/)-1	324	75	23.15%
root (/)-2	324	252	77.78%
root (/)-3	324	259	79.94%
root (/)-4	242	195	80.58%

Рисунок 3.24 - Таблиця помилок в ході Експерименту 2

3.6 Оптимізація хмарної інфраструктури для веб-додатку

У цьому розділі описано процес налаштування інфраструктури веб-додатку на платформі AWS. Основна увага приділяється налаштуванню Load Balancer,

інтеграції з CloudFront, а також використанню Auto Scaling для забезпечення продуктивності, масштабованості та стійкості до високих навантажень.

3.6.1 Вибір типу Load Balance

Для забезпечення балансування навантаження на веб-додаток було обрано **Application Load Balancer** (ALB). Цей тип Load Balancer дозволяє використовувати гнучкі методи маршрутизації та інтегрується з іншими сервісами AWS, такими як Auto Scaling. ALB працює на рівні запитів і підходить для додатків, що обробляють HTTP та HTTPS-трафік.

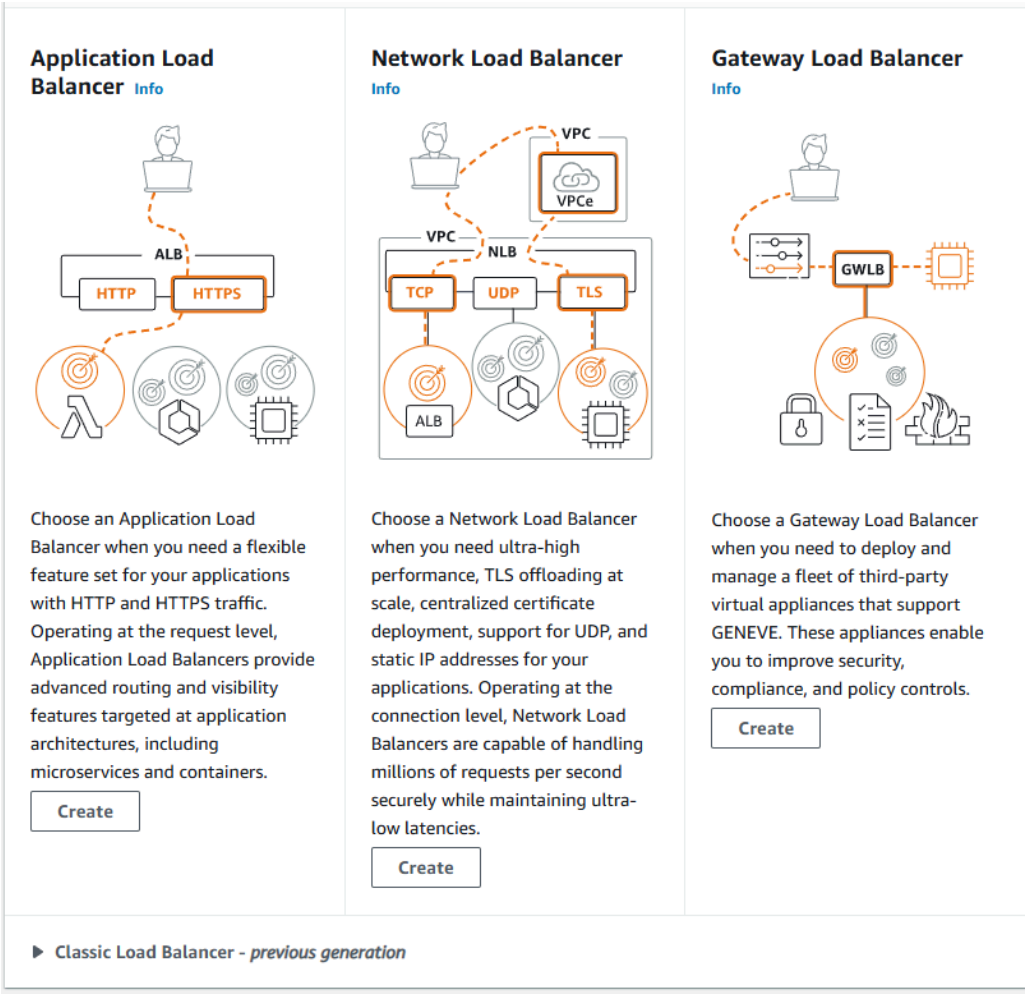


Рисунок 3.25 - Типи Load Balancer в AWS

Обрана конфігурація передбачала роботу балансувальника у режимі Internet-facing, що забезпечує доступ до додатку через інтернет. Для підтримки

сучасних протоколів було активовано режим IPv4, що дозволяє одночасно працювати з IPv4 адресами, адже в додатку не передбачено використання IPv6 адрес. Ці налаштування чітко видно на Рисунок 3.26, де представлена базова конфігурація ALB.

На Рисунку 3.27 зображено налаштування Load Balancer, які дозволяють встановити алгоритм розподілу трафіку, а також додаткові параметри для покращення продуктивності та надійності інфраструктури.

Load Balancer надає кілька варіантів алгоритмів, кожен із яких адаптований до певних сценаріїв використання:

— Round Robin (за замовчуванням)

— Least Outstanding Requests

— Weighted Random: Новий алгоритм, який випадково розподіляє запити між "здоровими" інстансами з урахуванням ваги, яка призначається кожному інстансу. Використання ваг дозволяє враховувати обчислювальну потужність серверів та забезпечувати ефективніший розподіл запитів у неоднорідних середовищах.

Механізми підвищення надійності

У налаштуваннях також передбачено активацію Anomaly Mitigation — функції автоматичного перенаправлення трафіку від інстансів, які виявлені як аномальні. Це підвищує стійкість системи до несподіваних збоїв або нестабільності одного з компонентів.

Додаткові параметри

— Slow Start Duration: Цей параметр дозволяє плавно інтегрувати нові інстанси до групи шляхом обмеження кількості запитів, що надходять до них у перші секунди після запуску. Встановлення значення "0 секунд" вимикає цей механізм.

— Cross-Zone Load Balancing: Опція забезпечує рівномірний розподіл запитів між інстансами у різних зонах доступності. Це важливий механізм для забезпечення географічної стійкості додатку.

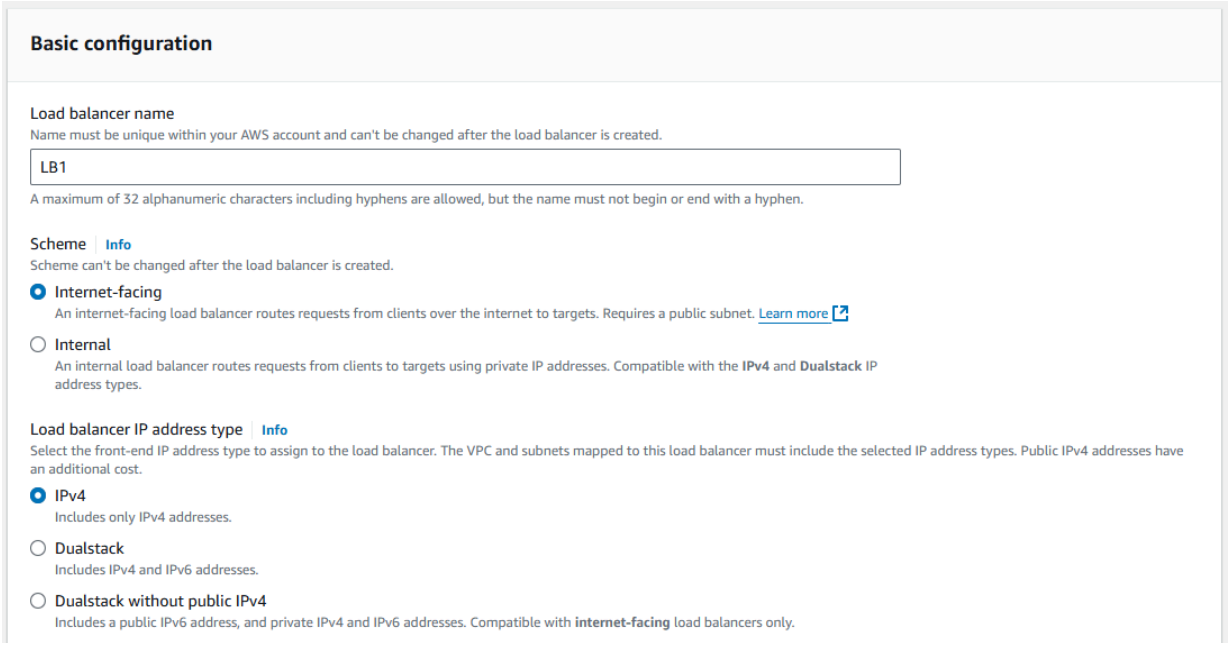


Рисунок 3.26 - Налаштування Load Balancer

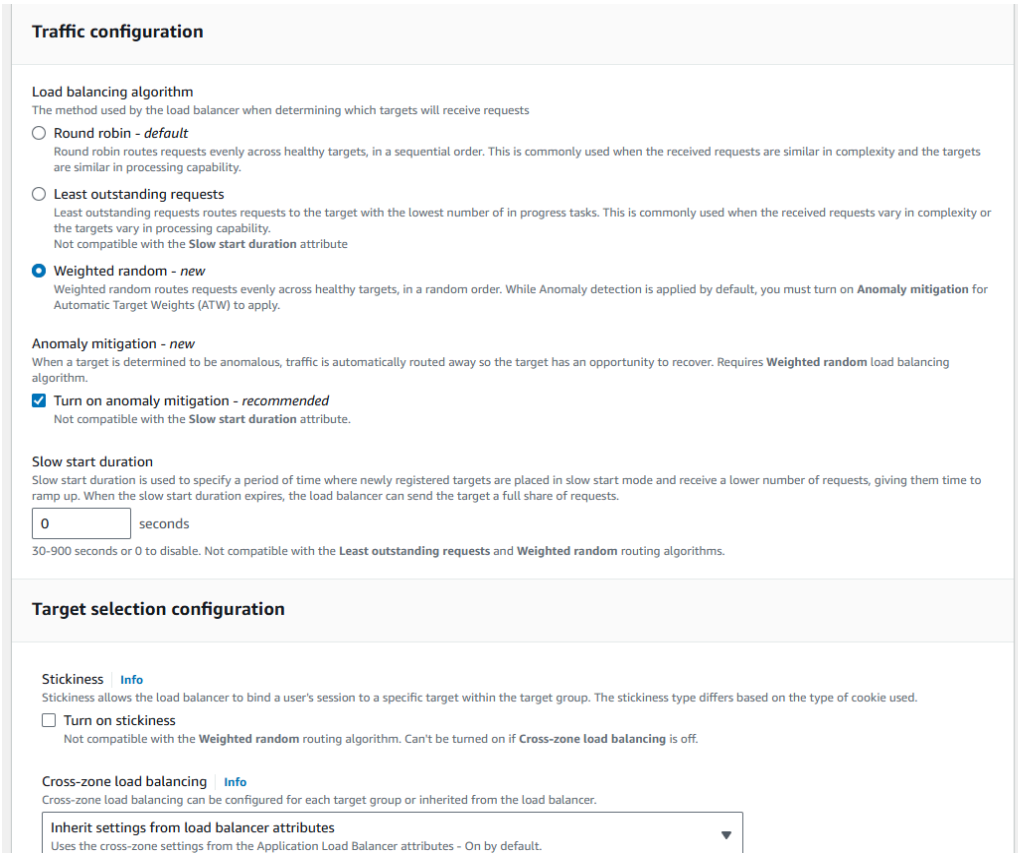


Рисунок 3.27 - Налаштування Load Balancer

3.6.2 Налаштування слухачів (Listeners)

Для обробки запитів було створено два слухачі. Слухач для HTTP (порт 80) налаштований на автоматичне перенаправлення всіх запитів на HTTPS, що забезпечує безпеку передачі даних. Основний трафік обробляється через HTTPS (порт 443), використовуючи сертифікати SSL/TLS, видані через **AWS Certificate Manager (ACM)**. Налаштування цих слухачів, а також пов'язані правила маршрутизації показані на Рисунок 3.28.

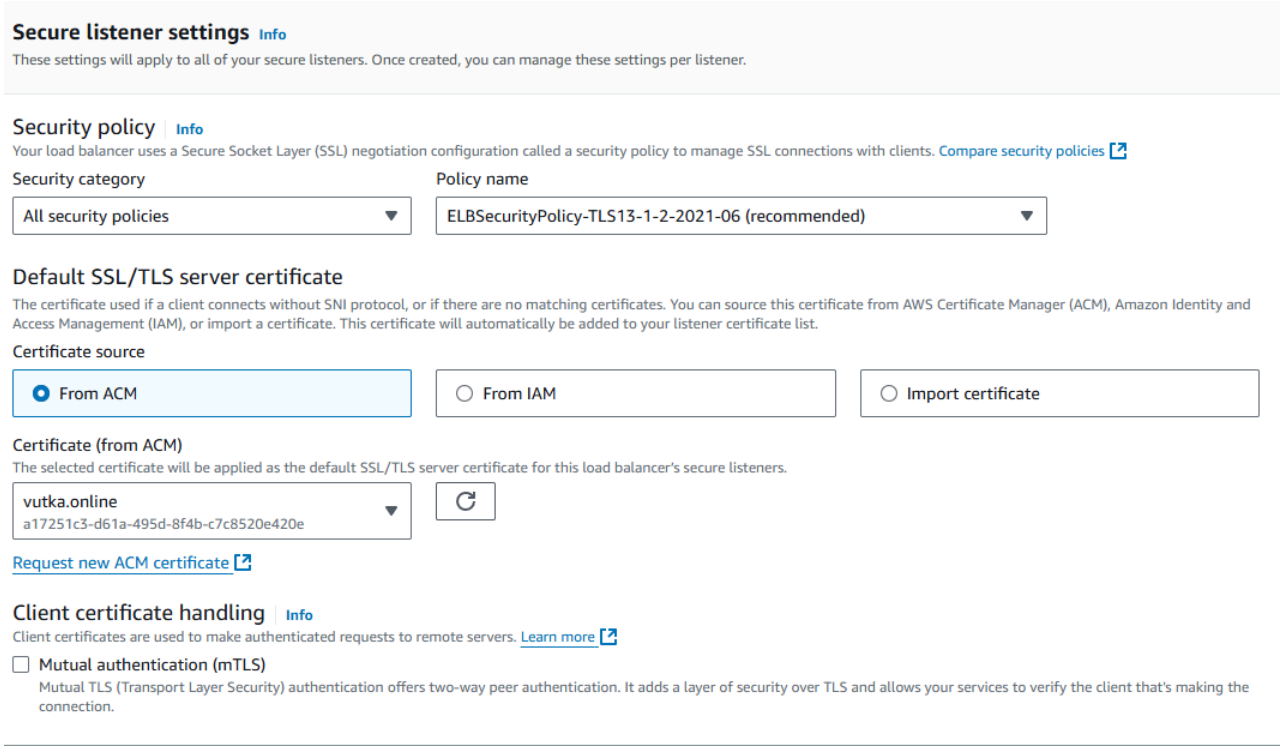


Рисунок 3.28 - Налаштування політик безпеки

3.6.3 Інтеграція CloudFront

Для прискорення роботи веб-додатку та оптимізації навантаження на сервери було інтегровано CloudFront як Content Delivery Network (CDN). CloudFront зменшує затримки при завантаженні контенту, кешуючи статичні файли, такі як зображення, CSS та JavaScript, на серверах, розташованих ближче до кінцевих користувачів.

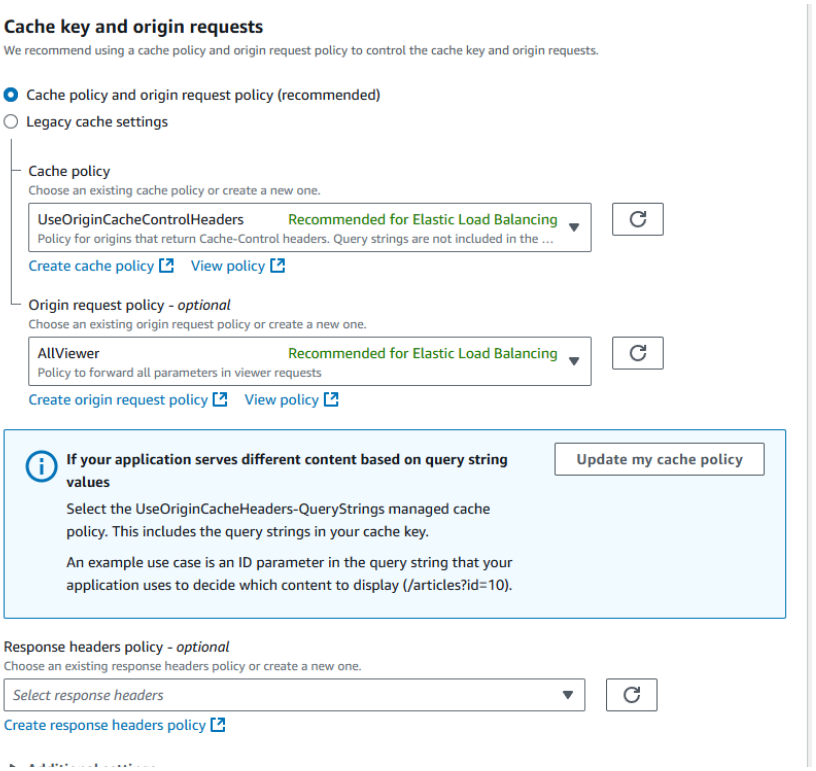
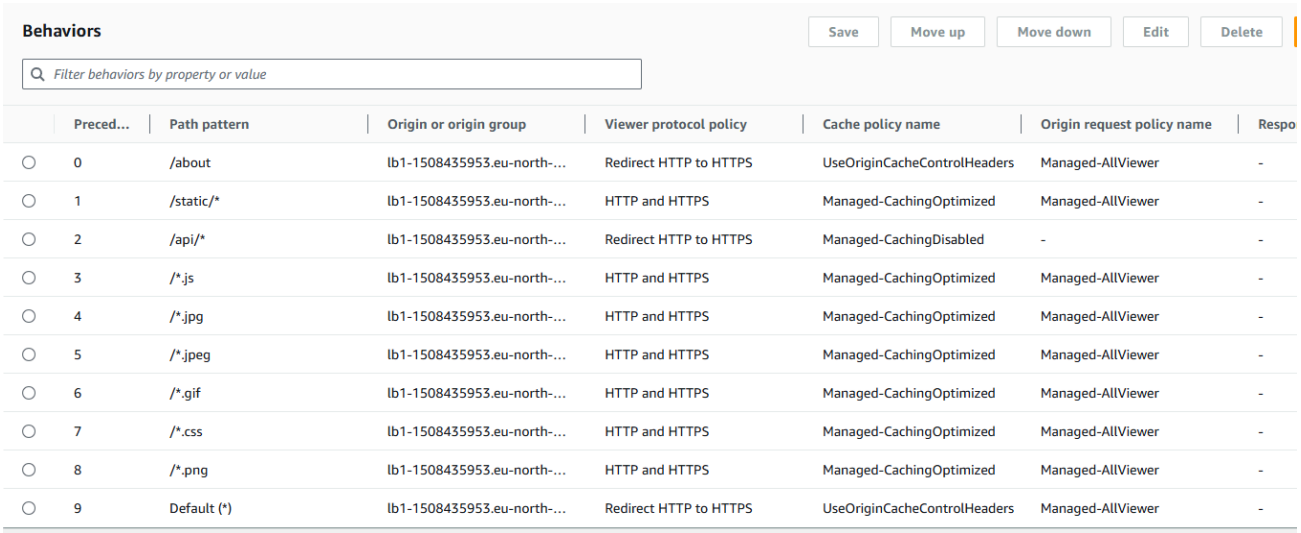


Рисунок 3.29 - Налаштування CloudFront

Джерелом контенту для CloudFront було обрано ALB. Для різних типів запитів, як показано на Рисунок 3.30, налаштовано окремі політики кешування: для статичного контенту використовується політика Managed-CachingOptimized, тоді як для API-запитів кешування вимкнено. Такі налаштування дозволяють ефективно розподіляти навантаження між серверами.



	Preced...	Path pattern	Origin or origin group	Viewer protocol policy	Cache policy name	Origin request policy name	Respo
○	0	/about	lb1-1508435953.eu-north-...	Redirect HTTP to HTTPS	UseOriginCacheControlHeaders	Managed-AllViewer	-
○	1	/static/*	lb1-1508435953.eu-north-...	HTTP and HTTPS	Managed-CachingOptimized	Managed-AllViewer	-
○	2	/api/*	lb1-1508435953.eu-north-...	Redirect HTTP to HTTPS	Managed-CachingDisabled	-	-
○	3	/*.js	lb1-1508435953.eu-north-...	HTTP and HTTPS	Managed-CachingOptimized	Managed-AllViewer	-
○	4	/*.jpg	lb1-1508435953.eu-north-...	HTTP and HTTPS	Managed-CachingOptimized	Managed-AllViewer	-
○	5	/*.jpeg	lb1-1508435953.eu-north-...	HTTP and HTTPS	Managed-CachingOptimized	Managed-AllViewer	-
○	6	/*.gif	lb1-1508435953.eu-north-...	HTTP and HTTPS	Managed-CachingOptimized	Managed-AllViewer	-
○	7	/*.css	lb1-1508435953.eu-north-...	HTTP and HTTPS	Managed-CachingOptimized	Managed-AllViewer	-
○	8	/*.png	lb1-1508435953.eu-north-...	HTTP and HTTPS	Managed-CachingOptimized	Managed-AllViewer	-
○	9	Default (*)	lb1-1508435953.eu-north-...	Redirect HTTP to HTTPS	UseOriginCacheControlHeaders	Managed-AllViewer	-

Рисунок 3.30 - Політики кешування контенту

3.6.4 Автоматичне масштабування (Auto Scaling)

Налаштування політик масштабування

Для масштабування інфраструктури було застосовано політику Target Tracking Scaling, яка налаштована на основі ключових метрик:

— Середній мережевий трафік (Average Network In, Bytes): Використовується для моніторингу обсягу вхідного трафіку на сервери.

— Кількість запитів на інстанс (Request Count per Target): Визначає кількість запитів, які надходять на один сервер через Load Balancer.

Ця політика дозволяє гнучко регулювати кількість серверів залежно від поточного навантаження, запобігаючи перевантаженню системи або неефективному використанню ресурсів.

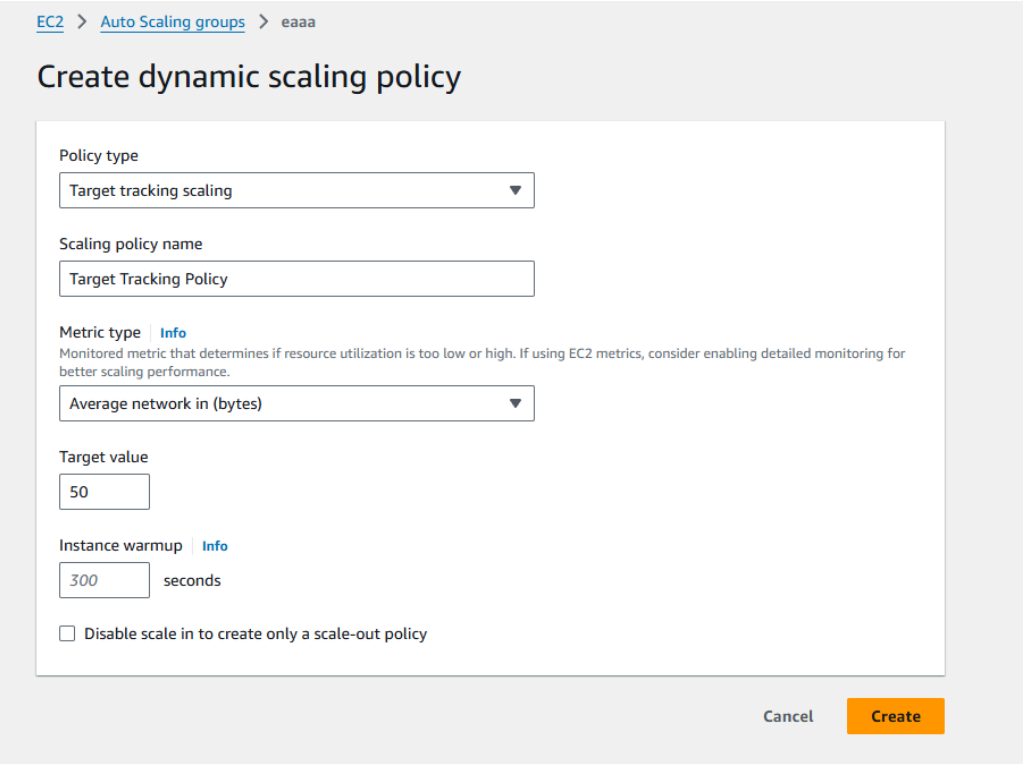


Рисунок 3.31 - Вікно налаштувань Auto Scaling groups

На Рисунку 3.32 зображено вибір метрики "Request Count per Target", яка дозволяє оптимально балансувати навантаження між серверами, додаючи нові інстанси за необхідності.

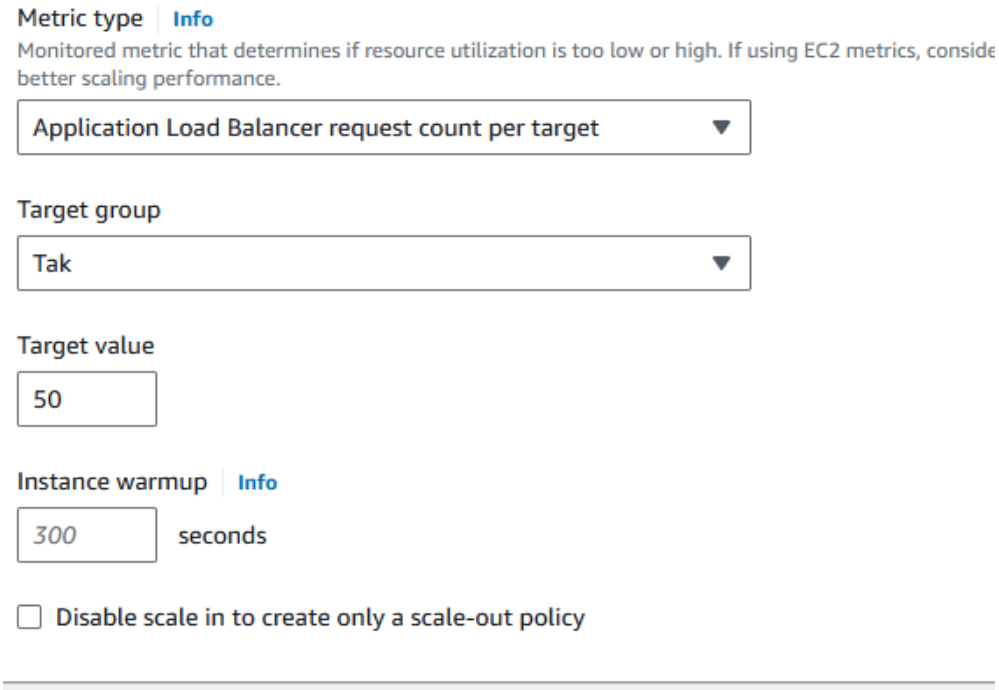


Рисунок 3.32 - Налаштування логіки створення інстансів

Час очікування перед масштабуванням

Для забезпечення стабільного приєднання нових інстансів після їх запуску встановлено параметр Warm-Up Period на 300 секунд. Цей час дає змогу новим інстансам повністю завантажити всі необхідні компоненти та інтегруватися в систему перед обробкою реальних запитів.

Конфігурація кількості інстансів

Група Auto Scaling була налаштована зі стартовою кількістю серверів у 3 інстанси, з можливістю масштабування до 8 інстансів залежно від рівня навантаження. Це налаштування дозволяє підтримувати баланс між витратами на ресурси та забезпеченням продуктивності веб-додатку.

На Рисунку 3.33 представлено список інстансів, які були створені Auto Scaling Group. Усі інстанси успішно зареєстровані в Load Balancer та готові до обробки запитів.

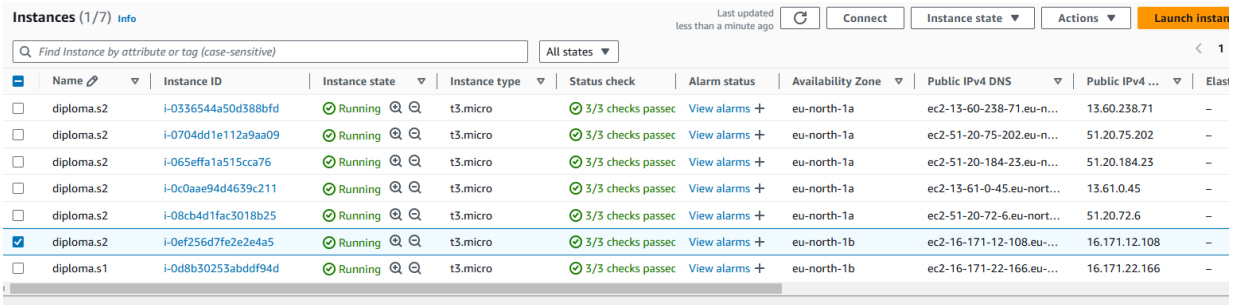


Рисунок 3.33 - Автоматично створені інстанси

3.6.5 Схема оптимізованої хмарної інфраструктури

Оптимізована хмарна інфраструктура, зображена на Рисунок 3.34, є зображенням багаторівневої системи, побудованої для забезпечення високої продуктивності та надійності веб-додатків.

Компоненти архітектури

— Cloudflare: Виступає як перший рівень захист, забезпечуючи захист від DDoS-атак та перенаправлення DNS запитів до CloudFront.

— CloudFront: Працює як глобальна мережа доставки контенту (CDN), оптимізуючи доставку кешованих даних користувачам, знижуючи навантаження на основний серверний рівень та скорочуючи час відповіді.

— Load Balancer: Забезпечує рівномірний розподіл динамічних запитів між кількома серверними інстансами.

— Автоматичні групи масштабування (Auto Scaling Groups): Містять кілька інстансів, кожен з яких об'єднує:

  - Nginx для керування HTTP-запитами.
  - Gunicorn як сервер додатків, що запускає бекенд-додаток.
  - Flask як каркас для веб-розробки.
  - Локальну базу даних, яка використовується для зберігання критично важливих даних.

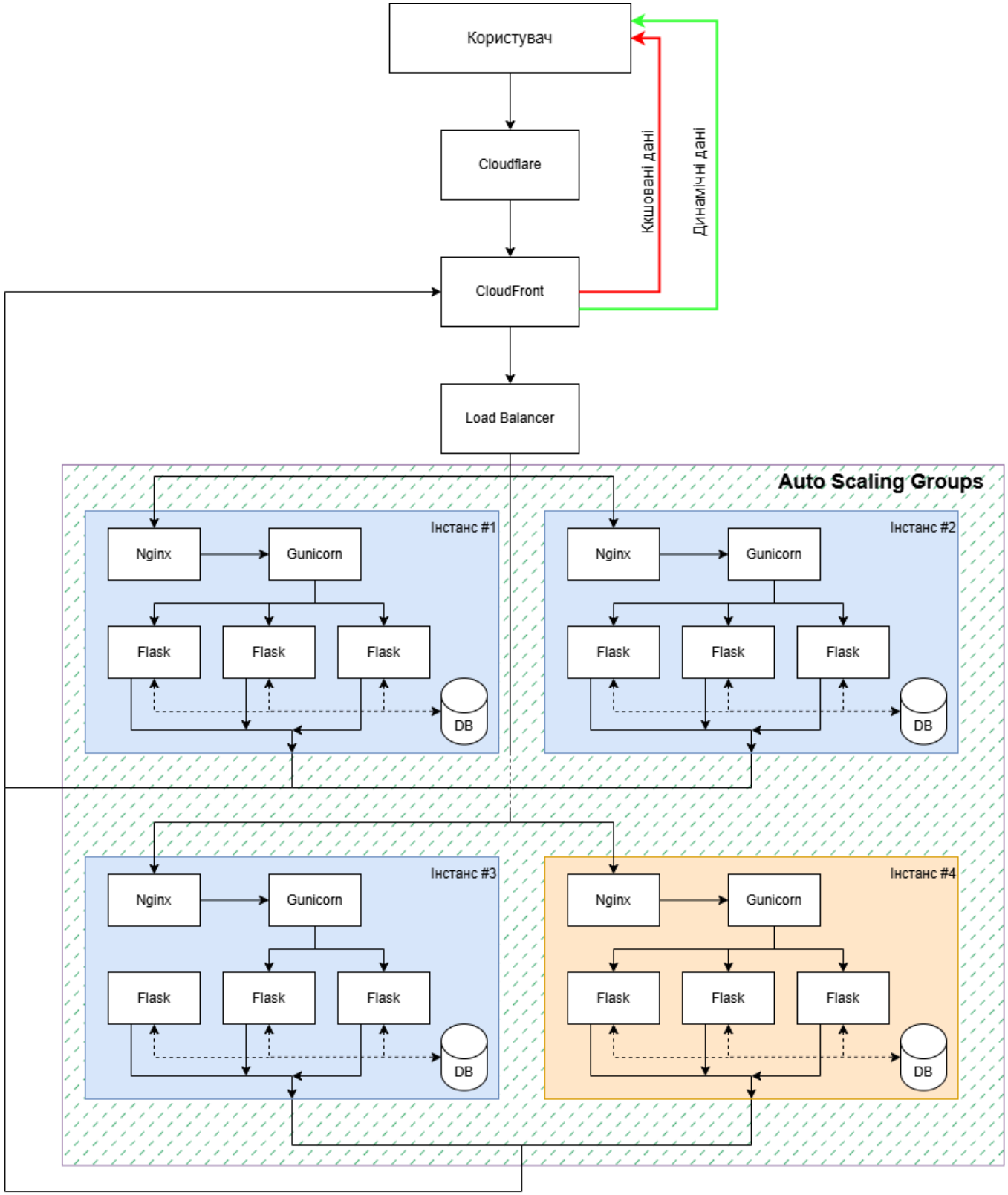


Рисунок 3.34 - Блок схема оптимізованої хмарної інфраструктури

3.7 Проведення тестування використовуючи методи оптимізації

У цьому розділі детально розглядається процес тестування після впровадження оптимізаційних методів, таких як балансування навантаження, автоматичне масштабування та використання технологій кешування. Метою тестування є перевірка ефективності застосованих оптимізацій у реальних умовах стабільного навантаження.

Для порівняння результатів було обрано аналогічні сценарії навантаження, як і в попередніх тестах без оптимізації. Це дозволяє оцінити вплив застосованих методів на продуктивність системи, час відгуку та кількість помилкових запитів. Основна увага приділяється перевірці стійкості системи, її здатності адаптуватися до змін у кількості користувачів та ефективності розподілу ресурсів.

Результати тестів у цьому розділі слугуватимуть основою для порівняння з попередніми експериментами та дозволять зробити висновки про ефективність оптимізаційних підходів, що використовуються для покращення роботи мережевої інфраструктури.

3.7.1 Експерименти 3: 200 користувачів з оптимізацією

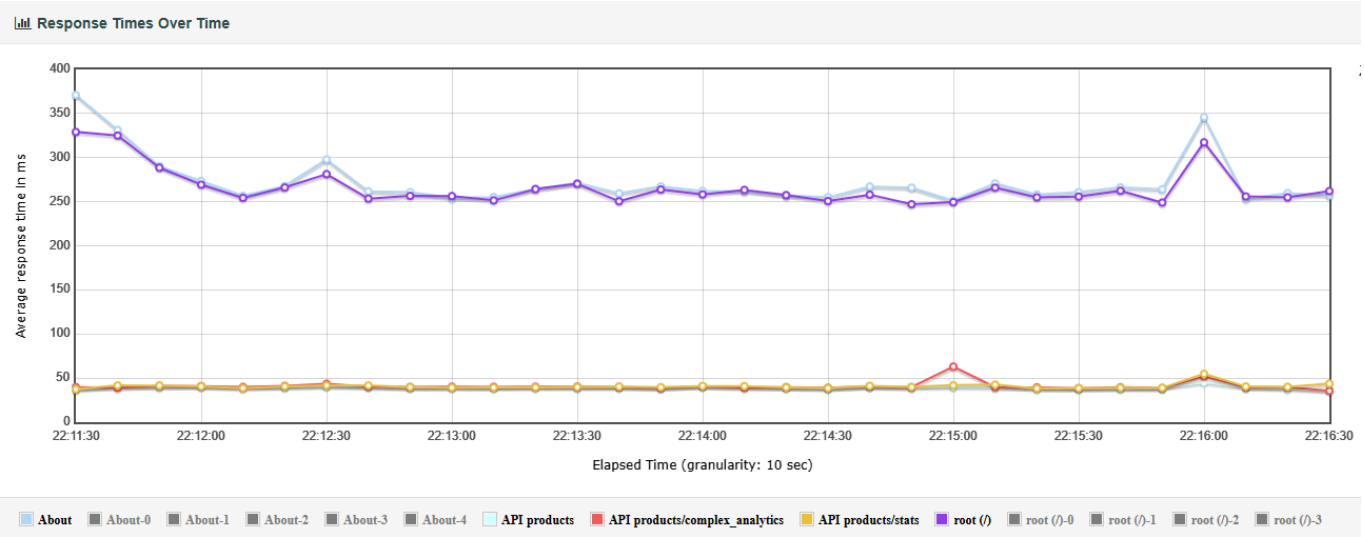


Рисунок 3.35 - Графік Response Time з оптимізацією - 200 користувачів

Експеримент з оптимізацією включав використання Amazon CloudFront, що забезпечувало доступність сторінок через CDN. У результаті час відповіді для більшості сторінок зменшився, хоча для сторінки **root(/)** він залишався високим. Це явище пояснюється специфікою роботи CloudFront, який розподілив медіазапити у різні регіони з CDN серверами (наприклад, США чи Західній Європі) замість безпосереднього звернення до сервера в Стокгольмі. На початку тесту спостерігається поступове зменшення часу відповіді, попри зростання кількості користувачів. Це пояснюється тим, що CloudFront почав кешувати дані, ефективно розподіляючи контент по CDN-серверам Amazon, що знизило навантаження на основний сервер.

Проте на позначці 22:16 графік демонструє пік часу відповіді, що, ймовірно, пов’язано із закінченням часу життя кешу або перерозподілом запитів на інший CDN-сервер. Подібне зростання часу відповіді є типовим для сценаріїв, де відбувається повторна побудова кешу або зміна маршруту запитів між регіонами. У ході подальшого налаштування час життя кешу було збільшено, що мало б усунути подібні піки у відповідях у тривалих навантажувальних тестах.

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	**585865**	**1**	**0.00%**
API products/complex_analytics	48699	2	0.00%
API products/stats	19527	0	0.00%
About	97656	0	0.00%
root (/)-3	48811	0	0.00%
About-3	48814	0	0.00%
root (/)	97665	0	0.00%
About-2	48814	0	0.00%
root (/)-1	48811	0	0.00%
About-1	48814	0	0.00%
About-0	48814	0	0.00%
root (/)-2	48811	0	0.00%
About-4	48814	0	0.00%
API products	29269	0	0.00%
root (/)-0	48811	0	0.00%

Рисунок 3.36 - Таблиця помилок в ході Експерименту 3

3.7.2 Експерименти 4: 500 користувачів з оптимізацією

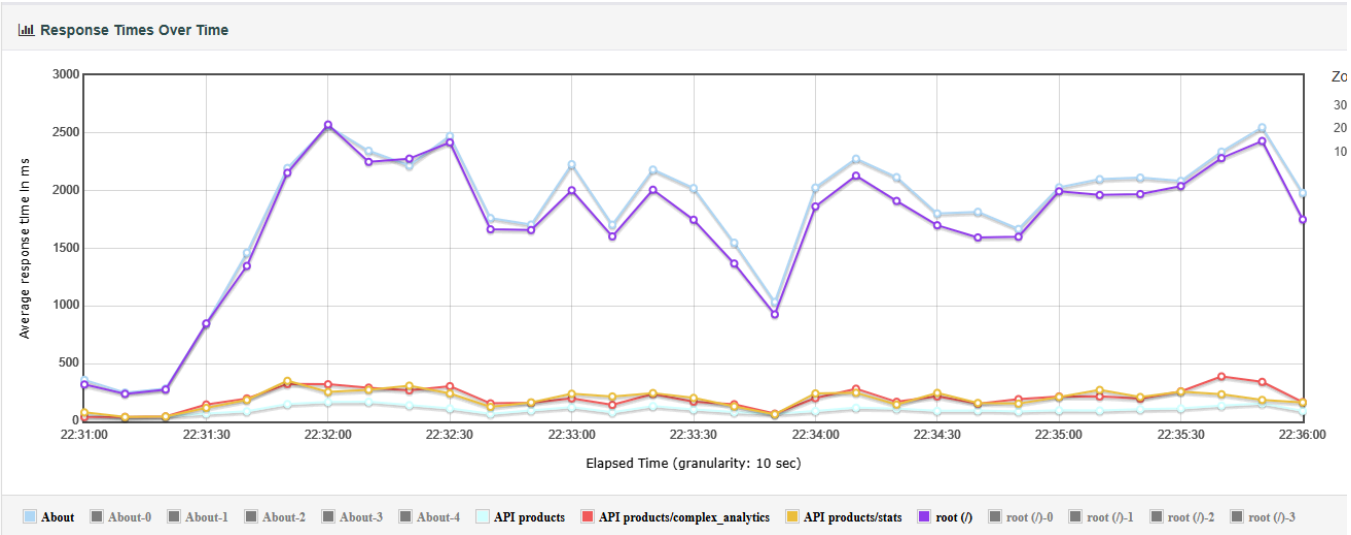


Рисунок 3.37 - Графік Response Time з оптимізацією - 500 користувачів

На Рисунку 3.36 представлено результати тестування оптимізованої інфраструктури при навантаженні 500 користувачів. У цьому експерименті система продемонструвала значно кращу стабільність у порівнянні з тестуванням без оптимізації. Хоча затримка у відповіді сервера дещо збільшилася, це очікуваний результат через високу кількість одночасних запитів. Водночас кількість помилок залишається практично нульовою, що свідчить про ефективність розподілу навантаження за допомогою Elastic Load Balancer.

Завдяки оптимізованому налаштуванню, Elastic Load Balancer забезпечив динамічне створення нових віртуальних машин, які дозволили ефективно розподілити трафік між серверами. Це було досягнуто завдяки алгоритму Round Robin, що забезпечує рівномірний розподіл запитів. Така архітектура дозволила уникнути перевантаження окремих серверів і гарантувати стабільну роботу навіть за умов високого навантаження.

Requests	Executions		
Label	#Samples	FAIL	Error %
Total	**383273**	**5**	**0.00%**
About	63781	0	0.00%
About-0	32021	0	0.00%
About-1	31888	1	0.00%
About-2	32004	1	0.00%
About-3	32007	1	0.00%
About-4	31929	0	0.00%
API products	19119	0	0.00%
API products/complex_analytics	31680	0	0.00%
API products/stats	12761	0	0.00%
root (/)	63721	0	0.00%
root (/)-0	32002	0	0.00%
root (/)-1	31830	1	0.00%
root (/)-2	31986	0	0.00%
root (/)-3	31988	1	0.00%

Рисунок 3.38 - Таблиця помилок в ході Експерименту 1

3.8 Аналіз результатів

Результати тестування продемонстрували значний вплив впроваджених методів оптимізації, зокрема використання **Amazon CloudFront** та **Elastic Load Balancer**, на продуктивність веб-додатку. Проведені експерименти виявили ключові аспекти роботи системи у сценаріях з оптимізацією та без.

У контрольному експерименті, де оптимізація не застосовувалася, система демонструвала задовільний час відповіді за умови стабільного навантаження у 200 користувачів. Проте зі збільшенням кількості користувачів до 500 спостерігалися значні затримки у відповіді серверів, особливо для сторінок із медіаконтентом. Окрім того, частота помилок та відмов сервера різко зростала, що свідчить про недостатню масштабованість архітектури без інструментів розподілу навантаження. Значна частина API-запитів завершувалася помилкою або перевищенням часу очікування.

Натомість у тестах із впровадженням оптимізації було досягнуто наступних покращень:

— Зниження часу відповіді:

Час відповіді суттєво покращився для більшості сторінок, зокрема для API-запитів. Це стало можливим завдяки використанню кешування статичних ресурсів через Amazon CloudFront та ефективному балансуванню трафіку на рівні Elastic Load Balancer. Навіть для сторінок із медіаконтентом час відповіді залишався стабільним.

— Зменшення кількості помилок:

У тестах із застосуванням оптимізації кількість помилок значно знизилася, а в деяких випадках помилки були відсутні взагалі. Це свідчить про ефективне масштабування інфраструктури завдяки динамічному створенню додаткових віртуальних машин через **Auto Scaling Group**.

— Стабільність під навантаженням:

Під час тестування на 500 користувачів система продемонструвала стабільний час відповіді для API-запитів та змогла уникнути перевантаження серверів. Вдалося підтримувати постійний рівень продуктивності навіть за умов тривалого навантаження завдяки алгоритму Round Robin у **Elastic Load Balancer** та його інтеграції з Auto Scaling Group.

Проте, попри значне покращення, залишаються окремі виклики:

— Медіаконтент: сторінки, які містять великий обсяг медіаконтенту, демонструють збільшення часу відповіді, навіть за умов використання кешування через **Amazon CloudFront**. Це свідчить про необхідність подальшої оптимізації, можливо, через більш агресивне кешування або використання додаткових точок обробки контенту.

Підсумовуючи, результати тестування підтвердили ефективність оптимізації інфраструктури за допомогою хмарних інструментів, таких як Amazon CloudFront, Elastic Load Balancer та Auto Scaling. Система змогла забезпечити стабільну та надійну роботу за високого навантаження, зменшивши час відповіді та практично виключивши помилки. Це створює стійку основу для подальшого вдосконалення та оптимізації продуктивності додатку.

3.9 Теоретичний огляд методів оптимізації мережевої інфраструктури в локальній мережі:

Оптимізація локальної мережі (LAN) відіграє важливу роль у забезпеченні її стабільності, продуктивності та безпеки. У цьому розділі розглядаються ключові методи, що включають управління трафіком, балансування навантаження та сегментацію мережі, із застосуванням сучасного обладнання, такого як MikroTik

CCR2216, MikroTik CRS518 та FortiADC 5000F. Основна увага приділяється практичним аспектам використання цих технологій для забезпечення ефективності та масштабованості мережі.

3.9.1 Управління трафіком через QoS

Quality of Service (QoS) є важливим інструментом для забезпечення стабільності мережі в умовах високого навантаження. Використання QoS дозволяє пріоритизувати критичний трафік, як-от VoIP та відеоконференції, зменшуючи затримки та втрати пакетів. Завдяки підтримці RouterOS, MikroTik CCR2216 дозволяє реалізувати складні політики управління трафіком, забезпечуючи стабільність навіть за пікового навантаження.

Рисунок 3.39 - MikroTik CCR2216-1G-12XS-2XQ

Таблиця 3.1 - Характеристики MikroTik CCR2216-1G-12XS-2XQ

MikroTik CCR2216-1G-12XS-2XQ	**Пропускна здатність:** 100G QSFP28 і 25G SFP28 порти. **Процесор:** 16-ядерний, 2 ГГц. **Операційна система:** RouterOS із підтримкою QoS.

Для огляду та застосування в локальній мережі було обрано **MikroTik CCR2216-1G-12XS-2XQ**, оскільки його висока продуктивність та підтримка QoS дозволяють ефективно пріоритизувати трафік у корпоративних мережах. Наприклад, це забезпечує стабільну роботу таких сервісів, як VoIP та відеоконференції, навіть у періоди пікового навантаження.

3.9.2 Балансування навантаження

Балансування навантаження є необхідним для розподілу трафіку між серверами або мережевими пристроями, забезпечуючи їхню високу доступність і стабільність роботи.

Рисунок 3.40 - FortiADC 5000F

Таблиця 3.2 - Характеристики FortiADC 5000F

FortiADC 5000F	**Пропускна здатність L4:** до 200 Гбіт/с. **Пропускна здатність L7:** до 150 Гбіт/с. **Алгоритми розподілу:** Least Connections, Weighted Round Robin. **Моніторинг:** Автоматичне перенаправлення трафіку до найменш завантажених серверів. **SSL-офлоадинг:** До 50 Гбіт/с.

FortiADC 5000F обраний через його здатність справлятися з великим обсягом трафіку, використовуючи інтелектуальні алгоритми розподілу запитів. Його можливість автоматично перенаправляти трафік до найменш завантажених серверів забезпечує стабільність навіть за високих навантажень. Це рішення ідеально підходить для оптимізації роботи веб-серверів у великих корпоративних мережах.

3.9.3 Сегментація мережі через VLAN

Сегментація за допомогою VLAN забезпечує ізоляцію трафіку між групами користувачів або пристроїв, що підвищує безпеку та ефективність роботи мережі.

Рисунок 3.41 - MikroTik CRS518-16XS-2XQ-RM

Таблиця 3.3 - Характеристики MikroTik CRS518-16XS-2XQ-RM

MikroTik CRS518-16XS-2XQ-RM	**Порти:** 16 × 25G SFP28, 2 × 100G QSFP28. **Операційна система:** RouterOS/SwitchOS із підтримкою VLAN. **Додаткові функції:** Гаряча заміна компонентів для зменшення часу простою.

MikroTik CRS518 вирізняється своїми розширеними можливостями у створенні VLAN. Його багатопортова архітектура дає змогу налаштовувати ізольовані підмережі для різних підрозділів компанії, таких як бухгалтерія, IT-відділ чи адміністрація. Завдяки цьому забезпечується не лише підвищена безпека даних, але й ефективніше використання мережевих ресурсів.

3.9.4 Блок-схема оптимізованої мережі

На Рисунку 3.41 представлено схему оптимізованої локальної мережі, побудованої з використанням сучасного обладнання MikroTik та Fortinet. Конструкція мережі забезпечує ефективне управління трафіком, високий рівень масштабованості, безпеку та стабільність навіть під високими навантаженнями.

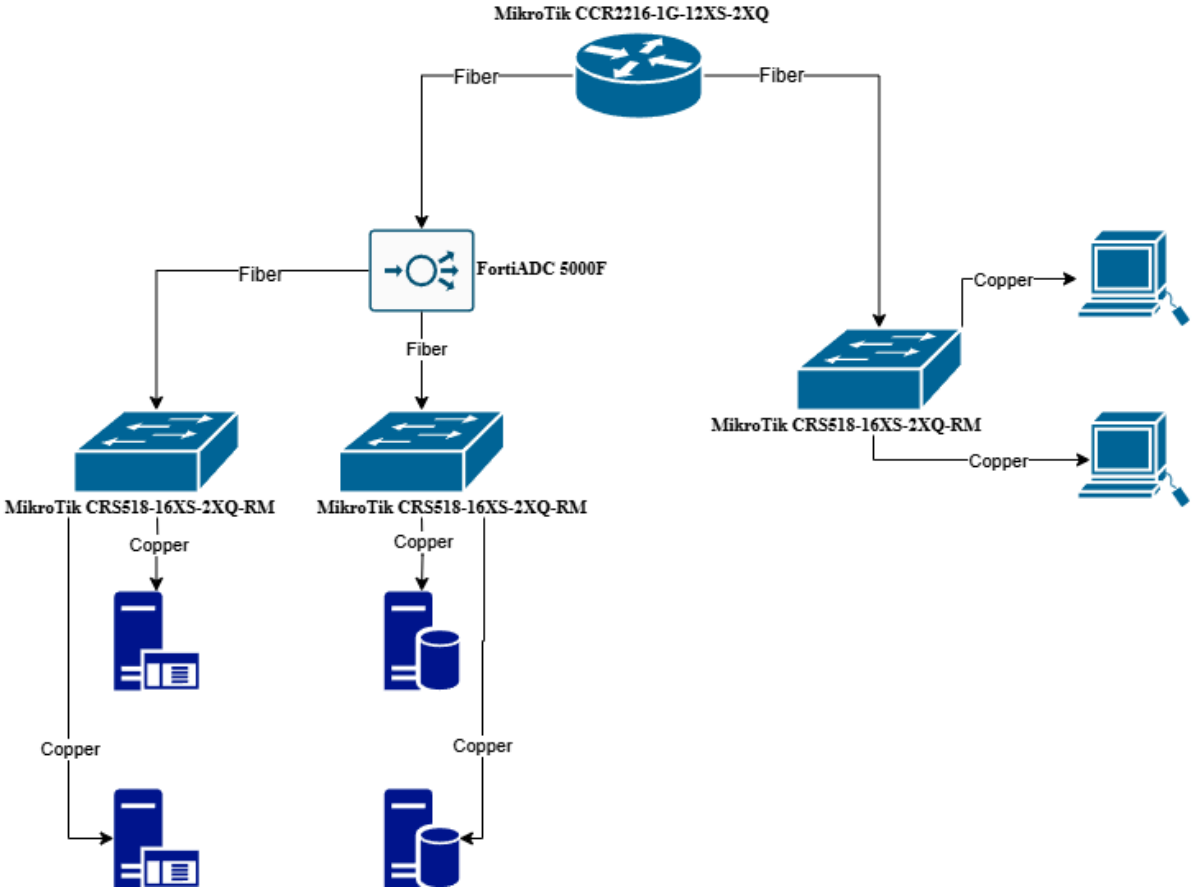


Рисунок 3.42 - Блок-схема оптимізованої мережі

Опис елементів блок-схеми

— **Клієнтські пристрої:** Користувачі мережі (ПК, ноутбуки, IP-телефони) підключені до комутатора доступу MikroTik CRS518 через високошвидкісні порти 25 Gigabit SFP28. Використання VLAN дозволяє сегментувати трафік, відокремлюючи, наприклад, трафік бухгалтерії, IT-відділу та інших підрозділів.

— **Комутатори доступу (MikroTik CRS518-16XS-2XQ-RM)**: Ці комутатори забезпечують фізичне підключення кінцевих пристроїв до мережі, ізолюючи їхній трафік за допомогою VLAN. Додатково комутатори передають сегментований трафік до маршрутизатора для подальшої обробки.

— **Маршрутизатор (MikroTik CCR2216-1G-12XS-2XQ)**: Маршрутизатор виконує функції управління трафіком, використовуючи QoS для пріоритизації важливих запитів, таких як відеоконференції чи VoIP. Це гарантує стабільність

навіть за умов значного навантаження. Він також спрямовує дані до балансувальника навантаження.

— **Балансувальник навантаження (FortiADC 5000F)**: FortiADC відповідає за рівномірний розподіл запитів між внутрішніми серверами, такими як веб-сервери, файлові сховища чи бази даних. Завдяки використанню інтелектуальних алгоритмів розподілу (наприклад, Round Robin) балансувальник підтримує стабільний час відповіді, зменшуючи затримки.

— **Внутрішні сервери**: Сервери, що обслуговують запити, розташовані в локальній мережі та об’єднані через високошвидкісні комутатори доступу. Балансувальник забезпечує рівномірне завантаження серверів, підвищуючи їхню продуктивність.

— **Інтернет-шлюз**: З’єднує внутрішню мережу із зовнішнім світом через маршрутизатор MikroTik CCR2216. Це забезпечує захищений і оптимізований доступ до глобальної мережі.

Оптимізована мережа демонструє високу гнучкість та надійність завдяки інтеграції VLAN, QoS, балансування навантаження та використанню сучасного обладнання. Така архітектура дозволяє ефективно обслуговувати навіть великі корпоративні структури, забезпечуючи стабільність і продуктивність усіх компонентів мережі.

ВИСНОВКИ

У ході виконання магістерської роботи була досягнута мета дослідження — оптимізація мережевої інфраструктури для забезпечення стабільної роботи під високими навантаженнями. Виконаний аналіз теоретичних основ, сучасних методів та технологій управління трафіком і балансування навантаження дозволив отримати нові знання про способи підвищення ефективності мережевих систем.

У роботі було вирішено наступні завдання:

1. Проведено детальний аналіз сучасних методів управління трафіком, включаючи QoS, кешування та використання ML-алгоритмів, що дозволило ідентифікувати найефективніші підходи для оптимізації.
2. Досліджено технології балансування навантаження із застосуванням як традиційних алгоритмів, так і програмно-визначених мереж (SDN). Це забезпечило можливість динамічного управління трафіком і зниження затримок.
3. Розроблено і протестовано експериментальну модель, яка демонструє переваги використання Load Balancer, CloudFront та інших сучасних інструментів.
4. Визначено практичні рекомендації для впровадження технологій оптимізації мережевих інфраструктур, які можуть бути використані в реальних умовах, зокрема в корпоративних середовищах та дата-центрах.

Результати тестування підтвердили, що використання SDN, кешування, балансування навантаження та автоматичного масштабування дозволяє значно зменшити затримки, покращити стабільність роботи системи і підвищити ефективність використання ресурсів.

Дослідження також відзначило важливість інтеграції хмарних платформ, таких як AWS, для забезпечення гнучкості та масштабованості мережевих систем. Реалізація запропонованих рішень може слугувати основою для подальших досліджень у галузі мережевих технологій.

Таким чином, виконане дослідження робить значний внесок у розвиток підходів до оптимізації мережевої інфраструктури та може бути використане як основа для впровадження ефективних рішень у реальних умовах високих навантажень.

ПЕРЕЛІК ДЖЕРЕЛ

1. Access, Distribution, and Core Layers Explained. *ComputerNetworkingNotes*. URL: https://www.computernetworkingnotes.com/ccna-study-guide/access-distribution-and-core-layers-explained.html.
2. Tasmiha Khan, Jackson G., Michael Goodwin. What Is Network Topology? | IBM. *IBM - United States*. URL: https://www.ibm.com/topics/network-topology.
3. 11 Types of Networks Explained: VPN, LAN & More - blog Planet VPN. *Planet VPN*. URL: https://freevpnplanet.com/blog/11-types-of-networks-explained-vpn-lan-more/.
4. Аналіз впливу затримки в мережі на продуктивність синхронних та асинхр. *peerdh.com*. URL: https://peerdh.com/uk/blogs/programming-insights/analyzing-the-impact-of-network-latency-on-synchronous-and-asynchronous-web-application-performance-2?utm_source.
5. Gillis A. S. What is packet loss and how do you fix it?. *Search Networking*. URL: https://www.techtarget.com/searchnetworking/definition/packet-loss.
6. What is deep packet inspection (DPI)? | fortinet. *Fortinet*. URL: https://www.fortinet.com/resources/cyberglossary/dpi-deep-packet-inspection.
7. GeeksforGeeks. Computer Network | Quality of Service and Multimedia - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/computer-network-quality-of-service-and-multimedia/.
8. IBM. What Is Software-Defined Networking (SDN)? | IBM. *IBM - United States*. URL: https://www.ibm.com/topics/sdn.
9. What is Network Function Virtualization (NFV)?. *Ciena Home | Ciena - Ciena*. URL: https://www.ciena.com/insights/articles/What-is-NFV-prx.html.

10. What is Secure SD-WAN (Software-Defined Wide Area Network)? | Fortinet. *Fortinet*. URL: https://www.fortinet.com/products/sd-wan.
11. What is Caching and How it Works | AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/caching/.
12. Zhiyong Xu, Laxmi Bhuyan, Y. Hu. Exploiting client cache: a scalable and efficient approach to build large Web cache. *18th International Parallel and Distributed Processing Symposium*, м. Santa Fe, 26–30 квіт. 2004 p. 2004.
13. GeeksforGeeks. Difference between K-Means and DBScan Clustering - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/difference-between-k-means-and-dbscan-clustering/.
14. Load Balancing Algorithms, Types and Techniques - Kemp. *Load Balancer For Always-On Application Experience - Kemp*. URL: https://kemptechnologies.com/load-balancer/load-balancing-algorithms-techniques.
15. Installing POX – POX Manual Current documentation. *Site not found · GitHub Pages*. URL: https://noxrepo.github.io/pox-doc/html/.
16. Ryu SDN Framework. *Ryu SDN Framework*. URL: https://ryu-sdn.org/.
17. Shavan Askar, Faris Keti. Performance Evaluation of Different SDN Controllers: A Review. *nternational Journal ofScience and Business*. 2021. Т. 5, № 6. С. 14.
18. ipoque | Shaping app traffic with DPI and application delivery… *ipoque*. URL: https://www.ipoque.com/blog/application-delivery-controller.
19. Scale Up vs Scale Out. *Portworx*. URL: https://portworx.com/blog/scale-up-vs-scale-out/.
20. Cloud Services - Build and Scale Securly - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/products/?aws-products-all.sort-by=item.additionalFields.productNameLowercase&aws-products-all.sort-order=asc&awsf.re:Invent=*all&awsf.Free%20Tier%20Type=*all&awsf.tech-category=*all.
21. Google Kubernetes Engine (GKE). *Google Cloud*. URL: https://cloud.google.com/kubernetes-engine.

22. Managed Kubernetes - Amazon Elastic Kubernetes Service (EKS) - AWS. *Amazon Web Services, Inc.* URL: https://aws.amazon.com/eks/.
23. Spotify Case Study. *Kubernetes*. URL: https://kubernetes.io/case-studies/spotify/.
24. Installation – Gunicorn 23.0.0 documentation. *Gunicorn - WSGI server – Gunicorn 23.0.0 documentation*. URL: https://docs.gunicorn.org/en/latest/install.html.
25. About Certbot. *Certbot*. URL: https://certbot.eff.org/pages/about.

Додаток А.

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Система управління мережевою інфраструктурою для підтримки високих навантажень

Виконав:
Сараненко А.Д
Науковий керівник:
Лащевська Н.О.

2025

Актуальність теми

- Зростання обсягів даних та кількості користувачів створює навантаження на мережі.
- Забезпечення стабільності та надійності мереж є критично важливим для бізнесу.
- Оптимізація мережевої інфраструктури дозволяє підвищити продуктивність та якість обслуговування.

Мета та завдання дослідження

Мета:

Оптимізація мережевої інфраструктури для забезпечення стабільної роботи під високими навантаженнями.

Завдання:

Аналіз сучасних методів управління трафіком.

Дослідження методів балансування навантаження.

Розробка і тестування експериментальної моделі.

Основні проблеми мережевих інфраструктур

- Затримки в передачі даних.
- Втрата пакетів через перевантаження.
- Обмежена масштабованість.
- Потреба в динамічному управлінні ресурсами.

Методи оптимізації мереж

- Балансування навантаження (Round Robin, Least Connections).
- Управління трафіком через QoS.
- Використання кешування (локальне, серверне, розподілене).
- Інтеграція хмарних рішень для масштабування.

Рисунок 1. - Методи оптимізації мережі

Практична частина: Налаштування хмарного середовища

- Налаштування тестового середовища на AWS.
- Використання Load Balancer для розподілу трафіку.
- Інтеграція з CloudFront для зменшення затримок.
- Використання інструментів, таких як Apache JMeter, для тестування.

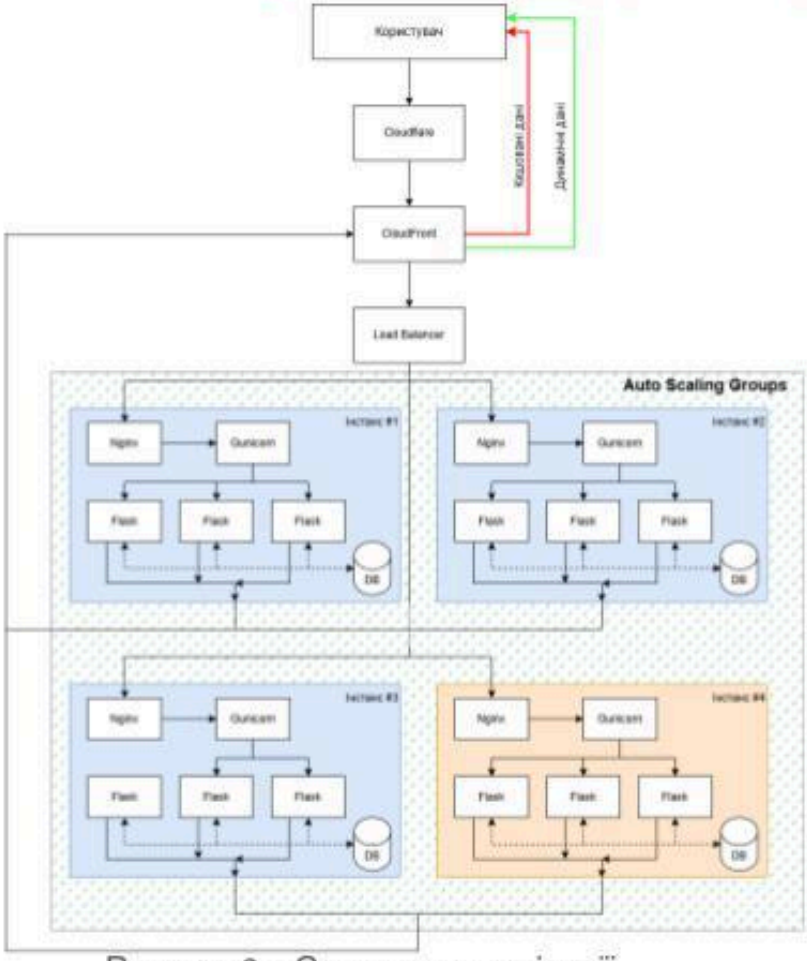


Рисунок 2. - Система оптимізації

Практична частина: Налаштування веб застосунку

- **Flask** — фреймворк для створення серверної частини застосунку.
- **Gunicorn** — сервер WSGI для розгортання застосунку.
- **Nginx** — веб-сервер для маршрутизації запитів та підвищення продуктивності.
- **SQLite** — вбудована реляційна база даних для зберігання та обробки даних.

```
 GNU nano 7.2                                                    /etc/systemd/s
[Unit]
Description=Gunicorn instance to serve my Flask app
After=network.target

[Service]
User=ubuntu
Group=www-data
WorkingDirectory=/home/ubuntu/myflaskapp
Environment="PATH=/home/ubuntu/myflaskapp/myenv/bin"
ExecStart=/home/ubuntu/myflaskapp/myenv/bin/gunicorn --workers 3 --bind 127.0.0.1:8000 app:app

[Install]
WantedBy=multi-user.target
```

Рисунок 3. - Налаштування Gunicorn

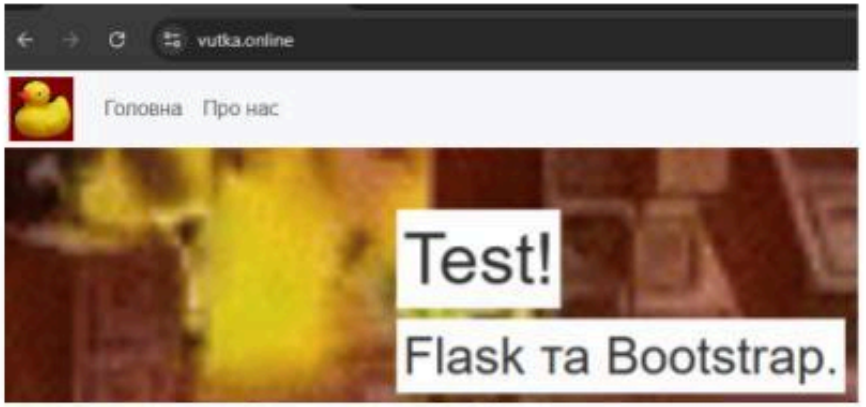


Рисунок 4. - Веб застосунок

Результати тестування

- Порівняння продуктивності системи без оптимізації та з використанням Load Balancer.
- Зменшення затримок та покращення часу відповіді серверів.
- Підвищення стійкості до високих навантажень.

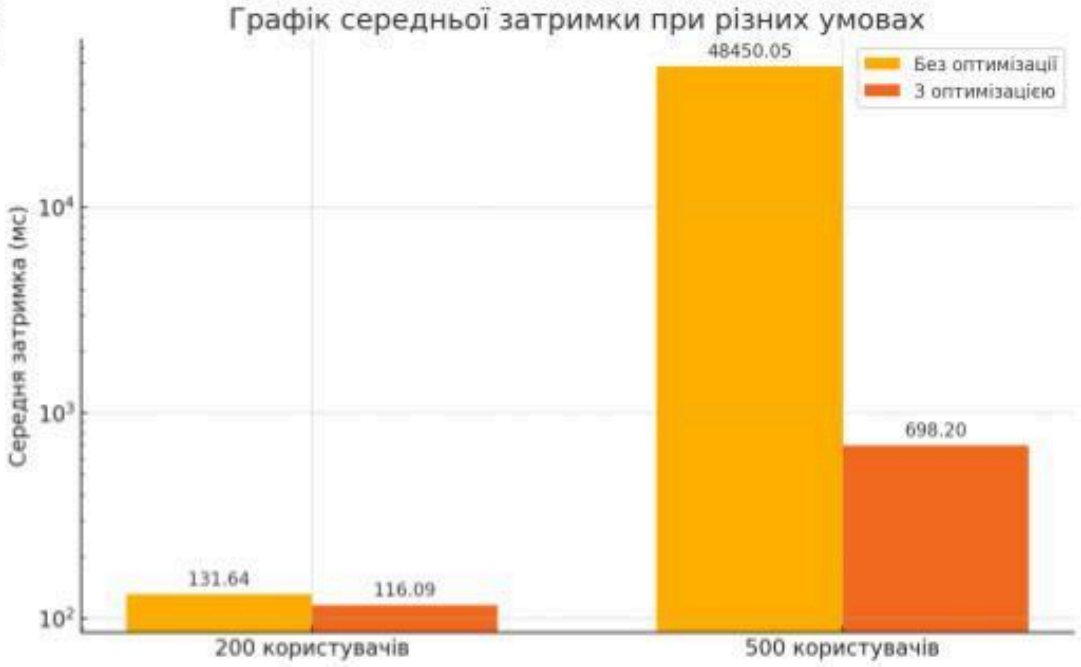


Рисунок 5. - Порівняльний графік оптимізованої та не оптимізованої системи

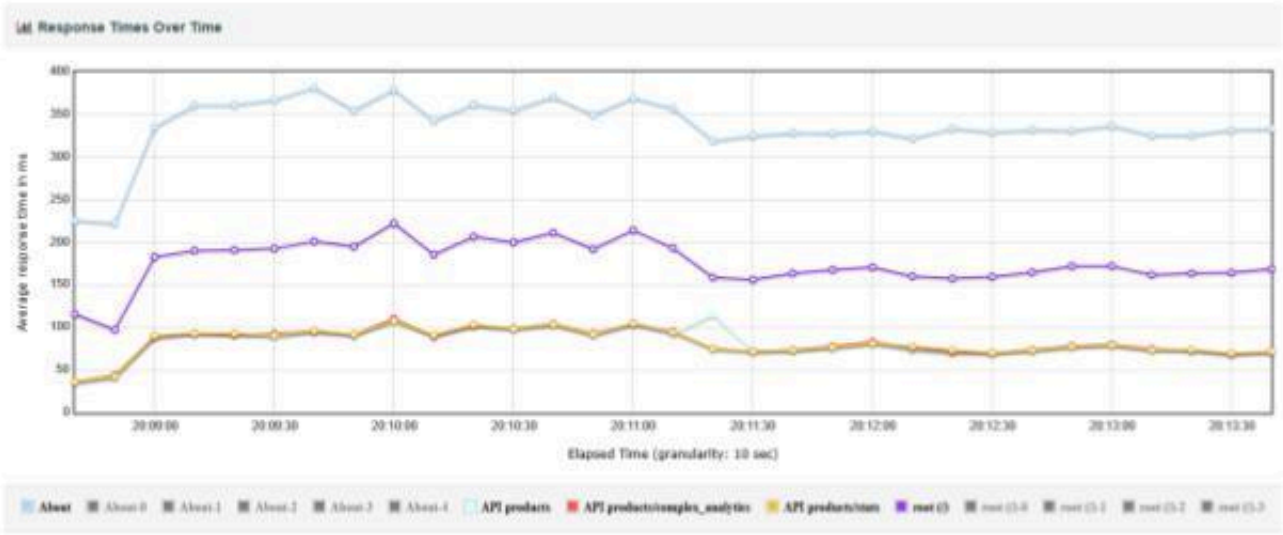


Рисунок 6. - Результат тестування не оптимізованої системи - 200 користувачів

Система без оптимізації - 200 користувачів

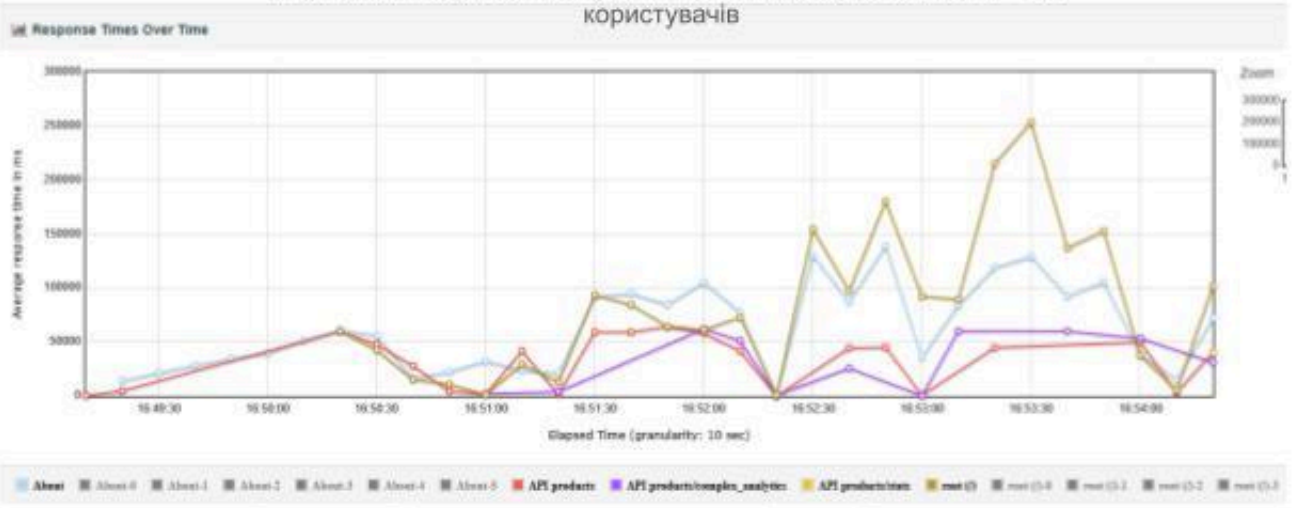


Рисунок 7. - Результат тестування не оптимізованої системи - 500 користувачів

Система без оптимізації - 500 користувачів

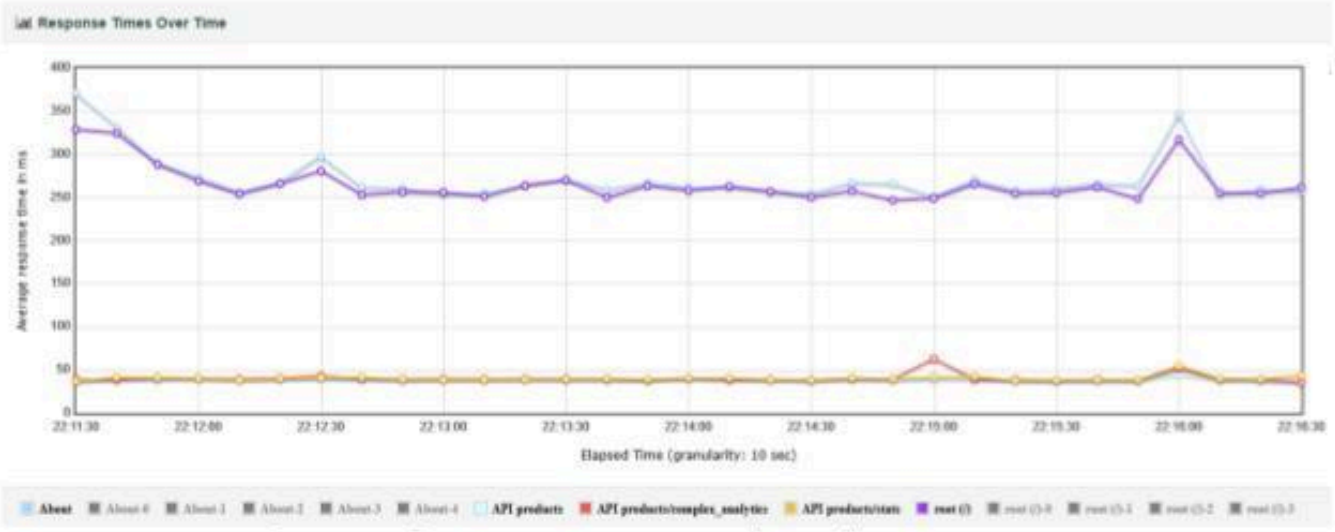


Рисунок 8. - Результат тестування оптимізованої системи - 200 користувачів

Оптимізована система - 200 користувачів

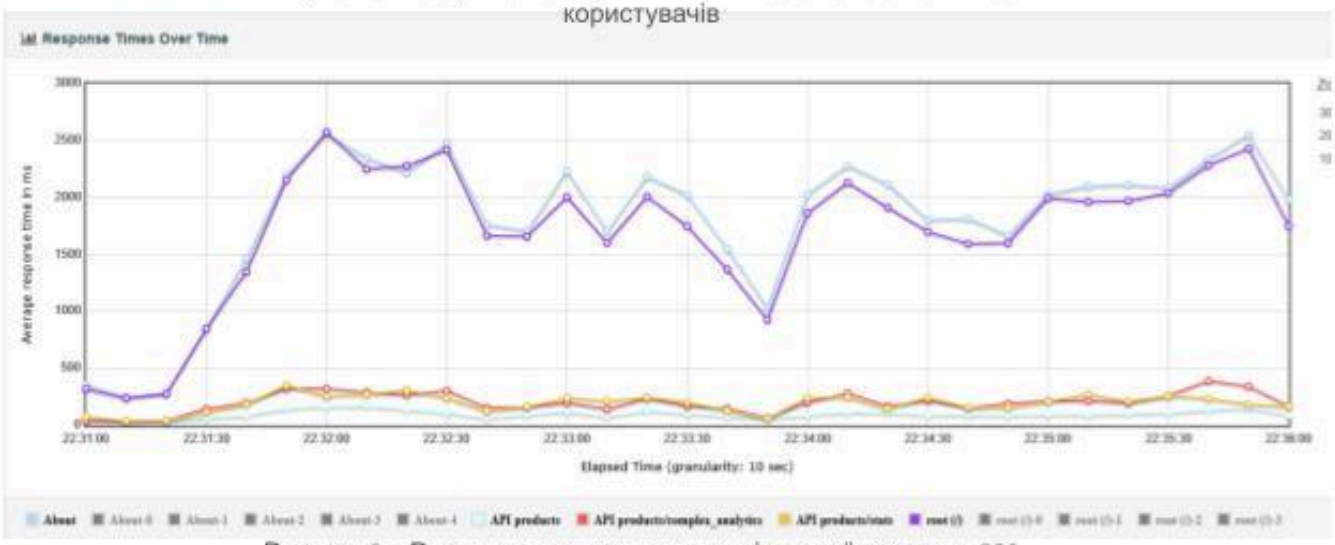


Рисунок 9. - Результат тестування оптимізованої системи - 500 користувачів

Оптимізована система - 500 користувачів

Висновки

- Оптимізація мережевої інфраструктури дозволяє підвищити її продуктивність і надійність.
- Методи, такі як кешування, Load Balancer та QoS, довели свою ефективність.
- Подальший розвиток пов'язаний із впровадженням SDN та алгоритмів машинного навчання.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Система управління мережевою інфраструктурою для підтримки високих навантажень

Виконав:
Сараненко А.Д
Науковий керівник:
Лащевська Н.О.

2025