Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота магістра

на тему: «Оптимізація архітектури зберігання логів у e-commerce компанії»

Виконав: студент групи К23-1М

Спеціальність 122 «Комп'ютерні науки»

Третяк М.Ю.
(прізвище та ініціали)

Керівник к.т.н. доц. Ульяновська Ю.В.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент_____
(місце роботи)

_____
(посада)

_____
(посада)

_____
(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Третяк М.Ю. Оптимізація архітектури зберігання логів у e-commerce компанії.

Дипломна робота (проект) на здобуття освітнього ступеня магістр за спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2024.

Об'єктом дослідження є системи зберігання логів у компаніях електронної комерції.

Предметом дослідження є технології та методи оптимізації витрат на зберігання логів у компаніях електронної комерції та їх програмна реалізація.

Метою роботи є зниження операційних витрат на зберігання логів у компаніях електронної комерції шляхом впровадження сучасних технологій та оптимізації архітектури логування.

Ця робота присвячена дослідженню методів оптимізації зберігання логів у системах електронної комерції. В роботі проаналізовано існуючі підходи до зберігання логів, визначено основні фактори, що впливають на витрати, і запропоновано ефективні рішення для їх мінімізації. Особлива увага приділена використанню пошукового рушія Amazon OpenSearch, який забезпечує високу масштабованість, швидкість обробки даних і інтеграцію з іншими сервісами. У роботі також розроблено алгоритм міграції даних з реляційної бази до пошукового рушія та розроблено архітектуру зберігання логів на основі сервісу Amazon OpenSearch. Проведено тестування впровадженого рішення. Отримані результати демонструють економічну ефективність обраного підходу, зниження витрат на зберігання та підвищення продуктивності системи логування.

Ключові слова: логування, Amazon OpenSearch, оптимізація витрат, електронна комерція, зберігання даних.

ABSTRACT

Tretiak M.Y. Optimization of the log storage architecture in the e-commerce company.

Diploma thesis (project) for the degree of Master's Degree in specialty 122 "Computer Science." - University of Customs and Finance, Dnipro, 2024.

The object of research is log storage systems in e-commerce companies.

The subject of the study is technologies and methods for optimizing log storage costs in e-commerce companies and their software implementation.

The aim of the work is to reduce operational costs of log storage in e-commerce companies by implementing modern technologies and optimizing logging architecture.

This work is dedicated to the study of methods for optimizing log storage in e-commerce systems. The work analyzes existing approaches to log storage, identifies key factors affecting costs, and proposes effective solutions to minimize them. Special attention is paid to the use of the Amazon OpenSearch engine, which provides high scalability, fast data processing, and integration with other services. The study also developed a data migration algorithm from a relational database to a search engine and developed a log storage architecture based on the Amazon OpenSearch service. The implemented solution was tested. The results demonstrate the economic efficiency of the chosen approach, cost reduction for storage, and improved performance of the logging system.

Keywords: logging, Amazon OpenSearch, cost optimization, e-commerce, data storage.

ЗМІСТ

ВСТУП

Логи в електронній комерції виконують важливу роль, забезпечуючи моніторинг системи та аналіз поведінки користувачів. У контексті сучасного розвитку цифрових технологій зростання обсягів даних, що генеруються у процесі функціонування електронних платформ, створює нові виклики для їхнього ефективного зберігання, обробки та аналізу.

Актуальність теми дослідження пов’язана з необхідністю зниження операційних витрат на зберігання логів у компаніях електронної комерції, що стикаються зі швидким зростанням обсягів даних. Традиційні підходи до зберігання логів, такі як реляційні бази даних, мають обмеження у масштабованості та продуктивності. У цьому контексті впровадження сучасних платформ є стратегічно обґрунтованим рішенням, яке дозволяє забезпечити високу ефективність аналізу даних при мінімізації витрат.

Враховуючи виклики, які постають перед бізнесом у сфері електронної комерції, дослідження спрямоване на оптимізацію процесу зберігання та обробки логів. Процес оптимізації включає аналіз сучасних підходів до логування, визначення ключових факторів витрат та розробку системи логування на основі пошукових рушіїв. Застосування пошукових рушіїв у контексті логування забезпечує масштабованість, швидкість пошуку, інтеграцію з іншими сервісами та економічну доцільність.

Новизна дослідження полягає у впровадженні підходу до оптимізації витрат на зберігання логів через використання пошукових рушіїв, зосередженні на здатності інтегрувати розподілені індексаційні структури для роботи з великими обсягами даних у реальному часі. Це відкриває нові можливості для оптимізації архітектур зберігання даних, забезпечуючи масштабованість і зниження витрат без втрати продуктивності та функціональності систем. Такий підхід є особливо корисним для компаній,

які шукають гнучкі та економічно вигідні рішення для роботи з великими масивами логів.

Метою дослідження є оптимізація системи логування та зниження операційних витрат на зберігання логів у компанії електронної комерції.

Для досягнення поставленої мети в кваліфікаційній роботі ставились та вирішувались наступні завдання дослідження:

1. Провести аналіз існуючих підходів до зберігання логів у системах електронної комерції.
2. Оцінити основні фактори, що впливають на вартість зберігання логів.
3. Розробити алгоритм переносу існуючих даних з реляційної бази даних до сервісу Amazon OpenSearch.
4. Розробити систему логування на основі сервісу Amazon OpenSearch.
5. Провести порівняльний аналіз витрат до та після впровадження системи логування.

Методи дослідження: В роботі використовувалися методи та технології обробки великих даних, пошуковий рушій Amazon OpenSearch, брокер повідомлень RabbitMQ, реляційна база даних MySQL, мова програмування PHP.

Об'єкт дослідження: Системи зберігання логів у компаніях електронної комерції.

Предмет дослідження: Технології та методи оптимізації витрат на зберігання логів у компаніях електронної комерції та їх програмна реалізація.

Практичне значення отриманих результатів: Можливість впровадження розроблених підходів у діяльність компаній електронної комерції, що дозволяє знизити операційні витрати та підвищити ефективність аналізу логів.

Структура роботи:

Розділ 1 Дослідження предметної області та постановка задачі дослідження. В даному розділі буде виконано пошук та аналіз існуючих підходів до зберігання логів.

Розділ 2 Аналіз засобів реалізації архітектури зберігання логів. В даному розділі буде проаналізовано та обрано кращу систему зберігання логів для компаній електронної комерції.

Розділ 3 Оптимізація архітектури логування e-commerce компанії. В даному розділі буде спроектовано, розроблено та протестовано систему зберігання логів.

Робота складається зі вступу, 3-х розділів, висновків, списку використаної літератури з 20 джерел. Обсяг роботи 71 сторінка, 12 рисунків та 5 формул.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ

1.1. Аналіз публікацій щодо використання логів в електронній комерції

У світі електронної комерції логи виконують ключову роль, забезпечуючи функціонування систем, аналіз поведінки користувачів та підвищення безпеки. Логи класифікуються за змістом інформації та джерелами їх формування. Системні логи відображають стан інфраструктури, такі як навантаження на сервери чи збої в роботі операційних систем. Логи додатків фіксують внутрішні процеси, включаючи обробку замовлень та взаємодію з API. Транзакційні логи містять дані про фінансові операції, а логи безпеки контролюють спроби несанкціонованого доступу. Користувацькі логи дозволяють аналізувати дії клієнтів, наприклад, перегляди товарів або оформлення покупок.

З точки зору джерел логів, клієнтські логи генеруються на пристроях користувачів, таких як браузери чи мобільні додатки, тоді як серверні логи створюються на серверах, які обслуговують ці запити. Дані можуть зберігатися як у сирому вигляді, так і у вигляді агрегованих логів, що спрощують доступ до важливих деталей. Ці дані стають основою для прийняття рішень, наприклад, виявлення аномалій у роботі системи або оптимізації маркетингових кампаній.

Сучасні підходи до обробки логів зосереджені на використанні спеціалізованих пошукових рушіїв, таких як Amazon OpenSearch, Elasticsearch та Splunk. Вони забезпечують високу швидкість обробки даних завдяки використанню інвертованих індексів та ефективним методам масштабування [4, 6, 7, 8]. Наприклад, Elasticsearch дозволяє проводити складні аналітичні запити, що робить його особливо корисним у контексті електронної комерції [10]. Amazon OpenSearch, своєю чергою, пропонує

зручну інтеграцію з іншими хмарними сервісами AWS, що підтверджено в дослідженнях щодо його продуктивності [1, 12, 16].

Особливої уваги заслуговують хмарні технології, такі як Amazon S3 і Google Cloud Storage. Вони забезпечують надійне та масштабоване зберігання даних, знижуючи витрати на підтримку локальної інфраструктури та підвищуючи гнучкість бізнесу [1, 12]. Використання хмарних рішень дозволяє легко впроваджувати політики життєвого циклу даних, які автоматизують переміщення логів між гарячими, теплими та холодними сховищами, оптимізуючи використання ресурсів [13, 19]. Також інтеграція з Kibana та OpenSearch Dashboards дозволяє виконувати візуалізацію даних і створювати дашборди для реального моніторингу ключових показників [6].

Ефективне використання логів сприяє моніторингу та діагностиці систем, аналізу поведінки клієнтів і покращенню користувацького досвіду. Наприклад, аналіз логів дозволяє виявляти популярні товари, оптимізувати пошукові запити на сайті та підвищувати конверсію, як це показано в дослідженні з використання Elasticsearch для покращення релевантності пошукових результатів [6]. Крім того, завдяки технологіям потокової обробки, таким як Apache Kafka, можна забезпечувати обробку даних у реальному часі, що є критично важливим для швидкої реакції на зміни у системах [5, 15].

Загалом, зростаючі обсяги даних створюють нові виклики для архітектури систем логування, зокрема у масштабованості, продуктивності та економічності. Оптимізація зберігання та обробки логів залишається ключовим завданням для бізнесів, які прагнуть забезпечити стабільність і ефективність своєї діяльності. Використання сучасних платформ, таких як Amazon OpenSearch, дозволяє досягти значного зниження витрат та підвищення продуктивності, що підтверджується численними дослідженнями [7, 11, 16].

1.2. Особливості обробки логів у високонавантажених системах

Високонавантажені системи (High-Load Systems) є критично важливими для багатьох галузей, у тому числі галузей, що пов’язані з електронною комерцією, фінансами, телекомунікаціями та онлайн-розвагами. У таких системах щодня генеруються великі обсяги логів, які містять ключову інформацію про стан системи, транзакції та взаємодію користувачів із сервісами. Враховуючи обсяг і швидкість генерації даних, обробка та зберігання логів у таких умовах потребують спеціалізованих підходів і технологій.

Основні проблеми обробки та зберігання логів у високонавантажених системах:

1) Сучасні високонавантажені системи генерують мільйони записів кожну годину. Наприклад, в e-commerce платформах кожен клік, перегляд товару чи оформлення замовлення створює окремий лог. Традиційні методи зберігання можуть бути неефективними, що вимагає застосування розподілених сховищ і спеціалізованих алгоритмів для роботи з великими обсягами даних.

2) Логи повинні оброблятися в реальному часі, особливо якщо вони використовуються для моніторингу системи чи виявлення аномалій. Наприклад, у разі кібератаки оперативний аналіз логів може допомогти запобігти серйозним наслідкам. Це потребує використання технологій потокової обробки, наприклад, Apache Kafka, Flink або Spark Streaming.

3) Системи повинні адаптуватися до збільшення обсягів даних. Це стосується не лише зберігання логів, а й забезпечення їхньої доступності для аналізу. Масштабованість досягається за допомогою хмарних сервісів або розподілених файлових систем, таких як HDFS чи Amazon S3 [1].

4) Високонавантажені системи повинні забезпечувати безперервний доступ до логів навіть у разі збоїв. Це вимагає застосування методів реплікації даних та резервного копіювання.

Технічні підходи до обробки логів у високонавантажених системах:

1) Для обробки великих обсягів даних широко використовуються розподілені системи, такі як Elasticsearch, Cassandra чи ClickHouse. Ці платформи дозволяють зберігати дані у кількох вузлах, забезпечуючи їхню доступність і швидкість пошуку.

2) Щоб полегшити подальшу обробку, логи зазвичай зберігаються у структурованому вигляді. Для цього використовуються формати JSON, Parquet чи Avro, які забезпечують гнучкість і ефективність стиснення даних.

3) Для роботи з логами в режимі реального часу використовуються технології потокової обробки даних. Наприклад, Apache Kafka дозволяє збирати дані з різних джерел і передавати їх до систем аналізу.

4) З метою економії місця часто використовуються алгоритми стиснення, такі як Gzip чи Snappy, а також методи дедуплікації для усунення повторюваних записів.

5) У високонавантажених системах важливо визначати, які логи зберігаються довготривало, а які – видаляються після певного часу. Це досягається за допомогою політик Log Retention Policies, які автоматизують цей процес.

Особливості зберігання логів:

1) Хмарні рішення, такі як Amazon OpenSearch, Google BigQuery або Azure Data Lake, забезпечують гнучкість і масштабованість. Вони дозволяють обробляти дані у великих обсягах без необхідності підтримки локальної інфраструктури.

2) Для швидкого доступу до популярних записів використовується кешування, наприклад, Redis чи Memcached. Це значно знижує навантаження на основне сховище.

3) Ураховуючи конфіденційність інформації, що зберігається в логах, застосовуються механізми шифрування та обмеження доступу. Наприклад, сервіси можуть використовувати TLS-шифрування [2] для передавання даних і рольові моделі доступу до сховищ.

Розглянемо практичні приклади:

1) У великих онлайн-магазинах, таких як Amazon, логи використовуються для відстеження транзакцій, аналізу поведінки користувачів і персоналізації рекомендацій. Для цього застосовуються розподілені системи зберігання, інтегровані з потоковою обробкою даних.

2) У фінансових додатках та банках логи використовуються для виявлення шахрайства та забезпечення відповідності регуляторним вимогам. Такі системи потребують високої точності та швидкості обробки.

3) Соціальні мережі, наприклад, Facebook чи Twitter щосекунди генерують велику кількість логів, які аналізуються для покращення алгоритмів ранжування контенту та виявлення спаму.

Обробка та зберігання логів у високонавантажених системах є складним завданням, яке вимагає використання спеціалізованих підходів і технологій. Ефективна робота з логами дозволяє підтримувати стабільність системи та забезпечувати її подальший розвиток, підвищуючи продуктивність, надійність і безпеку. Успішна реалізація таких процесів можлива завдяки розподіленим сховищам, потоковій обробці та хмарним рішенням, які здатні адаптуватися до постійно зростаючих потреб бізнесу. Для зберігання великих обсягів логів часто використовуються розподілені сховища, які забезпечують масштабованість і надійність. Потокова обробка даних дозволяє аналізувати логи в режимі реального часу, швидко реагуючи на аномалії або збої. Хмарні платформи надають можливість автоматично масштабувати інфраструктуру, адаптуючись до потреб системи, і спрощують управління логами завдяки вбудованим інструментам.

1.3. Фактори, що впливають на вартість зберігання логів у e-commerce

Логи у сфері електронної комерції є важливим інструментом для забезпечення стабільності бізнесу, моніторингу систем, аналітики та виявлення проблем. Однак постійне збільшення обсягів даних спричиняє суттєві витрати на їх зберігання. Щоб оптимізувати ці витрати без шкоди для якості та доступності інформації, e-commerce компанії повинні розуміти ключові фактори, які впливають на формування вартості зберігання даних.

Розглянемо ключові фактори, що визначають вартість зберігання логів:

1) Обсяг даних безпосередньо впливає на витрати. Електронна комерція генерує великі обсяги даних через численні взаємодії користувачів: перегляди сторінок, кліки, транзакції. Додатково обсяг залежить від тривалості зберігання даних, яка може визначатися внутрішніми політиками компанії або законодавчими вимогами.

2) Оптимізація формату зберігання, наприклад, використання Parquet або Avro, може значно зменшити обсяг даних порівняно з неструктурованими форматами. Водночас додавання метаданих для зручності аналізу збільшує обсяг, а отже, і вартість.

3) Локальні сервери підходять для невеликих компаній, але потребують значних витрат на інфраструктуру та обслуговування при масштабуванні. Хмарні сховища, такі як Amazon S3 чи Google Cloud Storage, пропонують масштабованість і гнучкість, однак можуть бути дорогими при зберіганні великих обсягів або частому доступі до даних [1].

4) Часто використовувані логи зберігаються у високошвидкісних системах (наприклад, SSD), що дорожче, ніж архіви для рідкісного доступу, такі як AWS Glacier чи Google Coldline Storage.

5) Стиснення логів за допомогою алгоритмів Gzip, Snappy чи Zstandard дозволяє значно скоротити їх обсяг. Уникнення дублювання даних через дедуплікацію також суттєво знижує витрати.

6) Open-source рішення, наприклад, ELK Stack [3], знижують початкові витрати, але вимагають ресурсів на налаштування та підтримку. Хмарні платформи пропонують гнучкі тарифи, але вимагають ретельного моніторингу витрат.

7) Автоматизація видалення старих логів або переміщення їх за допомогою політик життєвого циклу даних знижує витрати. Наприклад, архівування застарілих даних значно дешевше, ніж їх активне зберігання.

8) Регуляторні вимоги до зберігання логів протягом певного часу можуть значно збільшити витрати, особливо для глобальних платформ, що працюють у різних юрисдикціях.

9) Створення копій для забезпечення безперебійного доступу до даних збільшує вартість, оскільки потребує додаткового зберігання.

10) Обслуговування локальних серверів чи управління хмарними платформами вимагає кваліфікованого персоналу, що впливає на загальні витрати.

Пошукові движки, такі як OpenSearch, Elasticsearch чи Solr, є більш економічно вигідними для збереження логів, ніж реляційні бази даних, через їхню архітектуру, що спеціалізована на обробці великих обсягів напівструктурованих даних [4]. Вони зберігають дані у форматі JSON, що дає змогу працювати зі змінними структурами без необхідності підтримувати жорстку схему як в реляційних базах даних. Це суттєво спрощує роботи з логами у сховищі.

Пошукові движки здатні виконувати швидкий пошук по документам завдяки використанню інвертованого індексу [5]. Цей механізм ефективно опрацьовує текстові дані та часові ряди, що є основою більшості логів. У реляційних базах для досягнення подібної продуктивності необхідно створювати додаткові індекси, що збільшує витрати на зберігання і запис.

Масштабованість є ще однією перевагою пошукових движків. Їхня розподілена архітектура дозволяє легко додавати нові вузли для обробки великих обсягів даних, тоді як реляційні бази потребують складних

механізмів для синхронізації, що підвищує експлуатаційні витрати. Крім того, у пошукових движках реалізовані автоматизовані механізми управління життєвим циклом даних. Наприклад, старі логи можна автоматично видаляти або переносити в дешевші сховища, тоді як у реляційних базах такі задачі вимагають ручного налаштування.

У фінансовому аспекті пошукові движки мають значну перевагу. Вони забезпечують високу швидкість агрегацій і аналізу логів із мінімальними затратами на апаратні ресурси. Інструменти типу OpenSearch, часто є безкоштовними або доступними за умовно низькою ціною для компанії, тоді як багато комерційних реляційних баз даних мають суттєві витрати на ліцензії.

Для зниження витрат компанії можуть впроваджувати різні стратегії. Зокрема, автоматизація процесу видалення старих логів дозволяє зменшити обсяг даних, що зберігаються. Використання ефективних форматів даних також сприяє оптимізації простору, необхідного для зберігання. Часто використовувані логи доцільно зберігати у швидкодоступних сховищах, тоді як архівні дані можна розміщувати у холодних архівах. Зменшення розміру файлів досягається шляхом застосування компресії та дедуплікації. Перехід на хмарні рішення з гнучкими тарифами забезпечує додаткову економію, а використання спеціалізованих інструментів для моніторингу витрат і оптимізації ресурсів допомагає контролювати фінансові витрати.

Оптимізація витрат на зберігання логів є критичною для компаній у сфері електронної комерції, оскільки вона дозволяє забезпечити стабільність бізнес-процесів і високу якість послуг без значного фінансового навантаження. Розумне використання сучасних технологій, ефективних форматів даних і автоматизованих політик життєвого циклу даних дозволяє досягти балансу між витратами та ефективністю [6].

1.4. Висновок до першого розділу

У першому розділі було детально розглянуто сучасні архітектурні підходи до зберігання логів, їхні ключові особливості та вимоги, а також фактори, що впливають на операційні витрати. Аналіз показав, що у сфері електронної комерції, де обсяг даних зростає зі значною швидкістю, ефективне зберігання логів є важливим аспектом забезпечення надійності, безпеки та аналітичної функціональності систем.

Централізоване зберігання логів дозволяє спростити управління даними та забезпечити швидкий доступ до них. Водночас цей підхід вимагає значних ресурсів для підтримки єдиного сховища, що може стати критичним у разі збою. Розподілені системи надають перевагу у масштабованості та надійності за рахунок дублювання даних, проте їхня реалізація є більш складною з точки зору налаштування та координації вузлів. Хмарні технології відкривають можливості для масштабованості та інтеграції з іншими сервісами, але створюють залежність від провайдерів і можуть генерувати значні витрати при великому обсязі даних. Гібридні архітектури дозволяють поєднувати переваги локального та хмарного зберігання, забезпечуючи гнучкість і резервування.

Аналіз витрат на зберігання логів показав, що найбільший вплив на вартість мають обсяг даних, вибір технологій зберігання, частота доступу до логів, а також оптимізація структури та формату даних. Використання таких підходів, як компресія, дедуплікація та автоматизація життєвого циклу даних, може суттєво скоротити витрати. Проте для вибору оптимальної архітектури компанії мають враховувати свої бізнес-вимоги, технічні можливості та бюджет.

Отже, оптимізація зберігання логів є багатогранним завданням, яке потребує врахування різноманітних технічних, економічних та організаційних факторів. У контексті електронної комерції ефективне

управління логами сприяє зниженню витрат, покращенню аналітики та забезпеченню стабільності бізнес-процесів.

Метою роботи є зниження операційних витрат на зберігання логів у компанії електронної комерції шляхом переходу з реляційних баз даних на сервіс Amazon OpenSearch.

Для досягнення поставленої мети в дипломній роботі ставились та вирішувались наступні завдання дослідження:

1. Провести аналіз існуючих підходів до зберігання логів у системах електронної комерції, зокрема реляційних баз даних і сучасних платформ, таких як OpenSearch.
2. Оцінити основні фактори, що впливають на вартість зберігання логів, включаючи обсяг даних, частоту доступу, компресію, місце зберігання, а також регуляторні вимоги.
3. Розробити алгоритм переходу з реляційної бази даних на OpenSearch, враховуючи аспекти міграції даних, оптимізації структури логів і налаштування платформи для мінімізації витрат.
4. Провести тестування OpenSearch як платформи для зберігання логів у реальних умовах, оцінюючи її продуктивність, масштабованість та економічну ефективність. Зробити порівняльний аналіз операційних витрат до та після оптимізації.

РОЗДІЛ 2. АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ АРХІТЕКТУРИ ЗБЕРІГАННЯ ЛОГІВ

2.1. Огляд сучасних архітектур логування

ELK Stack є потужним і гнучким інструментом для роботи з логами [3]. Його можливості охоплюють всі аспекти обробки даних: від збору і зберігання до аналізу та візуалізації (Рис. 2.1). Ця технологія має відкритий код та підходить для компаній різного масштабу, забезпечуючи інструменти для ефективного управління логами та моніторингу бізнес-процесів.

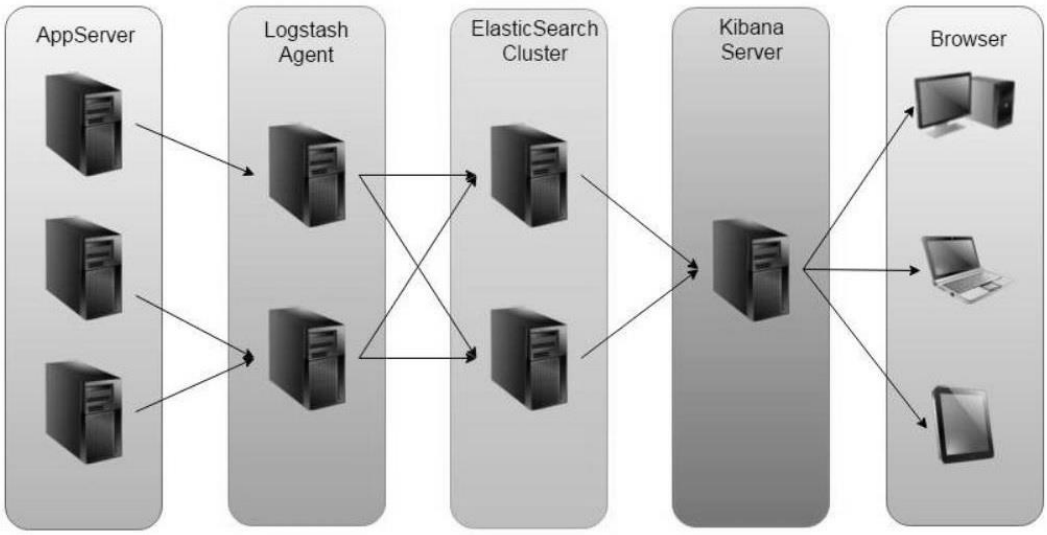


Рисунок 2.1 – Схема роботи ELK Stack

Elasticsearch є розподіленою пошуковою і аналітичною системою, яка лежить в основі ELK Stack. Вона забезпечує зберігання даних і швидкий доступ до них за допомогою повнотекстового пошуку.

Розглянемо основні особливості Elasticsearch [7]:

1) Система підтримує горизонтальне масштабування, що дозволяє зберігати і обробляти великі обсяги даних. Кластери Elasticsearch складаються з вузлів, які працюють разом, забезпечуючи високу доступність та продуктивність.

2) Дані в Elasticsearch організовуються у вигляді індексів, що дозволяє виконувати запити з використанням мов запитів DSL (Domain Specific Language). Завдяки цьому користувачі можуть швидко знаходити потрібну інформацію навіть у великих масивах логів.

3) Elasticsearch автоматично визначає типи даних, що дозволяє швидко інтегрувати систему з новими джерелами логів.

4) Elasticsearch надає REST API, який забезпечує інтеграцію з різними системами, що робить його гнучким і універсальним рішенням.

Logstash – це інструмент для збору, обробки і маршрутизації даних у ELK Stack. Він дозволяє отримувати дані з різних джерел, обробляти їх і передавати до Elasticsearch для зберігання [3].

Ключові особливості Logstash включають:

1) Logstash може працювати з різними форматами, такими як Syslog, файли логів, бази даних та хмарні сервіси. Це дозволяє централізувати управління логами в одній системі.

2) За допомогою спеціальних фільтрів Logstash може трансформувати дані перед їх передачею в Elasticsearch. Наприклад, видалення дублікатів, нормалізація форматів або додавання метаданих.

3) Logstash оптимізований для роботи у високонавантажених середовищах, що робить його надійним інструментом для збору великих обсягів даних.

4) Інструмент підтримує десятки плагінів для вводу, обробки та виводу даних, що дозволяє налаштовувати систему під конкретні потреби.

Kibana — це інтерфейс для візуалізації та аналізу даних, що зберігаються в Elasticsearch. Завдяки Kibana користувачі можуть створювати дашборди, проводити аналіз логів у реальному часі та отримувати інсайти з даних.

Особливості сервісу Kibana:

1) Інструмент має зручний веб-інтерфейс, який дозволяє створювати графіки, таблиці та інші візуалізації без потреби у програмуванні.

2) Kibana дозволяє моніторити потоки даних у реальному часі, що є критично важливим для моніторингу інфраструктури та бізнес-процесів.

3) Інтеграція з Elasticsearch забезпечує можливості пошуку та фільтрації даних. Користувачі можуть налаштовувати запити, які відповідають їхнім потребам.

4) Kibana дозволяє створювати кастомізовані дашборди для відображення ключових метрик і візуалізацій. Це робить інструмент ідеальним для відстеження показників продуктивності.

Різниця між Kibana та OpenSearch Dashboards полягає в їхньому походженні, ліцензійній політиці та орієнтації на певні екосистеми. Kibana є офіційним продуктом Elastic і входить до складу Elastic Stack (раніше ELK Stack). Його функціональність тісно інтегрована з іншими продуктами Elastic, такими як Elastic APM чи Elastic Security. Натомість OpenSearch Dashboards є форком Kibana, створеним у межах OpenSearch Suite після розділення проектів Elasticsearch і OpenSearch. Цей інструмент розробляється спільнотою на чолі з AWS і оптимізований для роботи з OpenSearch [8].

Функціонально обидва інструменти надають схожі можливості для створення візуалізацій, аналітичних дашбордів, аналізу часових рядів та налаштування попереджень. Однак Kibana пропонує низку розширених функцій, доступних лише в платних версіях Elastic Stack, таких як машинне навчання, інтеграція з картографічними даними та SIEM-рішення. OpenSearch Dashboards, у свою чергу, орієнтований на відкриті інновації та забезпечує нові можливості, зокрема ті, що спрямовані на потреби спільноти.

Kibana підтримує лише Elasticsearch і потребує узгодженості версій із цим движком. OpenSearch Dashboards інтегрується виключно з OpenSearch і є ідеальним вибором для тих, хто прагне використовувати рішення з відкритим кодом без ліцензійних обмежень. Таким чином, обидва інструменти виконують схожу роль, проте відрізняються підходом до ліцензування, екосистемною інтеграцією та спрямованістю розвитку.

ELK Stack має наступні переваги:

1) ELK Stack доступний безкоштовно, що робить його привабливим для компаній, які шукають доступні рішення.

2) Завдяки розподіленій архітектурі ELK Stack може обробляти великі обсяги даних, що актуально для компаній зі складною інфраструктурою.

3) Система підтримує широкий спектр джерел даних і може бути адаптована до будь-яких вимог.

4) Kibana забезпечує інструменти для аналізу та презентації даних у зрозумілому форматі.

Недоліки ELK Stack:

1) Робота з великими обсягами даних потребує значних обчислювальних ресурсів для підтримки продуктивності.

2) Для ефективного використання ELK Stack потрібно володіти спеціальними знаннями.

3) Відсутність вбудованого контролю доступу. Базова версія не включає деякі функції безпеки, які доступні лише у комерційних розширеннях.

Альтернативою ELK Stack може виступати технологія Splunk. Splunk — це потужна платформа для збору, індексації та візуалізації великих обсягів машинних даних у реальному часі. Розроблена з орієнтацією на бізнес, Splunk дозволяє користувачам моніторити, аналізувати та візуалізувати дані, що надходять із різноманітних джерел: серверів, додатків, мережевих пристроїв, баз даних та багатьох інших. Splunk здобула популярність завдяки своїй здатності працювати з великими обсягами даних і надавати корисні інсайти з них за допомогою інтуїтивно зрозумілих інтерфейсів [9].

Основна функція Splunk полягає в тому, щоб допомогти організаціям управляти своїми великими обсягами даних, що генеруються у вигляді логів. Платформа здатна збирати дані з широкого спектра джерел: серверів,

додатків, пристроїв IoT, мереж, баз даних тощо. Дані, зібрані Splunk, індексуються і стають доступними для пошуку та аналізу в реальному часі.

Splunk надає користувачам можливість будувати дашборди, генерувати звіти та автоматизувати процеси моніторингу, що дозволяє знижувати витрати часу на аналіз і покращувати реакцію на інциденти. Платформа надає також інструменти для побудови алертів, що допомагають оперативно реагувати на відхилення в системах, таких як аномалії у виконанні програм, мережеві загрози або збої в інфраструктурі.

Splunk пропонує зручний веб-інтерфейс, що дозволяє користувачам не тільки збирати та зберігати дані, а й створювати різноманітні візуалізації, такі як графіки, гістограми, діаграми та інтерактивні дашборди. Це дозволяє легко відслідковувати важливі метрики та швидко отримувати доступ до критично важливих даних.

Інтерфейс Splunk підтримує потужні можливості пошуку з використанням мови запитів SPL (Search Processing Language), яка дозволяє створювати складні запити для фільтрації та аналізу великих обсягів інформації. Цей функціонал значно спрощує процес пошуку потрібних даних серед величезних масивів логів і дозволяє отримувати важливу інформацію навіть за короткий проміжок часу.

Splunk підтримує горизонтальне масштабування, що дозволяє організаціям збільшувати потужність системи, додаючи нові сервери в кластер. Це важливо, коли мова йде про обробку та аналіз величезних обсягів даних, що генеруються великими підприємствами або в умовах високонавантажених середовищ. Важливим аспектом є те, що Splunk оптимізований для роботи з даними в реальному часі, що забезпечує оперативний доступ до актуальної інформації та дозволяє швидко реагувати на інциденти. Продуктивність залежить від налаштувань інфраструктури та оптимізації процесів індексації. У великих середовищах, де дані генеруються постійно, потрібно ефективно управляти кількістю індексів і запитів, щоб система могла працювати швидко і без затримок. Splunk має можливість

працювати з розподіленими кластерами, що дозволяє забезпечити високу доступність і відмовостійкість при обробці великих обсягів даних.

Однією з важливих функцій Splunk є можливості контролю доступу. Платформа дозволяє точно налаштувати права доступу до різних рівнів даних і функцій. Це дуже важливо для великих компаній, де потрібно забезпечити конфіденційність та безпеку даних, а також налаштувати доступ до певної інформації лише для авторизованих користувачів.

Ця технологія також пропонує вбудовані механізми для аудиту доступу та дій користувачів. Це дозволяє компаніям проводити моніторинг активності та забезпечувати відповідність різноманітним вимогам щодо безпеки. Однією з головних переваг є його універсальність. Платформа може працювати з величезними обсягами різноманітних даних і забезпечувати глибокий аналіз на всіх етапах роботи. Відразу після інтеграції Splunk надає користувачам можливість запускати запити для отримання важливої інформації, будувати візуалізації для аналізу трендів або виявлення аномалій.

Ще однією ключовою перевагою є здатність працювати з даними в реальному часі. Ця можливість надає компаніям миттєвий доступ до важливих даних, що дозволяє оперативно реагувати на зміни в системах або виникнення проблем. Завдяки цьому Splunk активно використовується для моніторингу інфраструктури, управління безпекою та оптимізації роботи додатків.

Проте, як і у будь-якої потужної платформи, у Splunk є деякі недоліки. Одним із них є висока вартість, особливо для великих організацій, де обсяг даних може бути дуже ліцензійне обмеження на кількість індексованих даних, що може призвести до значних витрат на зберігання та обробку інформації.

Після аналізу існуючих технологій було з'ясовано, що оптимальним рішенням для зберігання логів на даний момент є Amazon OpenSearch. Ця технологія пропонує унікальне поєднання продуктивності, гнучкості та безпеки для роботи з логами. Відкритий вихідний код дозволяє організаціям

уникнути додаткових ліцензійних витрат, що особливо важливо для підприємств, які обробляють великі обсяги даних. Завдяки високій швидкості пошуку, розподіленій архітектурі та можливостям аналітики OpenSearch є ідеальним рішенням для моніторингу та діагностики систем у реальному часі [8].

Використання OpenSearch як основної платформи для зберігання та аналізу логів дозволяє оптимізувати робочі процеси, знизити операційні витрати та забезпечити надійну роботу критично важливих систем. Вибір OpenSearch є стратегічно обґрунтованим кроком для будь-якого підприємства, що прагне підвищити ефективність обробки даних у сучасних умовах.

Розглянемо переваги OpenSearch для зберігання логів:

1) Сучасні системи генерують величезні обсяги логів, і їхнє зберігання вимагає масштабованих рішень. OpenSearch дозволяє обробляти та зберігати терабайти даних завдяки архітектурі, що базується на розподілених вузлах. Кожен вузол у кластері відповідає за частину даних, що забезпечує балансування навантаження та безперебійну роботу системи. Завдяки цьому логістику можна масштабувати горизонтально – додаючи нові вузли у разі зростання обсягу даних.

2) Зберігання логів має бути ефективним не лише в контексті обсягу даних, але й швидкості пошуку. OpenSearch використовує технологію інвертованих індексів, що забезпечує миттєвий пошук потрібних логів навіть у великих масивах даних. Завдяки цьому адміністратори можуть швидко знаходити помилки, діагностувати проблеми та аналізувати поведінку систем у реальному часі.

3) OpenSearch підтримує інтеграцію з іншими системами для збирання, обробки та аналізу логів, такими як Logstash і Beats. Це дозволяє легко налаштувати процес доставки логів до OpenSearch з різних джерел, включаючи сервери додатків, бази даних, мережеві пристрої та API. Також OpenSearch підтримує сумісність з існуючими інструментами на базі

Elasticsearch, що дозволяє мігрувати на OpenSearch з мінімальними витратами.

4) Однією з ключових переваг OpenSearch є його ліцензія Apache 2.0. Це забезпечує прозорість, гнучкість і відсутність прихованих обмежень у використанні платформи. Для підприємств, які прагнуть уникати прив'язки до постачальників (vendor lock-in), OpenSearch є надійним вибором, оскільки користувачі мають повний доступ до вихідного коду та можуть його модифікувати відповідно до власних потреб.

5) OpenSearch надає інструменти для забезпечення безпеки логів. Вбудовані функції, такі як шифрування даних, контроль доступу на основі ролей (RBAC) та аудит дій, допомагають захистити конфіденційні дані. Це особливо важливо для підприємств, що працюють у сферах фінансів, охорони здоров'я та електронної комерції, де безпека є критичною вимогою.

6) OpenSearch Dashboards дозволяє будувати інтуїтивно зрозумілі візуалізації для аналізу логів у режимі реального часу. Адміністратори можуть налаштовувати дашборди для моніторингу основних метрик, таких як кількість помилок, час відповіді та продуктивність системи. Це допомагає швидко виявляти аномалії, що значно скорочує час на діагностику проблем.

7) OpenSearch включає модуль Anomaly Detection, який використовує алгоритми машинного навчання для автоматичного виявлення відхилень у логах. Це дозволяє своєчасно реагувати на потенційні проблеми або загрози без необхідності вручну аналізувати великі обсяги даних.

Використання OpenSearch без інтеграції з Logstash може бути доцільним у випадках, коли система потребує меншої складності або має специфічні вимоги до обробки даних. Цей підхід передбачає використання інших методів для передачі даних до OpenSearch, наприклад, через API, Beats, власні скрипти або інші системи обробки даних.

Головною перевагою такого підходу є його простота. Відсутність додаткового компоненту у вигляді Logstash зменшує складність налаштування та обслуговування системи. Це також знижує навантаження на

ресурси, оскільки немає потреби запускати окремий сервіс для обробки даних. Замість цього можна використовувати прямі інтеграції через API або спеціалізовані інструменти, що забезпечує більшу гнучкість у створенні індивідуальних сценаріїв. Такий підхід сприяє швидшій передачі даних, оскільки усуває проміжні етапи, які були б необхідні при використанні Logstash. Крім того, відмова від Logstash може знизити витрати, особливо якщо враховувати вартість підтримки та ліцензування комерційних розширень.

Однак цей підхід має і недоліки. Відсутність Logstash обмежує можливості обробки даних, оскільки цей інструмент надає потужний функціонал для їх трансформації. В разі його відсутності необхідно реалізовувати такі операції власноруч або залучати інші інструменти, що може ускладнити інтеграцію. Підключення до різних джерел даних також може вимагати більше зусиль для створення кастомних рішень, адже стандартні плагіни, які пропонує Logstash, будуть недоступні. Відсутність таких стандартних конекторів створює додаткові труднощі при роботі з деякими типами даних чи протоколами, що може вимагати написання власного коду інтеграції. Це підвищує ризики помилок і вимагає значних ресурсів на тестування та підтримку системи.

Таким чином, OpenSearch сам по собі надає потужні інструменти для збору, індексації та пошуку даних, однак його можливості в попередній обробці можуть бути обмеженими. Якщо дані потребують нормалізації, фільтрації чи агрегування до потрапляння в сховище, використання Logstash або інших інструментів обробки стає майже обов’язковим. Також використання OpenSearch без Logstash є доцільним, якщо пріоритетом є простота налаштування, продуктивність або економія ресурсів. Проте цей підхід не завжди підходить для складних сценаріїв із численними джерелами даних або з потребою у складній обробці інформації. У таких випадках варто зважити всі переваги і недоліки, щоб зробити оптимальний вибір для конкретного проекту.

2.2. Огляд методів оптимізації операційних витрат на зберігання логів

Зберігання логів є невід'ємною частиною моніторингу, аналізу та підтримки IT-систем, особливо в e-commerce компаніях, де велика кількість транзакцій і запитів генерує величезні обсяги даних. Логи допомагають виявляти помилки, контролювати продуктивність системи, аналізувати поведінку користувачів та забезпечувати безпеку. Проте з часом витрати на зберігання логів стрімко зростають, особливо коли дані накопичуються в геометричній прогресії. Саме тому оптимізація операційних витрат на їх зберігання стає критичним завданням для бізнесу, що прагне підвищити ефективність роботи інфраструктури.

Одним із ключових методів оптимізації є використання архітектур, оптимізованих для зберігання логів. У сучасному IT-середовищі застосування розподілених систем, таких як OpenSearch або Elasticsearch, дозволяє ефективно розподіляти дані між кількома вузлами [10]. Це забезпечує горизонтальне масштабування, тобто можливість додавати нові сервери або сховища у міру зростання обсягу логів. Розподілена архітектура також мінімізує ризики надмірного навантаження на окремі вузли, що покращує продуктивність та надійність системи. Крім того, важливим кроком є встановлення політик життєвого циклу логів. Логи, залежно від їхньої важливості, можуть зберігатися протягом різного часу. Наприклад, "гарячі" дані, що є найактуальнішими для аналітики, зберігаються у швидкодоступних сховищах протягом кількох тижнів. "Холодні" дані, які є менш критичними, можуть переміщуватися до менш продуктивних та дешевших сховищ через 3-6 місяців. Архівні дані старше 6 місяців можна стискати та переносити у хмарні архіви або на фізичні носії з низькою вартістю зберігання.

Ще одним методом оптимізації витрат є компресія логів, яка дозволяє значно зменшити обсяг даних. Використання сучасних алгоритмів стиснення,

таких як gzip або LZ4, дозволяє досягти зменшення розміру файлів на 50-90%, залежно від їхньої структури. У поєднанні з агрегацією логів компресія дозволяє мінімізувати обсяги даних для зберігання. Агрегація передбачає об'єднання логів за певними часовими інтервалами або метриками. Наприклад, замість зберігання кожного запиту до сервера можна записувати агреговані дані: кількість запитів за хвилину, середній час відповіді сервера або кількість помилок. Це значно зменшує обсяг даних без втрати корисної інформації [11].

Перехід на хмарні сховища є ефективним способом оптимізації витрат на логування. Платформи, такі як Amazon S3, Google Cloud Storage або Azure Blob Storage, пропонують гнучкі моделі зберігання даних за принципом "pay-as-you-go", де оплата здійснюється виключно за фактично використаний обсяг. Хмарні рішення дозволяють легко масштабувати ресурси для зберігання, що особливо важливо при непередбачуваних сплесках даних. Для зниження витрат на хмарне зберігання варто застосовувати політики життєвого циклу даних, які автоматично переміщують логи з "гарячих" сховищ у "холодні", а згодом архівують їх. Також перед завантаженням даних у хмару рекомендується застосовувати компресію, щоб скоротити їх обсяг [12].

Оптимізація індексації логів є ще одним важливим аспектом. Індексація дозволяє виконувати швидкий пошук у великих обсягах даних, проте вона потребує додаткових ресурсів. Для зменшення витрат варто обмежити кількість полів, що індексуються. Наприклад, можна індексувати лише ключові поля, такі як час події, тип помилки чи ідентифікатор користувача. Це значно знижує споживання пам'яті та дискового простору. Додатково важливо налаштувати ротацію індексів, щоб кожен індекс відповідав за певний часовий інтервал (наприклад, день або тиждень). Це спрощує управління даними та знижує витрати на їх обробку.

Індекси в OpenSearch можуть бути класифіковані за їхнім призначенням і характеристиками доступу до даних, що дозволяє

оптимізувати систему зберігання логів для досягнення балансу між продуктивністю, вартістю та доступністю. Основні типи індексів включають гарячі, теплі, холодні та заморожені, кожен із яких має унікальні властивості й відповідає різним сценаріям використання [13].

Гарячі індекси містять найактуальніші та найчастіше запитувані дані. Вони розташовуються на вузлах із високопродуктивними ресурсами, такими як SSD та оперативна пам’ять, що забезпечує низьку латентність для операцій запису та читання. Ці індекси використовуються для даних, які активно оновлюються або запитуються, наприклад, для логів, отриманих протягом останніх кількох днів. Гарячі індекси є критичними для бізнес-аналітики та моніторингу в реальному часі.

Теплі індекси використовуються для даних, які рідше оновлюються або запитуються, але все ще мають аналітичну цінність. Вони зберігаються на вузлах із менш дорогими ресурсами, наприклад, HDD, і підходять для історичних даних, які періодично використовуються для аналізу або створення звітів. Теплі індекси є ефективним рішенням для логів, що зберігаються кілька тижнів або місяців тому, забезпечуючи компроміс між продуктивністю та витратами.

Холодні індекси створені для даних, які практично не запитуються, але повинні зберігатися для відповідності регуляторним вимогам або архівним потребам. Вони зберігаються на економному обладнанні з низьким рівнем продуктивності, що дозволяє значно зменшити витрати на зберігання. Дані в таких індексах доступні для пошуку, але швидкість виконання запитів значно нижча. Це ідеальне рішення для логів, які потрібні для аудиторських перевірок або юридичних цілей.

Заморожені індекси, на відміну від інших, не зберігаються в оперативній пам’яті, а доступ до них здійснюється безпосередньо з диску. Це забезпечує мінімальне споживання ресурсів, але запити до таких даних є повільними. Заморожені індекси зазвичай використовуються для

довготривалого зберігання великих обсягів даних, які потрібні лише у виняткових випадках.

Для оптимізації системи зберігання логів важливим є використання політик управління життєвим циклом даних (ILM), що дозволяють автоматизувати переміщення даних між різними типами індексів залежно від їхньої актуальності. Наприклад, нові логи автоматично потрапляють до гарячих індексів, а з часом переміщуються до теплих або холодних. Це забезпечує ефективне використання ресурсів. Крім того, використання різних класів зберігання дозволяє оптимізувати витрати: гарячі індекси зберігаються на швидких SSD, теплі та холодні — на HDD, а заморожені можуть бути розміщені на об'єктному сховищі, наприклад, Amazon S3 [12].

Популярною стратегією зменшення витрат є скорочення обсягу даних у менш пріоритетних індексах шляхом зменшення кількості реплік або видалення їх для заморожених індексів. Після закінчення терміну зберігання дані можна архівувати або видаляти. Масштабування також відіграє важливу роль: гарячі вузли обслуговують поточні запити, тоді як теплі та холодні вузли залишаються активними лише тоді, коли потрібний доступ до їхніх даних.

Грамотне використання гарячих, теплих, холодних та заморожених індексів дозволяє створити економічно ефективну й продуктивну архітектуру для зберігання логів. Це дає змогу забезпечити швидкий доступ до актуальних даних, мінімізувати витрати на зберігання історичних даних і задовольнити всі бізнес-вимоги до аналітики й архівування.

Автоматизація процесів управління логами також сприяє зниженню витрат. Інструменти, такі як Logstash, Fluentd та Beats, дозволяють автоматизувати процеси збору, обробки та зберігання логів. Це зменшує потребу в ручній роботі та підвищує ефективність роботи системи. Автоматизація включає фільтрацію логів, видалення непотрібних даних та застосування компресії у реальному часі перед їх записом у сховище. Крім

того, інструменти автоматизації дозволяють забезпечити централізоване управління логами з різних джерел, що спрощує їх обробку.

Для зменшення операційних витрат можна використовувати інструмент OpenSearch під назвою ILM-політики, що дозволяє проводити автоматизацію процесів управління індексами на різних етапах їхнього життєвого циклу. ILM-політики дозволяють оптимізувати використання ресурсів, знизити витрати на зберігання та підвищити ефективність системи шляхом автоматичного переміщення, стискання або видалення індексів у відповідності до їхньої актуальності та бізнес-вимог [14].

Основна ідея ILM полягає в тому, щоб визначити кілька послідовних фаз, які проходить індекс у своєму життєвому циклі. Найпоширенішими фазами є гаряча, тепла, холодна та видалення. На кожному з цих етапів виконується певний набір дій, які визначаються політикою. Наприклад, у гарячій фазі індекс залишається повністю активним і використовується для операцій читання та запису, забезпечуючи високу продуктивність. У теплій фазі індекс може бути переведений у стан тільки для читання, а також переміщений на дешевші ресурси. Холодна фаза призначена для тривалого зберігання даних на ресурсах із мінімальною вартістю, а видалення передбачає очищення індексу після закінчення його терміну зберігання.

Політики ILM дозволяють автоматизувати ці процеси. Адміністратор визначає умови переходу між фазами, такі як вік індексу, обсяг збережених даних чи рівень активності. Наприклад, індекс може автоматично переходити з гарячої у теплу фазу після досягнення певної кількості днів, або переміщуватися у холодну фазу після зменшення частоти запитів. Це дозволяє зменшити ручну роботу, пов'язану з управлінням індексами, і забезпечити відповідність їхнього стану актуальним потребам.

Важливим компонентом ILM є можливість стиснення даних у теплих і холодних фазах, що зменшує обсяг дискового простору, необхідного для зберігання. Також можна зменшити кількість реплік, що використовується

для індексу, або навіть видалити репліки для архівних даних у холодній фазі, оскільки такі дані зазвичай мають низький пріоритет доступу.

Застосування ILM політик забезпечує ефективний розподіл даних у кластері. Гарячі індекси розміщуються на високопродуктивних вузлах із SSD, теплі та холодні — на економних вузлах із HDD, а індекси в холодній фазі можуть бути навіть перенесені в об'єктні сховища, такі як Amazon S3 [12]. Це дозволяє значно зменшити витрати на інфраструктуру без шкоди для доступності даних.

У підсумку, політики ILM є незамінним інструментом для побудови масштабованих і економічно ефективних систем зберігання даних. Вони дозволяють знизити витрати, автоматизувати управління індексами та забезпечити доступність даних протягом усього їхнього життєвого циклу, одночасно задовольняючи потреби бізнесу в аналітиці, моніторингу й архівуванні.

Таким чином, оптимізація операційних витрат на зберігання логів передбачає впровадження ряду стратегій, які охоплюють вибір оптимізованої архітектури, компресію та агрегацію даних, перехід на хмарні рішення, оптимізацію індексації та автоматизацію процесів. Використання цих методів дозволяє досягти значної економії ресурсів, одночасно забезпечуючи надійність та доступність логів. Застосування сучасних рішень, таких як OpenSearch, є одним з найбільш ефективних підходів завдяки його можливостям розподіленого зберігання, масштабування та гнучкого управління життєвим циклом даних. Впровадження оптимізованих методів зберігання логів дозволяє e-commerce компаніям значно знизити витрати, підвищити продуктивність інфраструктури та забезпечити стабільний розвиток бізнесу.

2.3. Вибір програмних засобів для реалізації оптимізованої архітектури зберігання логів

Вибір програмних засобів для зберігання та обробки логів є ключовим етапом побудови надійної та ефективної архітектури. Оптимізована система повинна забезпечувати високу продуктивність, масштабованість, надійність і низькі операційні витрати. На сучасному етапі розвитку технологій інструменти для зберігання логів не тільки вирішують задачі збереження, а й інтегруються в аналіз даних та автоматизацію процесів.

Розглянемо конкретні інструменти, які використовуються для реалізації оптимізованої архітектури зберігання логів: Zend Framework, Cron, Aurora MySQL, AmazonMQ та Amazon OpenSearch Service. Кожен із цих інструментів виконує важливу роль у загальній системі обробки, зберігання та аналізу логів.

Zend Framework є одним із популярних PHP-фреймворків для побудови надійних та масштабованих веб-додатків. У контексті реалізації системи логування він використовується як основа для обробки запитів, генерації логів і інтеграції з іншими сервісами. Zend Framework має наступні переваги та особливості.

Однією з ключових причин вибору Zend Framework є його модульна структура (Рис. 2.2). Фреймворк надає окремі компоненти для роботи з логами, що дозволяє створювати адаптивні системи зберігання даних. Наприклад, компонент Log дозволяє записувати журнали у різні формати, включаючи текстові файли, бази даних, консольні виходи або навіть віддалені сервери через HTTP-запити. Це забезпечує розробникам високу гнучкість і можливість інтеграції з іншими сервісами.

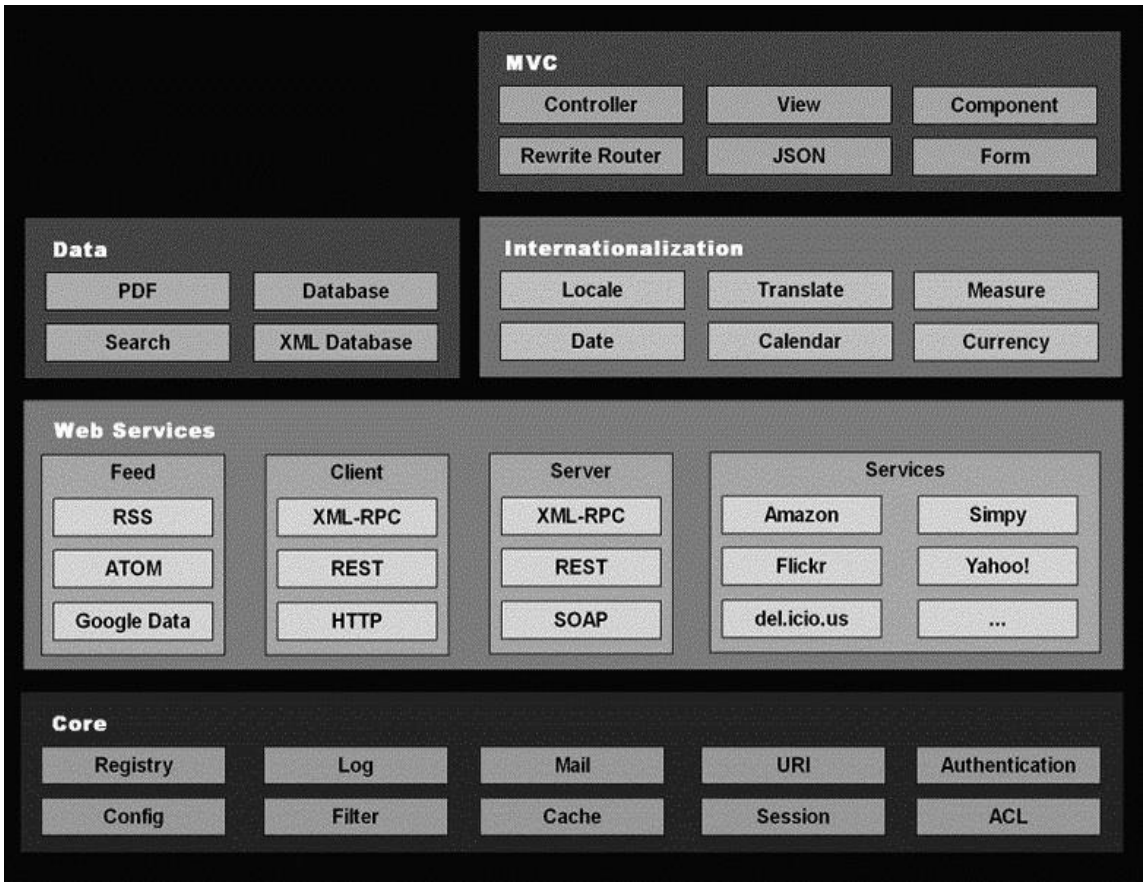


Рисунок 2.2 – Огляд модулів Zend Framework

Іншою важливою перевагою є підтримка обробки великих обсягів даних завдяки вбудованій оптимізації. Zend Framework дозволяє реалізувати багаторівневі стратегії логування, використовуючи фільтри й пріоритети для розмежування важливості подій. Це означає, що критичні помилки можуть зберігатися окремо від менш важливих записів, таких як інформаційні повідомлення чи налагоджувальні дані. Такий підхід знижує навантаження на системи зберігання і прискорює доступ до важливих даних.

Фреймворк також має високу сумісність із популярними базами даних і хмарними платформами. Завдяки цьому Zend Framework ідеально підходить для інтеграції з сучасними системами аналітики, такими як Elasticsearch або Splunk, що використовуються для аналізу логів у реальному часі. Це забезпечує можливість розширення функціоналу зберігання логів до інструменту для бізнес-аналітики, моніторингу та виявлення аномалій у роботі систем. Окрім технічних переваг, Zend Framework має розвинену

спільноту розробників і добре документовану базу знань. Це спрощує навчання нових розробників і підтримку системи в довгостроковій перспективі. Крім того, наявність великої кількості готових модулів та інтеграцій дозволяє скоротити час і витрати на розробку.

Безпека є важливим фактором, що виділяє Zend Framework серед інших. Вбудовані механізми захисту, фільтрація даних та захист від SQL - ін’єкцій забезпечують надійність роботи з конфіденційними логами. Це особливо важливо для систем, які потребують зберігання особистих даних користувачів або фінансової інформації.

Таким чином, Zend Framework виступає основою системи, на якій будується логіка обробки і первинної обробки логів перед їх передачею на подальші етапи.

Наступним елементом системи логування є Cron. Cron – це стандартний інструмент для планування завдань у Unix-подібних операційних системах. У архітектурі логування Cron використовується для автоматизації періодичних завдань, таких як архівація, очистка або агрегація логів.

За допомогою Cron можна автоматизувати такі процеси:

1) Періодичне очищення старих логів для зменшення використання дискового простору.

2) Перенесення логів із оперативних сховищ (наприклад, SSD) у архівні сховища на базі Amazon S3 [12].

3) Регулярна реплікація логів у бази даних, такі як Aurora MySQL.

Однією з головних переваг Cron є його здатність виконувати завдання у визначений час або з певною періодичністю. Це дозволяє регулярно збирати логи з різних систем і пристроїв, забезпечуючи їхню актуальність. Наприклад, за допомогою Cron можна автоматизувати процеси копіювання логів на віддалений сервер, архівування старих записів або запуск скриптів для очищення застарілих даних. Такий підхід мінімізує ручне втручання та підвищує ефективність управління даними.

Крім того, Cron підтримує високу гнучкість у налаштуванні. Використовуючи файл конфігурації crontab, адміністратори можуть задавати складні розклади виконання завдань, враховуючи години, дні тижня, місяці та навіть певні комбінації цих параметрів. Це особливо корисно для систем зберігання логів, де можуть існувати різні вимоги до частоти збору та обробки даних залежно від джерела або важливості інформації.

Інтеграція з іншими інструментами є ще однією сильною стороною Cron. У сучасних системах логування Cron часто використовується у зв’язці зі скриптами на Python, Bash або іншими мовами програмування. Це дозволяє легко інтегрувати Cron із системами моніторингу, такими як Nagios або Zabbix, а також із базами даних або сховищами логів, такими як Elasticsearch чи Splunk. Наприклад, Cron може запускати скрипти для парсингу логів, передавання їх до бази даних або генерації звітів.

Безпека та стабільність Cron також є важливими факторами для вибору цього інструменту. Оскільки Cron є вбудованим компонентом більшості Unix-подібних операційних систем, він має перевірену часом архітектуру і високий рівень надійності. Адміністратори можуть задавати права доступу до завдань, обмежуючи можливість їхнього запуску неавторизованими користувачами. Це забезпечує додатковий рівень захисту для конфіденційних логів і критично важливих даних.

Ще одним значущим аспектом є простота використання. Конфігурація завдань не вимагає глибоких технічних знань, і завдяки чіткій синтаксичній структурі файлу crontab (Рис. 2.3) розробники та адміністратори можуть легко налаштовувати та змінювати розклад виконання. Це сприяє швидкому впровадженню Cron у вже існуючі процеси логування без значних витрат часу.

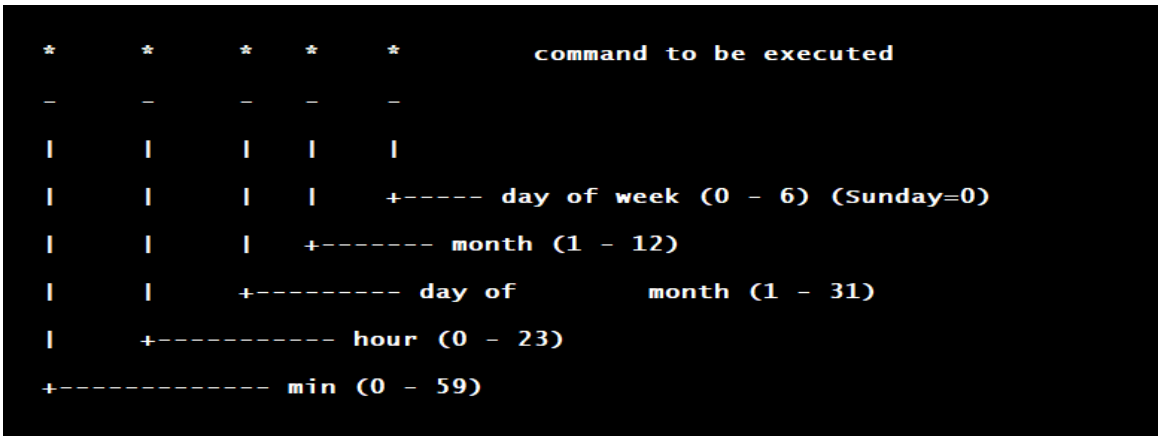


Рисунок 2.3 – Схема синтаксису Cron для налаштування автоматизації

Cron також забезпечує підтримку журналювання власної роботи. Це означає, що адміністратори можуть переглядати результати виконання завдань, включно з інформацією про помилки, які виникли під час виконання. Така функціональність спрощує моніторинг і налагодження автоматизованих процесів, підвищуючи загальну стабільність системи.

Таким чином, використання Cron для реалізації системи зберігання логів забезпечує регулярність, гнучкість і надійність автоматизованих процесів. Його можливості інтеграції, безпека та простота конфігурації роблять його незамінним інструментом для збору, обробки та управління логами в сучасних динамічних середовищах.

Для храніння системної інформації системи логування буде використовуватися Amazon Aurora MySQL, що є високопродуктивною базою даних, яка забезпечує високу швидкість збереження та доступу до даних з оптимізованими витратами. Використання Aurora MySQL у контексті зберігання логів є обґрунтованим завдяки наступним особливостям:

1) Aurora MySQL забезпечує продуктивність до п’яти разів вищу, ніж звичайні MySQL-бази, завдяки унікальній архітектурі, що дозволяє оптимізувати роботу з великими обсягами даних.

2) Aurora MySQL підтримує автоматичне горизонтальне масштабування до 128 терабайтів, що дозволяє зберігати величезні обсяги логів без додаткових налаштувань.

3) Aurora MySQL використовує модель оплати за фактично використаний обсяг ресурсів, що дозволяє оптимізувати витрати на зберігання та обробку даних порівняно з традиційними реляційними СУБД.

4) Aurora MySQL автоматично створює репліки даних і забезпечує високий рівень відмовостійкості, що є критично важливим для збереження важливих логів.

Таким чином, Aurora MySQL є ідеальним рішенням для збереження структурованих логів у випадках, коли необхідна висока надійність і продуктивність.

В якості брокера повідомлень система храніння логів використовує AmazonMQ, що є керованою службою черг повідомлень на базі популярного брокера RabbitMQ. У архітектурі оптимізованого зберігання логів AmazonMQ використовується для передачі логів від додатків до систем зберігання, таких як OpenSearch (Рис. 2.4).

Рисунок 2.4 – Схема роботи AmazonMQ

RabbitMQ та Amazon MQ є рішеннями для обміну повідомленнями між системами, але вони мають суттєві відмінності в архітектурі, ліцензуванні, підтримці та випадках використання.

Розглянемо різницю між двома технологіями. RabbitMQ — це популярний брокер повідомлень із відкритим кодом, який підтримує кілька протоколів, зокрема AMQP, MQTT, STOMP і HTTP. Його основна перевага полягає у гнучкості та можливості адаптації до різних сценаріїв. RabbitMQ зазвичай використовується для розробки і розгортання в інфраструктурах, де потрібний високий рівень контролю над конфігурацією і продуктивністю. Це дозволяє компаніям налаштовувати систему під свої унікальні вимоги, зокрема інтегрувати її в багатокомпонентні середовища. RabbitMQ може

працювати локально, в хмарі або в контейнерах, і потребує самостійного адміністрування. Amazon MQ є керованим сервісом від AWS, заснованим на Apache ActiveMQ. Він автоматизує налаштування, управління, масштабування та резервне копіювання, позбавляючи користувачів необхідності адмініструвати сервери вручну. Amazon MQ підтримує стандартні протоколи обміну повідомленнями, включно з AMQP, MQTT, STOMP і OpenWire, що робить його сумісним із багатьма існуючими додатками. Основна перевага Amazon MQ полягає в інтеграції з іншими сервісами AWS, такими як Lambda, S3 або DynamoDB, що дозволяє створювати складні хмарні архітектури з мінімальними зусиллями [15].

Ключова різниця між ними полягає в моделі розгортання та керування. RabbitMQ вимагає повного контролю над інфраструктурою, але й потребує більше ресурсів для налаштування та підтримки. Це підходить для компаній, які хочуть повністю управляти своєю системою. Amazon MQ, натомість, орієнтований на спрощення обслуговування і зменшення операційних витрат, але прив’язує користувачів до екосистеми AWS.

Розглянемо переваги технології AmazonMQ [16]:

1) Завдяки механізмам чергування повідомлень AmazonMQ забезпечує гарантовану доставку логів навіть у разі тимчасових збоїв системи.

2) AmazonMQ дозволяє обробляти великі обсяги даних, завдяки чому можна налаштувати ефективний збір і передачу логів від Zend Framework до Amazon OpenSearch Service.

3) AmazonMQ легко інтегрується з іншими сервісами AWS, що забезпечує безперебійну роботу системи з мінімальними витратами.

Amazon OpenSearch Service є центральним елементом архітектури зберігання та аналізу логів. Це повністю керована служба на основі OpenSearch, яка забезпечує швидкий пошук і аналітику для великих обсягів неструктурованих даних. Цей сервіс є найкращим вибором для зберігання логів на сьогодні з наступних причин:

1) OpenSearch оптимізований для роботи з логами у форматі JSON, що дозволяє здійснювати миттєвий пошук за ключовими словами, фільтрами та умовами.

2) OpenSearch підтримує політики ILM для автоматичного переміщення логів у холодні або архівні сховища, що дозволяє оптимізувати витрати на зберігання.

3) Сервіс забезпечує горизонтальне масштабування, що дозволяє обробляти великі обсяги даних з мінімальними затримками.

4) Завдяки інтеграції з іншими інструментами AWS, OpenSearch спрощує передачу, обробку та візуалізацію логів.

Обрані програмні засоби – Zend Framework, Cron, Aurora MySQL, AmazonMQ та Amazon OpenSearch Service – забезпечують побудову надійної, масштабованої та економічно ефективної архітектури для зберігання логів. Кожен інструмент виконує свою роль: від генерації та передачі логів до їх зберігання та аналізу. Поєднання цих технологій дозволяє оптимізувати операційні витрати та забезпечити високу продуктивність системи.

Отже, Zend Framework виступає основою для розробки додатків, забезпечуючи структуровану обробку даних і інтеграцію з іншими сервісами. Cron автоматизує виконання завдань, таких як періодичний збір і відправлення логів, що сприяє підтриманню стабільної роботи системи. Aurora MySQL забезпечує ефективне зберігання метаінформації, пов'язаної з логами, та підтримує швидкий доступ до даних завдяки своїй оптимізованій архітектурі для роботи в хмарному середовищі. AmazonMQ відіграє ключову роль у передачі повідомлень між компонентами системи, забезпечуючи надійну та асинхронну комунікацію навіть за умов високого навантаження. Amazon OpenSearch дозволяє швидко індексувати, зберігати й аналізувати великі обсяги даних, надаючи інструменти для візуалізації та побудови аналітичних дашбордів.

2.4. Порівняльний аналіз OpenSearch та реляційних баз даних

Реляційні бази даних, такі як MySQL, PostgreSQL або Microsoft SQL Server, забезпечують традиційний підхід до зберігання даних, що ґрунтується на строго визначених схемах і відношеннях. Вони пропонують високий рівень структурованості та надійності, зокрема підтримку транзакцій, забезпечення цілісності даних і багаті можливості для складного аналізу за допомогою SQL-запитів. У контексті логування їх зазвичай використовують для зберігання структурованих логів, які потребують чіткого форматування.

OpenSearch є сучасним інструментом для аналізу великих обсягів неструктурованих або слабо структурованих даних. Його архітектура оптимізована для пошуку, агрегації та візуалізації логів у реальному часі. Він здатний ефективно обробляти дані, що надходять із різних джерел, таких як сервери додатків, системи моніторингу та мережеве обладнання. Відкрита архітектура OpenSearch дозволяє інтегрувати його з різноманітними джерелами даних та інструментами візуалізації, такими як Kibana.

Одним із ключових аспектів порівняння є швидкість роботи з даними. Реляційні бази даних демонструють високу продуктивність при виконанні складних транзакційних операцій та забезпеченні консистентності даних. Однак із ростом обсягів логів їх продуктивність може знижуватися через перевантаження індексаційних механізмів та обмеження обчислювальної потужності під час виконання складних запитів. OpenSearch, навпаки, оптимізований для роботи з великими обсягами даних завдяки горизонтальному масштабуванню та ефективному алгоритму розподілу даних між вузлами кластеру. Це дозволяє йому швидко обробляти запити навіть при значних обсягах логів [17].

Зручність використання також є важливим фактором. Реляційні бази даних вимагають створення схем і налаштування таблиць перед початком роботи, що може ускладнювати інтеграцію нових джерел логів. Водночас

вони пропонують добре задокументовані інтерфейси та багатий вибір засобів аналізу, що робить їх зручними для класичних застосунків. OpenSearch не потребує суворого дотримання структурованих схем, що спрощує інтеграцію нових даних. Завдяки засобам пошуку та агрегації, він дозволяє оперативно знаходити необхідну інформацію та будувати динамічні візуалізації.

Додатково, у випадку OpenSearch, інтеграція із системами типу Fluentd або Logstash дозволяє автоматизувати обробку вхідних логів.

Операційні витрати також відрізняються. Реляційні бази даних зазвичай дешевші у впровадженні для невеликих обсягів даних, оскільки їх можна розгорнути на одному сервері. Проте їх масштабування вимагає значних інвестицій у додаткове обладнання та ліцензії. OpenSearch, будучи системою з відкритим вихідним кодом, дозволяє знизити витрати на програмне забезпечення, а його горизонтальне масштабування забезпечує ефективніше використання ресурсів у разі зростання обсягів логів. Однак варто врахувати, що супровід кластеру OpenSearch потребує спеціалізованих знань та ресурсів. У середовищі з понад 100 млн записів логів на добу витрати на хмарну інфраструктуру для OpenSearch можуть бути на 30% нижчими, ніж у реляційних баз даних, але вимагають команди спеціалістів для підтримки.

Що стосується математичних алгоритмів, реляційні бази даних покладаються на оптимізовані механізми індексації та виконання SQL-запитів, що підходить для структурованих даних. OpenSearch використовує розподілені обчислення, деревоподібні структури індексації та алгоритми агрегації, які забезпечують швидкий пошук і аналіз навіть у нерегулярних масивах даних. Це робить його придатним для складних аналітичних завдань, таких як пошук аномалій чи побудова часових рядів.

Архітектурно OpenSearch використовує підхід "поділ і завоювання". Дані розбиваються на шардові індекси [18], кожен із яких обробляється незалежно. Це значно зменшує затримки, у той час як реляційні бази даних

часто працюють із монолітними таблицями, що може спричиняти затримки при масштабних запитах.

Отже, OpenSearch та реляційні бази даних мають різні переваги та сфери застосування у контексті логування. Реляційні бази даних є найкращим вибором для структурованих логів, де важлива транзакційна консистентність і можливість виконання складних SQL-запитів. OpenSearch, зі свого боку, оптимально підходить для аналізу великих обсягів неструктурованих логів, швидкого пошуку та динамічної візуалізації даних. У випадках із високим навантаженням або великим обсягом логів OpenSearch демонструє кращу масштабованість і продуктивність. Вибір між цими двома підходами залежить від специфіки завдання, доступних ресурсів та вимог до аналізу даних.

2.5. Висновок до другого розділу

У другому розділі було проаналізовано сучасні архітектури логування, методи оптимізації операційних витрат та вибрано програмні засоби для побудови ефективної системи зберігання логів. Розглянуті рішення та підходи дозволяють досягти балансу між продуктивністю, масштабованістю та економічною доцільністю в умовах великих обсягів даних.

Аналіз архітектур логування показав, що сучасні системи, такі як ELK, OpenSearch та інші платформи, надають необхідний функціонал для збору, обробки та аналітики логів. Особливу увагу було приділено OpenSearch, який поєднує відкритість, високу гнучкість та оптимізацію витрат завдяки хмарним можливостям. Саме OpenSearch забезпечує ефективну обробку великих масивів логів із мінімальними ресурсними витратами.

Застосування сучасних методів оптимізації витрат у поєднанні з правильно обраними програмними засобами відкриває нові можливості для ефективного зберігання, аналізу та управління логами навіть у масштабних системах. Наприклад, використання таких інструментів, як Amazon OpenSearch, дозволяє створювати розподілені системи зберігання, які забезпечують високу продуктивність і масштабованість. Інтеграція з хмарними рішеннями, такими як Amazon S3, сприяє зниженню витрат на підтримку локальної інфраструктури, одночасно підвищуючи гнучкість і доступність даних. Використання інструментів візуалізації, таких як Kibana або OpenSearch Dashboards, допомагає компаніям швидко отримувати аналітичні інсайти та приймати обґрунтовані рішення.

Методи оптимізації операційних витрат, розглянуті у підрозділі, доводять, що застосування структурної оптимізації логів, розподілених систем, компресії, автоматизації процесів та політик управління життєвим циклом дозволяють суттєво знизити фінансові витрати. Особливо важливим є використання хмарних технологій, які пропонують гнучкі моделі оплати та автоматичне масштабування відповідно до навантаження.

Окрему увагу приділено питанням стійкості системи до можливих збоїв і втрати даних. Виявлено, що використання реплікації на рівні індексів і вузлів кластеру дозволяє підтримувати безперервний доступ до логів навіть за умов технічних несправностей. Такий підхід забезпечує високу надійність системи, що є критично важливим для бізнесів із високими вимогами до доступності інформації.

На основі вибору програмних засобів було визначено інструменти, які найкраще підходять для реалізації оптимізованої архітектури: Zend Framework, Cron, Aurora MySQL, AmazonMQ та Amazon OpenSearch Service. Поєднання цих технологій дозволяє побудувати надійну, масштабовану та економічно ефективну систему логування, що відповідає сучасним викликам і потребам бізнесу. Інтеграція обраних інструментів із системами моніторингу, бізнес-аналітики та безпеки створює єдину екосистему, яка

дозволяє аналізувати дані у режимі реального часу. Це відкриває додаткові можливості для прийняття стратегічних рішень, персоналізації користувацького досвіду та підвищення ефективності операцій.

Таким чином, проведене дослідження демонструє, що ефективна система логування впливає на внутрішні бізнес-процеси та на якість обслуговування клієнтів. Завдяки швидшому виявленню помилок і аналізу поведінки користувачів компанії можуть оперативніше впроваджувати зміни, що позитивно позначається на задоволеності клієнтів та їх лояльності. Застосування сучасних методів оптимізації витрат у поєднанні з правильно обраними програмними засобами дозволяє ефективно зберігати, аналізувати та управляти логами у великих системах. Це не лише забезпечує надійну роботу архітектури логування, але й сприяє зниженню витрат на інфраструктуру та підвищенню загальної продуктивності системи.

РОЗДІЛ 3. ОПТИМІЗАЦІЯ АРХІТЕКТУРИ ЛОГУВАННЯ E-COMMERCE КОМПАНІЇ

3.1. Розробка моделі архітектури логування на основі реальних даних e-commerce компанії

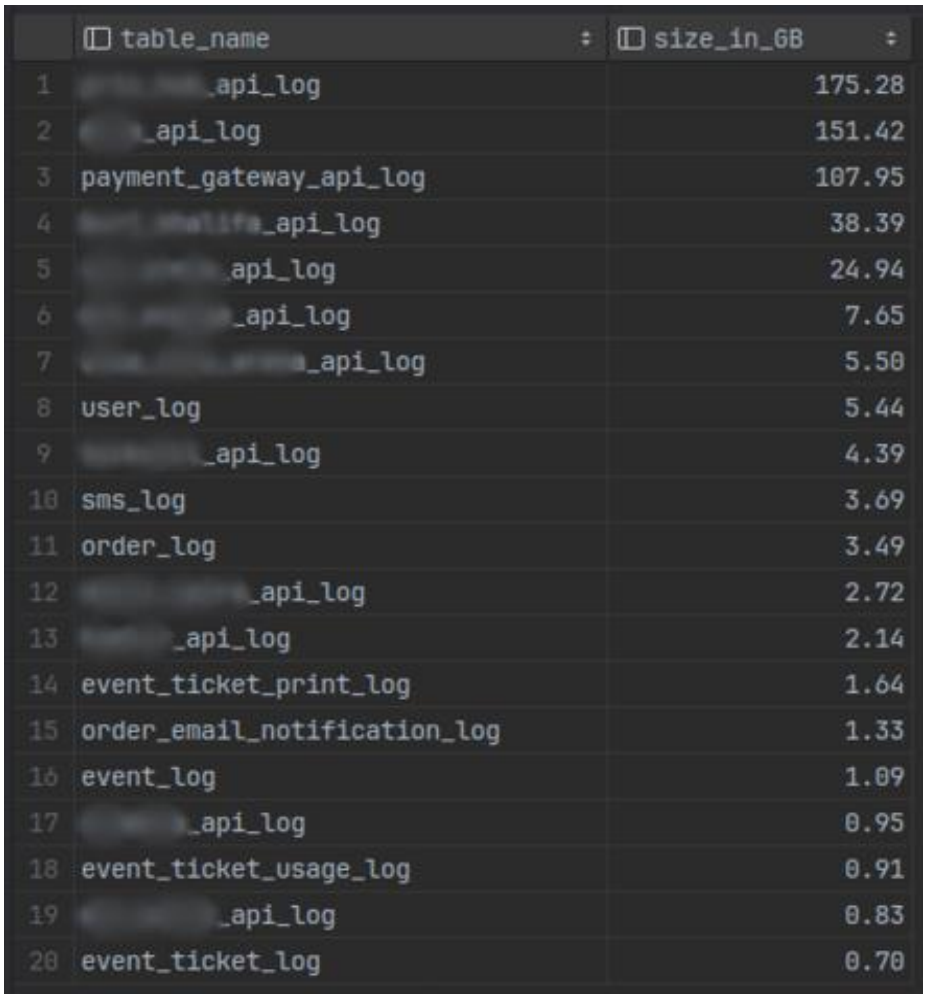
Однією з ключових проблем, що виникають у процесі масштабування систем логування в e-commerce компаніях, є збереження великих обсягів даних та забезпечення ефективного доступу до них. У поточній архітектурі логів однієї з популярних e-commerce компаній з продажу електронних квитків на Близькому Сході, всі логи зберігаються у базі даних Amazon Aurora MySQL. Це рішення на етапі впровадження задовольняло базові вимоги до зберігання логів, однак зі зростанням кількості записів воно стало менш ефективним як з точки зору швидкості доступу до даних, так і операційних витрат.

Щоб підрахувати найбільші таблиці бази даних MySQL e-commerce компанії та вивести їх розмір у гігабайтах, можна скористатися системною таблицею «information_schema.tables» (Рис. 3.1).

```
SELECT
    table_name AS `table_name`,
    ROUND(SUM(data_length + index_length) / (1024 * 1024 * 1024), 2) AS `size_in_GB`
FROM
    information_schema.tables
WHERE
    table_schema NOT IN ('information_schema', 'mysql', 'performance_schema', 'sys') -- Виключаємо системні БД
    AND table_type = 'BASE TABLE'
    AND table_name LIKE '%_log'
GROUP BY
    table_schema, table_name
ORDER BY
    `size_in_GB` DESC
LIMIT 20;
```

Рисунок 3.1 – SQL-запит для підрахунку розміру таблиць логування у гігабайтах

За результатами підрахунку усі логи e-commerce компанії займають понад 540 ГБ (Рис. 3.2).



	table_name	size_in_GB
1	[redacted]_api_log	175.28
2	[redacted]_api_log	151.42
3	payment_gateway_api_log	107.95
4	[redacted]_api_log	38.39
5	[redacted]_api_log	24.94
6	[redacted]_api_log	7.65
7	[redacted]_api_log	5.50
8	user_log	5.44
9	[redacted]_api_log	4.39
10	sms_log	3.69
11	order_log	3.49
12	[redacted]_api_log	2.72
13	[redacted]_api_log	2.14
14	event_ticket_print_log	1.64
15	order_email_notification_log	1.33
16	event_log	1.09
17	[redacted]_api_log	0.95
18	event_ticket_usage_log	0.91
19	[redacted]_api_log	0.83
20	event_ticket_log	0.70

Рисунок 3.2 – Найбільші таблиці логів e-commerce компанії

Першим кроком у процесі міграції буде перенесення таблиці user_log, яка займає 5.44 ГБ (Рис. 3.2) та містить журнали активності користувачів. Ця таблиця є важливим джерелом даних для моніторингу та аналітики поведінки користувачів, а також є хорошим прикладом для тестування можливостей OpenSearch у реальних умовах.

Для реалізації системи логування було створено модель, яка враховує наступні основні компоненти:

1) Aurora MySQL – реляційна база даних, яка використовується для зберігання логів у структурованому вигляді.

2) AmazonMQ (RabbitMQ) – черга повідомлень для асинхронного оброблення логів.

3) OpenSearch – движок пошуку і аналітики, що забезпечує швидкий доступ до логів, їхню індексацію та візуалізацію.

4) OpenSearchFacade – спеціально створений програмний інтерфейс для спрощення роботи з OpenSearch (створення індексів, додавання документів).

5) Worker – фоновий обробник, який відповідає за міграцію даних та обробку логів у реальному часі.

На основі цих компонентів було створено ефективну архітектуру, яка забезпечує збір, зберігання, перенесення та обробку логів у e-commerce середовищі. Центральною особливістю системи є використання OpenSearch для зберігання і обробки логів завдяки його високій продуктивності та можливостям масштабування.

Процес перенесення логів з Aurora MySQL до OpenSearch було реалізовано таким чином, щоб мінімізувати вплив на продуктивність системи. Основні кроки процесу міграції наведено нижче:

1) У Aurora MySQL створюється таблиця «open_search_log», куди додається інформація та налаштування про усі необхідні логи.

2) Створюється індекс у OpenSearch за допомогою інтерфейсу OpenSearchFacade. Індекси в OpenSearch оптимізовані для пошуку, що дозволяє ефективно обробляти великі обсяги логів у подальшому.

3) Перенесення даних з Aurora MySQL до Amazon OpenSearch з використанням алгоритму пакетної обробки з посторінковою пагінацією на основі зміщення.

4) Після перенесення всіх логів, процес міграції завершується, а дані стають доступними для подальшого пошуку та аналізу через OpenSearch.

Для перенесення даних використовується алгоритм пакетної обробки з посторінковою пагінацією на основі зміщення (Рис. 3.3):

1) Ініціалізується змінна offset зі значенням 0 та лічильник кількості записів на ітерацію count, що дорівнює 5000.

2) Визначається порядок сортування даних за полем «id_user_log» у зростаючому порядку.

3) У циклі do-while зчитуються дані з бази MySQL у вигляді списку записів розміром 5000 штук за допомогою обмеження LIMIT.

4) Для кожного запису формується документ.

5) Сформовані документи відправляються у OpenSearch методом bulkCreateDocuments, що дозволяє оптимізувати швидкість додавання.

6) Після успішного додавання документів кеш оброблених даних очищується для зменшення використання пам’яті.

7) Значення offset збільшується на 5000, і цикл повторюється з паузою у 1 секунду для зниження навантаження на систему.

8) Процес завершується, коли нові дані відсутні у вибірці з бази даних.

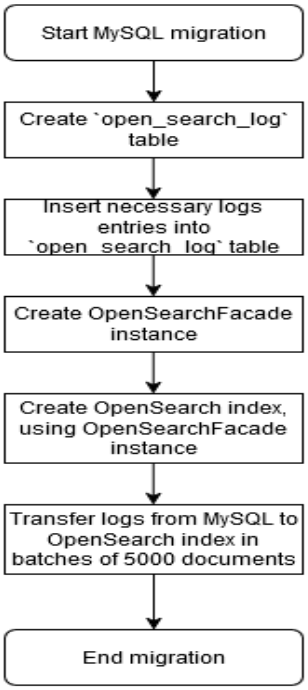


Рисунок 3.3 – Алгоритм міграції даних з Aurora MySQL до Amazon OpenSearch

Після завершення міграції система переходить до режиму обробки логів у реальному часі. Основні етапи цього процесу (Рис. 3.4) наступні:

1) Користувач виконує дію: При взаємодії користувача з e-commerce платформою генерується лог-повідомлення.

2) Повідомлення передається у RabbitMQ: Лог-повідомлення надсилається в чергу RabbitMQ для асинхронної обробки. Такий підхід дозволяє уникнути затримок у роботі основної системи.

3) Worker обробляє повідомлення: Фоновий обробник зчитує повідомлення з RabbitMQ і перевіряє його на повноту даних.

4) Перевірка індексу у OpenSearch: Якщо індекс вже існує, лог додається як документ у OpenSearch. Якщо індекс відсутній, він створюється автоматично.

5) Видалення конфіденційної інформації: Перед додаванням документу до OpenSearch, видаляється вся чутлива інформація, що забезпечує відповідність політикам конфіденційності.

6) Додавання логів до OpenSearch: Логи додаються до відповідного індексу OpenSearch, після чого повідомлення видаляється з черги RabbitMQ.

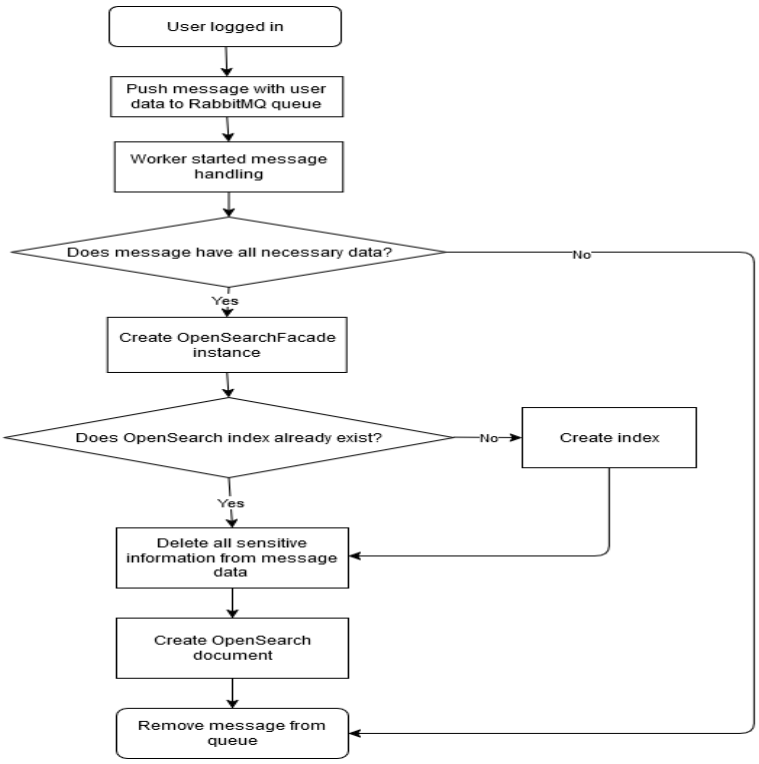


Рисунок 3.4 – Алгоритм обробки логів користувача у реальному часі

Завдяки цьому процесу система забезпечує ефективну обробку логів у реальному часі, зберігаючи їх у OpenSearch для подальшого пошуку та аналізу.

Розроблена модель архітектури логування має низку ключових переваг:

1) OpenSearch дозволяє горизонтально масштабувати систему шляхом додавання нових вузлів до кластеру, що забезпечує обробку великих обсягів логів.

2) Завдяки індексації даних OpenSearch забезпечує миттєвий пошук і можливість глибокої аналітики логів.

3) Використання RabbitMQ і OpenSearchFacade дозволяє легко інтегрувати нові джерела даних у систему логування.

4) Видалення чутливої інформації перед збереженням у OpenSearch забезпечує захист конфіденційних даних.

5) Перенесення логів до OpenSearch знижує навантаження на Aurora MySQL, що покращує продуктивність основної бази даних.

Запропонована модель архітектури логування поєднує надійність реляційних баз для збору структурованих даних та можливості OpenSearch для їхнього аналізу й індексації. Розроблена архітектура орієнтована на e-commerce компанії, що працюють з великими потоками даних, такими як клієнтські дії, платіжна інформація та дані про продуктивність. Завдяки гнучкості й масштабованості, ця модель легко адаптується до зростання обсягів даних і нових вимог ринку.Завдяки використанню AmazonMQ для асинхронної обробки повідомлень та OpenSearchFacade для взаємодії з OpenSearch, система забезпечує високу продуктивність, масштабованість і гнучкість. Процес міграції логів з Aurora MySQL до OpenSearch реалізовано у вигляді пакетної обробки, що дозволяє ефективно переносити великі обсяги даних. Ця модель є оптимальним рішенням для e-commerce компаній, які працюють з великими обсягами логів і потребують швидкого доступу до них для аналітики та моніторингу.

3.2. Реалізація оптимізованої системи логування з використанням Amazon OpenSearch

У процесі реалізації оптимізованої системи логування особливу увагу приділено взаємодії між ключовими компонентами: Pusher, Worker, Facade, Client, Index/UserLog та GarbageCollector. Кожен із цих елементів має чітко визначену роль і функціонал, забезпечуючи ефективну роботу системи як єдиного цілого.

Розглянемо кожен з елементів системи більш детально. Pusher є відправною точкою у процесі передачі логів з існуючої системи у OpenSearch. Основна роль Pusher полягає у створенні черги для обробки пакетів логів та їх передачі у OpenSearch через Facade. Цей компонент реалізує механізм масової передачі логів, що мінімізує кількість запитів до OpenSearch, а отже, знижує навантаження на систему та мережеву інфраструктуру.

Pusher тісно взаємодіє з іншими елементами системи: він отримує дані з Index/UserLog, де вони форматуються у вигляді документів, що відповідають вимогам OpenSearch, а потім передає їх через Facade до черги AmazonMQ. Перед передачею виконується перевірка цілісності даних та підготовка пакетів для масової індексації. Завдяки цьому Pusher забезпечує стабільність та надійність процесу перенесення логів.

Воркер SaveOpenSearchLog призначений для обробки та збереження логів у системі OpenSearch. Його основна задача — отримати повідомлення з певними даними, перевірити їх на коректність і, якщо все в порядку, передати їх для збереження в індекс OpenSearch. Воркер спочатку перевіряє, чи є всі необхідні параметри в отриманому повідомленні, а також чи відповідає вказаний клас типу логу необхідним вимогам. Після цього створюється об'єкт цього класу, який визначає індекс для логу, і дані очищуються перед тим, як бути записаними в OpenSearch.

Використання воркерів для обробки даних має низку переваг порівняно з виконанням логіки в рантаймі. Ці переваги особливо важливі для високонавантажених систем і сценаріїв, де критичними є продуктивність, масштабованість та стабільність.

По-перше, воркери дозволяють розділити обробку задач на асинхронні процеси, що зменшує затримки в основному потоці програми. Наприклад, якщо виконання складних обчислень або інтеграція із зовнішніми сервісами відбувається під час обробки HTTP-запиту, це може збільшувати час відповіді для користувача. Використання воркерів переносить такі задачі у фоновий режим, дозволяючи основному процесу швидше завершити роботу.

По-друге, воркери забезпечують масштабованість. У рантаймі кожна задача виконується в межах того ж самого ресурсу (CPU або пам’яті), що й основна логіка додатка, що може призводити до перевантаження. Воркери дозволяють розподіляти обробку задач між окремими потоками або навіть серверами, підвищуючи продуктивність і забезпечуючи стійкість до пікових навантажень.

По-третє, використання воркерів підвищує стабільність системи. У рантаймі будь-яка помилка або збій під час виконання складної логіки може вплинути на роботу всієї програми, призводячи до збоїв або деградації сервісу. Воркери ізольовані від основного потоку, тому помилки в них не блокують основні процеси. Крім того, за допомогою черг можна автоматично повторювати задачі, що не вдалося виконати з першого разу.

По-четверте, воркери ефективніше працюють із задачами, що потребують великої кількості ресурсів. У рантаймі такі задачі можуть вичерпувати доступну пам’ять або обчислювальні потужності, що негативно впливає на доступ до ресурсу для інших користувачів. Воркери надають можливість виділяти обробку ресурсоємних задач в окремі процеси з обмеженням ресурсів, забезпечуючи тим самим стабільну роботу основної програми.

Також воркери спрощують масштабування за рахунок розподіленої обробки. У рантаймі складніше додавати нові ресурси для обробки задач, тоді як воркери легко масштабується горизонтально, що є критично важливим для великих систем із динамічним навантаженням.

У разі, якщо середовище є локальним, воркер також створює індекс у OpenSearch, що забезпечує наявність потрібної структури для зберігання логів. Потім дані логу форматуються, і їх передають для запису в OpenSearch. Якщо під час роботи виникає помилка, вона фіксується у системі моніторингу для подальшого аналізу. Воркер працює за визначеним алгоритмом, що забезпечує стабільну роботу з логами в різних середовищах.

Facade виконує роль центрального інтерфейсу між бізнес-логікою системи та інфраструктурою OpenSearch. Цей компонент є абстракцією, що об’єднує низькорівневі виклики API OpenSearch та надає зручний інтерфейс для виконання основних операцій: створення індексів, масової передачі даних, оновлення та пошуку документів.

Ключовою особливістю Facade є його здатність працювати з великими обсягами даних у режимі реального часу. Він виконує обробку даних блоками, використовуючи механізм «bulk» API OpenSearch, що дозволяє одночасно передавати велику кількість логів. У разі виникнення помилок Pusher аналізує відповіді OpenSearch і автоматично повторює спробу передачі проблемних пакетів.

Завдяки Facade усі запити до OpenSearch централізовані й стандартизовані, що забезпечує зручність підтримки системи та її подальшої модифікації. Facade отримує дані від Pusher у вигляді підготовлених пакетів і передає їх до Client для подальшої обробки. Крім того, він контролює процес передачі, обробляючи помилки та повідомлення про статус виконання операцій.

Facade також реалізує методи для створення нових індексів та управління ними, що дозволяє легко додавати нові типи логів у систему. Він

взаємодіє з Index/UserLog для отримання налаштувань індексації та визначення структури документів індексу.

Створення Facade надає наступні переваги:

1) Усі операції, пов'язані з OpenSearch (створення індексів чи виконання bulk-запитів) приховані за єдиним інтерфейсом, що спрощує код і робить його чистішим.

2) Facade дозволяє виконувати bulk-запити для створення документів, що є більш ефективним, ніж додавання документів по одному. Це суттєво знижує час на індексацію великих обсягів даних.

3) Facade підтримує перевірку існування індексів перед їх створенням, автоматичне оновлення індексів і видалення конфіденційних даних із логів перед їхнім завантаженням.

4) Facade дозволяє уникнути дублювання коду у різних частинах системи, оскільки він є універсальним інтерфейсом для роботи з OpenSearch.

Client є найнижчим рівнем взаємодії з OpenSearch. Цей компонент відповідає за ініціалізацію з’єднання з OpenSearch, виконання HTTP-запитів та обробку їх результатів. Client використовує OpenSearch SDK для PHP, що дозволяє працювати з API OpenSearch на низькому рівні.

Основна задача Client полягає у виконанні команд, отриманих від Facade, таких як масова індексація документів, створення індексів або пошук даних. Client є ключовим елементом, який забезпечує безпосередню інтеграцію системи з інфраструктурою OpenSearch.

Client також реалізує механізми обробки помилок, такі як повторні спроби виконання запитів у разі тимчасових збоїв або проблем з мережею. Завдяки цьому забезпечується стабільна робота системи навіть при пікових навантаженнях.

Index/UserLog є компонентом, що визначає структуру індексації логів у OpenSearch. Цей елемент відповідає за опис схеми даних, що включає поля індексів та їх типи даних. Для індексу «user_log» визначено ключові поля: ідентифікатор користувача, IP-адреса, текстове повідомлення, тип події та

час створення логу. Index/UserLog формує документи у структурованому вигляді, що відповідає вимогам OpenSearch. Ця структура є критично важливою для ефективного пошуку та аналітики логів у подальшому. Крім того, компонент Index/UserLog надає можливість налаштування індексівю. Наприклад, можна налаштувати кількість шардів та реплік, інтервали оновлення та політику збереження даних.

GarbageCollector є компонентом, що відповідає за управління ресурсами системи та видалення застарілих даних. Основною функцією GarbageCollector є реалізація retention policy, що визначає термін зберігання логів у системі.

GarbageCollector взаємодіє з Facade та Client, виконуючи запити на видалення документів, що більше не відповідають критеріям зберігання. Це дозволяє оптимізувати використання ресурсів OpenSearch та підтримувати стабільність системи. Цей елемент забезпечує управління пам’яттю під час обробки великих обсягів даних. Він виконує очищення кешу та звільнення ресурсів після обробки кожного пакету логів, що дозволяє уникнути перевантаження системи.

Оптимізована система логування побудована на тісній взаємодії між усіма ключовими компонентами:

1) Index/UserLog визначає структуру даних та надає підготовлені документи.

2) Pusher отримує ці документи, формує пакети для масової індексації та передає їх у воркер.

3) Worker передає дані у Facade.

4) Facade координує процес передачі, використовуючи Client для виконання запитів до OpenSearch.

5) Client реалізує низькорівневі виклики API OpenSearch, виконуючи передачу даних та обробку відповідей.

6) GarbageCollector забезпечує підтримку системи шляхом очищення застарілих даних і управління ресурсами.

Така архітектура забезпечує ефективне перенесення логів з Aurora MySQL до OpenSearch, дозволяючи досягти високої продуктивності, масштабованості та надійності системи логування. Взаємодія між компонентами побудована таким чином, щоб забезпечити мінімальне навантаження на ресурси системи та максимальну ефективність обробки великих обсягів даних.

3.3. Ефективне налаштування сервісу Amazon OpenSearch

Для забезпечення максимальної ефективності роботи сервісу необхідно ретельно налаштувати його компоненти: індекси, шардінг, реплікацію, кешування та інші аспекти. Розглянемо конкретні рекомендації для налаштування та математичні основи алгоритмів, які використовуються в OpenSearch, та порівняємо їх з алгоритмами Amazon RDS.

Ефективність роботи OpenSearch починається з правильного налаштування індексів. Зменшення розміру індексу можна досягти за рахунок вимкнення зберігання невикористовуваних полів, таких як «_source», та використання алгоритмів компресії. Для економії пам’яті рекомендується застосовувати компресію «zstd», яка забезпечує збалансовану швидкість та ступінь стискання.

Структура mapping відіграє ключову роль у швидкості запитів. Для точних збігів слід використовувати тип «keyword», а для повнотекстового пошуку — «text». Динамічна типізація може бути відключена (налаштування «dynamic: false»), що сприяє стабільності структури індексів. Аналіз тексту оптимізується за допомогою кастомних аналізаторів. Наприклад, застосування «lowercase» для нормалізації текстів із різними регістрами або використання «stop» для вилучення малозначущих слів.

Розмір сегментів можна оптимізувати за допомогою параметра «index.merge.scheduler.max_thread_count», який залежить від кількості ядер процесора. Це допомагає уникати блокувань при злитті сегментів.

Розділення індексів на шарди дозволяє рівномірно розподіляти навантаження між вузлами [19]. Розглянемо формулу (3.1) визначення оптимальної кількості шардів:

$$N = ceil(\frac{S}{M}), \tag{3.1}$$

де — S — загальний розмір індексу;

M — максимально рекомендований розмір однієї шарди (зазвичай не більше 50 ГБ).

Наприклад, якщо індекс займає 500 ГБ, то за формулою (3.1) для нього потрібно не менше 10 шардів:

$$N = ceil\left(\frac{500}{50}\right) = 10$$

Для підвищення продуктивності пошуку рекомендується мінімізувати кількість реплік під час індексації, збільшуючи їх лише для читання у великих кластерах. Рівень реплікації налаштовується командою «PUT /my-index/_settings { "index": {"number_of_replicas": 2}}» [19].

Політики життєвого циклу (ILM) дозволяють автоматизувати архівування та видалення старих даних. Наприклад, після 30 днів дані можуть бути переміщені до «теплих» вузлів, а після 90 днів — видалені. Використання в ILM технологій, таких як машинне навчання, дозволяє адаптуватися до змін і підвищувати точність маркування товарів. Це також сприяє зменшенню операційних витрат, оскільки автоматизація процесів допомагає знизити витрати на робочу силу та скоротити час, необхідний для виконання операцій, пов'язаних із маркуванням товарів.

OpenSearch використовує декілька типів кешування для прискорення запитів. Query Cache зберігає результати однакових запитів, знижуючи

навантаження на обчислення. Розглянемо формулу (3.2) оцінки ефективності кешування:

$$E = \left(\frac{H}{T}\right) * 100, \tag{3.2}$$

де — H — кількість попадань у кеш;
T — загальна кількість запитів.

Кешування налаштовується через параметр «PUT /_cluster/settings {" persistent": {"indices.queries.cache.size": 10%}}» [19]. Shard Request Cache корисний для агрегаційних запитів і активується через «index.requests.cache.enable». Поле Field Data Cache потрібно регулярно очищати, щоб уникнути перевантаження пам’яті.

Зробимо порівняльний аналіз Amazon OpenSearch з більш класичним підходом зберігання даних Amazon Relational Database Service. Amazon OpenSearch використовує інвертовані індекси для прискорення пошуку. В інвертованому індексі кожне слово чи фраза асоціюється з усіма документами, в яких це слово зустрічається, що дозволяє швидко здійснювати пошук за ключовими словами [5]. Розглянемо формулу (3.3) інвертованого індексу, що дозволяє значно зменшити час пошуку:

$$index(\omega) = \{d_1, d_2, \dots, d_n\}, \tag{3.3}$$

де — ω — слово;
d_n — документ, що містить слово.

RDS використовує традиційні реляційні індекси (B-дерева для швидкого пошуку за ключем), що ефективно працюють зі структурованими даними. Якщо логи зберігаються в таблиці, то пошук по полях можна реалізувати через індекси, що також працюють за принципом B-дерева чи хешування [17].

Amazon OpenSearch має вбудоване масштабування, яке підтримує горизонтальне масштабування (шардинг) і автоматичний баланс

навантаження [20]. Кожен шард — це частина даних, яка може бути розміщена на окремому сервері. Завдяки розподіленій архітектурі, навантаження розподіляється по кількох серверах рівномірно. Це дозволяє ефективно працювати з великими обсягами даних, що поступово зростають. Розглянемо формулу (3.4) масштабування Amazon OpenSearch:

$$TotalData = \sum_{i=1}^{m} Shards_i, \tag{3.4}$$

де — m — кількість шардів;

$Shards_i$ — обсяг даних в кожному шарді.

Amazon RDS може масштабуватися вертикально (збільшення потужностей одного сервера) або горизонтально за допомогою реплікацій. Однак вертикальне масштабування обмежене технічними можливостями конкретної СУБД, а горизонтальне масштабування через реплікацію складніше при роботі з великими обсягами логів, оскільки структура таблиць вимагає додаткової логіки для обробки розподілених запитів [17].

OpenSearch розроблений для швидкого пошуку по великих обсягах неструктурованих даних. Завдяки використанню інвертованих індексів, запити на пошук по ключових словах виконуються дуже швидко. OpenSearch може здійснювати запити за час, пропорційний логарифму від кількості документів. Час пошуку можна розрахувати за формулою (3.5):

$$Time(search) = O(\log n), \tag{3.5}$$

де — n — кількість документів у системі.

RDS не призначений для такої інтенсивної роботи з неструктурованими даними, як OpenSearch. Хоча індекси в RDS забезпечують швидкий доступ до даних, складні пошукові запити можуть мати більший час відгуку, особливо при великих обсягах неструктурованих логів. Час пошуку у RDS залежить від типу запиту та індексу, але зазвичай його можна розрахувати за формулою (3.5) для простих запитів по індексах.

OpenSearch добре підходить для складних аналітичних запитів з агрегацією даних, таких як обчислення статистик або виявлення аномалій в логах. Він використовує алгоритми агрегації на основі індексів для швидкої обробки великих обсягів даних. Агрегація в OpenSearch може виконуватися за допомогою алгоритмів, таких як рівномірний розподіл або вибірка з підмножин для швидкої обробки запитів [20].

RDS здатний виконувати агрегацію даних через SQL-запити, такі як GROUP BY або JOIN, але зазвичай це менш ефективно, ніж в OpenSearch при роботі з великими неструктурованими даними, оскільки РСУБД орієнтовані на обробку структурованих даних.

OpenSearch здатний виконувати обробку та пошук даних в реальному часі завдяки своїй архітектурі, яка підтримує швидке індексування та агрегацію даних. Кожен запис може бути індексований за допомогою алгоритмів на основі методу векторного простору, що дозволяє швидко отримувати результат при великих навантаженнях.

Отже, Amazon OpenSearch є більш оптимізованим для зберігання і пошуку логів завдяки своїй архітектурі, заснованій на інвертованих індексах, розподіленому масштабуванню та здатності до швидкої агрегації. Amazon RDS більш підходить для зберігання структурованих даних, де важлива цілісність та організація даних, але він менш ефективний для роботи з великими обсягами неструктурованих логів.

Математично OpenSearch перевершує RDS у роботі з великими масивами неструктурованих даних завдяки горизонтальному масштабуванню, що дозволяє розподіляти навантаження між кількома вузлами, і оптимізації пошуку через інвертовані індекси. У той же час RDS краще справляється з вертикальною масштабованістю й забезпеченням транзакційної цілісності, що робить його ідеальним вибором для роботи з добре структурованими даними. Таким чином, вибір між цими системами залежить від типу даних і вимог до продуктивності, пошуку й аналізу.

3.4. Порівняльний аналіз витрат компанії до та після оптимізації

Міграція логів з Amazon RDS в Amazon OpenSearch стала важливим кроком для оптимізації ресурсів компанії. Основним результатом цього переходу стало значне зменшення обсягу логів користувача. Обсяг вдалося скоротити у 3.85 рази: з 5.4GB до 1.4GB при однаковій кількості записів, що склала 55674785 рядків (Рис. 3.5). Така оптимізація стала можливою завдяки використанню ефективніших алгоритмів компресії та зберігання даних, які надає Amazon OpenSearch у порівнянні з Amazon RDS. Ці алгоритми дозволяють більш раціонально використовувати дисковий простір, забезпечуючи високу швидкість доступу до даних без втрати їхньої цілісності.

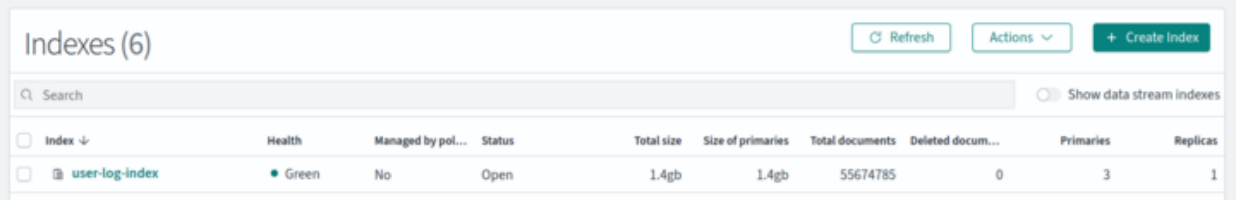


Рисунок 3.5 – OpenSearch індекс «user_log» після перенесення логів

Розглянемо детальніше, які переваги забезпечує зменшення обсягу логів. До проведення оптимізації загальний обсяг логів e-commerce компанії складав 540.45GB. Після перенесення цих даних до OpenSearch їхній розмір зменшився до $\frac{540.45}{3.85} \approx 140$ GB. Це не лише забезпечило економію місця, але й створило додаткові можливості для розширення логів у майбутньому без необхідності значного збільшення обсягів сховища. Важливо зазначити, що ця оптимізація не вплинула на вміст логів. Усі рядки даних були збережені в повному обсязі, що підтверджує надійність обраного рішення.

Критично важливим аспектом стала фінансова вигода від реалізації цього проекту. Раніше компанія витрачала близько 9275 доларів США на місяць на зберігання логів у сервісі Amazon RDS (Рис. 3.6). Ця сума включала

як вартість самого сховища, так і пов'язані з ним операційні витрати, зокрема резервне копіювання, обробку запитів та забезпечення доступності даних.

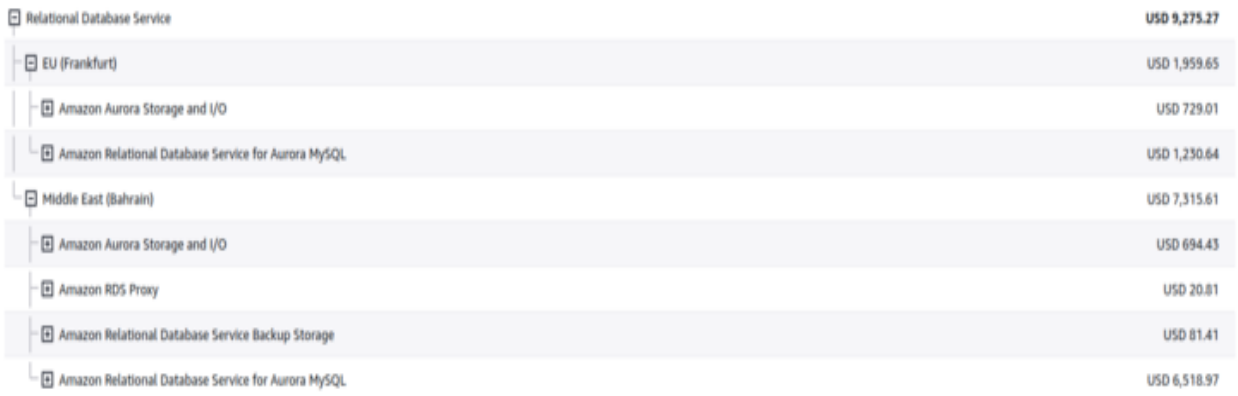


Рисунок 3.6 – Операційні витрати компанії на сервіс Amazon RDS до оптимізації

Після переносу логів компанії на Amazon OpenSearch витрати на сервіс Amazon RDS скоротилися до 6704 доларів США на місяць (Рис. 3.7). Це стало можливим завдяки меншій потребі у ресурсах для зберігання та обробки даних, а також завдяки ефективнішій архітектурі OpenSearch, яка дозволяє більш економно використовувати обчислювальні потужності.

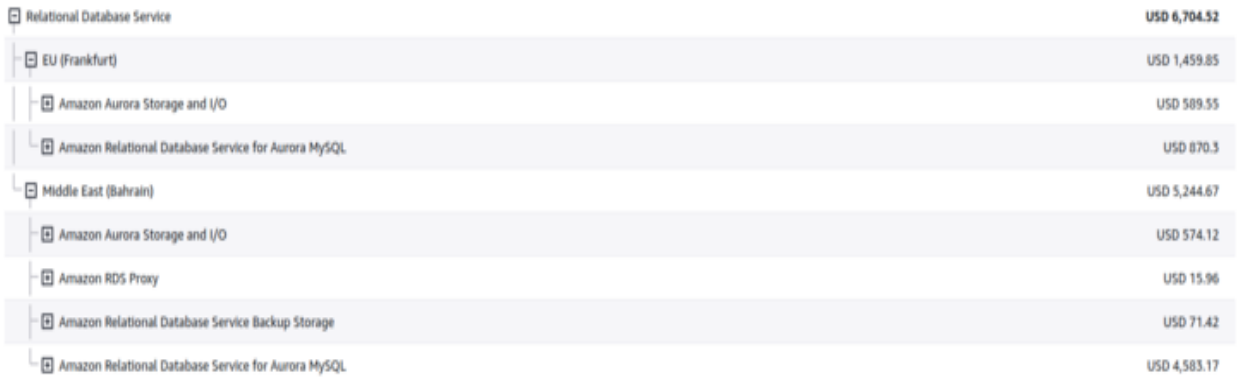


Рисунок 3.7 – Операційні витрати компанії на сервіс Amazon RDS після оптимізації

Також після реалізації системи логування додалися витрати компанії на сервіс Amazon OpenSearch (Рис. 3.8) у вигляді 1103 доларів США щомісяця.

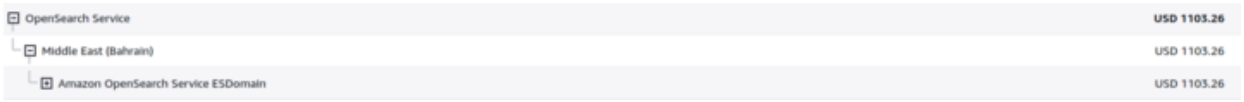


Рисунок 3.8 – Операційні витрати компанії на сервіс Amazon OpenSearch

Додатково, варто відзначити зменшення непрямих витрат. Менший обсяг даних скоротив час, необхідний для виконання пошукових запитів, що позитивно вплинуло на продуктивність системи загалом. Зниження затримок у роботі з логами дозволило покращити аналітичні можливості команди, оскільки тепер аналіз даних відбувається швидше та з меншою затратою ресурсів. Це, у свою чергу, сприяє прийняттю більш обґрунтованих рішень, підвищуючи загальну ефективність роботи компанії.

Таким чином, реалізація оптимізованої системи логування дозволила скоротити місячні операційні витрати e-commerce компанії на 1468 доларів США — з 9275 до 7807. Перехід на Amazon OpenSearch дозволив досягти як економії фінансових ресурсів, так і підвищення ефективності використання інфраструктури. Зменшення операційних витрат у поєднанні з підвищенням продуктивності створює значний потенціал для подальшого розвитку та масштабування бізнесу. Це рішення демонструє, наскільки важливим є вибір оптимальної технологічної платформи для досягнення стратегічних цілей компанії.

3.5. Висновок до третього розділу

У третьому розділі детально розглянуто перехід e-commerce компанії з Amazon RDS на Amazon OpenSearch, який вибрано як основну систему зберігання та обробки логів. Основним завданням цього процесу було замінити існуючу архітектуру на базі MySQL, яка в умовах зростаючого

обсягу даних виявилася менш ефективною з точки зору продуктивності та вартості.

Перехід на OpenSearch дозволив вирішити кілька ключових проблем. Насамперед, завдяки інвертованим індексам і механізму bulk-запитів вдалося суттєво прискорити пошук і аналітику даних. Це стало особливо важливим для компанії, яка обробляє великі обсяги логів у реальному часі. Інвертовані індекси забезпечують швидкий доступ до даних, а використання спеціально розробленого інтерфейсу OpenSearchFacade дозволило спростити роботу з цим сервісом, включаючи створення індексів і додавання нових документів.

Міграція логів з Aurora MySQL до OpenSearch реалізована з урахуванням потреб у мінімізації навантаження на систему. Завдяки використанню пакетної обробки та алгоритмів посторінкової пагінації міграція виконувалася без втрати даних та з мінімальним впливом на продуктивність основної бази даних. Після завершення цього процесу логи стали доступними для миттєвого пошуку та аналізу в OpenSearch, що значно підвищило швидкість і точність обробки запитів.

Важливим аспектом стало скорочення обсягу логів завдяки компресії та ефективній структурі зберігання даних в OpenSearch. Ефективна структура зберігання в OpenSearch забезпечила швидкий доступ до інформації навіть за великої кількості даних, покращивши продуктивність запитів і мінімізувавши час на пошук. Після перенесення обсяг даних зменшився у 3,85 рази без втрати інформації, що створило додатковий простір для майбутнього зростання. Крім того, значно знизилися витрати на зберігання та обробку логів — міграція дозволила зменшити операційні витрати компанії на 16%.

Новий підхід продемонстрував гнучкість та масштабованість архітектури. OpenSearch забезпечив можливість горизонтального масштабування шляхом додавання нових вузлів у кластер, що гарантує стабільну роботу навіть за умов зростання навантаження. Застосування RabbitMQ для асинхронної обробки повідомлень та OpenSearchFacade для

інтеграції з OpenSearch дозволило створити архітектуру, яка ефективно підтримує великі обсяги даних і швидко адаптується до змін.

Перехід на OpenSearch також сприяв підвищенню безпеки даних. Завдяки видаленню конфіденційної інформації перед збереженням у OpenSearch компанія забезпечила відповідність політикам конфіденційності, не поступаючись ефективністю аналітики.

У результаті реалізована система логування на базі OpenSearch дозволила не лише оптимізувати операційні процеси, але й підвищити загальну продуктивність роботи компанії. Вона стала надійним інструментом для обробки великих обсягів логів, забезпечуючи високу швидкість пошуку, гнучкість масштабування та економічну ефективність. У довгостроковій перспективі впровадження такої архітектури стало важливим стратегічним кроком, що підвищив конкурентоспроможність компанії. Оптимізація витрат на інфраструктуру, скорочення часу на обробку даних і підвищення надійності процесів сприяють не лише ефективності поточних операцій, але й створюють базу для подальшого масштабування й розвитку бізнесу в умовах цифрової трансформації.

ВИСНОВОК

У процесі виконання кваліфікаційної роботи було проведено глибокий аналіз сучасних методів зберігання та обробки логів у сфері електронної комерції. Основною метою дослідження було зниження операційних витрат на зберігання логів шляхом оптимізації існуючих архітектур і впровадження сучасних інструментів, зокрема, пошукового рушія Amazon OpenSearch.

Проаналізовані існуючі підходи до зберігання логів виявили ключові проблеми, які виникають у високонавантажених системах. Традиційні реляційні бази даних демонструють обмеження в масштабованості та продуктивності, особливо при роботі з великими обсягами напівструктурованих даних. Розподілені системи, такі як Elasticsearch, OpenSearch та Splunk, забезпечують кращу продуктивність і масштабованість завдяки своїй архітектурі, що орієнтована на обробку логів у реальному часі. Особливу увагу було приділено технологіям компресії, які дозволяють знижувати обсяг збережених логів без втрати їхньої інформативності.

Детально вивчено фактори, що впливають на вартість зберігання логів. Було встановлено, що збільшення витрат компанії спричиняють великі обсяги даних, вибір методів зберігання, частота доступу до логів та регуляторні вимоги. Використання політик життєвого циклу даних, таких як гарячі, теплі, холодні та заморожені індекси, дозволяє раціоналізувати використання ресурсів і знизити витрати. Такий підхід також забезпечує ефективну інтеграцію з хмарними платформами.

В рамках практичної частини роботи було розроблено алгоритм переходу з реляційної бази даних на платформу Amazon OpenSearch. Даний алгоритм враховує ключові аспекти міграції даних, включаючи підготовку та оптимізацію структури логів, налаштування індексів і забезпечення сумісності з існуючими сервісами. Тестування показало, що впровадження

OpenSearch дозволяє досягти значного зниження витрат на зберігання та підвищити продуктивність системи логування.

Проведений порівняльний аналіз операційних витрат до та після оптимізації продемонстрував економічну ефективність впровадженого рішення. Зниження витрат стало можливим завдяки впровадженню стиснення логів, оптимізації індексації та автоматизації управління життєвим циклом даних. Крім того, використання розподіленої архітектури сприяло підвищенню надійності системи та зменшенню часу доступу до даних.

Практичне значення роботи полягає у можливості застосування отриманих результатів для оптимізації процесів логування у компаніях електронної комерції. Запропонований підхід дозволяє забезпечити масштабованість і гнучкість систем зберігання логів, знизити витрати на інфраструктуру та підвищити ефективність аналітики. Це особливо актуально для бізнесів, що працюють із великими обсягами даних та стикаються із завданнями оперативного моніторингу та аналізу поведінки користувачів.

Підсумовуючи, виконана робота підтверджує доцільність використання сучасних пошукових рушіїв, таких як Amazon OpenSearch, для оптимізації зберігання логів у електронній комерції. Запропоновані рішення сприяють підвищенню продуктивності, зниженню витрат та забезпеченню конкурентоспроможності компаній у динамічних умовах ринку. Отримані результати відкривають перспективи для подальших досліджень, зокрема у напрямках автоматизації обробки логів за допомогою штучного інтелекту та інтеграції з іншими сервісами аналітики даних.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Roger Nem. Amazon Simple Storage Service (Amazon S3). AWS. URL: https://community.aws/content/2c6dBEmQPpXeGxBxdu0EHkXPXpC/amazon-simple-storage-service-amazon-s3 (дата звернення: 24.09.2024).

2. Опплігер Р. SSL і TLS: Теорія і практика. – 3-є вид. – Шам: Springer, 2023. – 383 с.

3. Joakim Eriksson, Anawil Karavek. A comparative analysis of log management solutions: ELK stack versus PLG stack. Diva Portal. URL: https://www.diva-portal.org/smash/get/diva2:1771279/FULLTEXT01.pdf (дата звернення: 24.09.2024).

4. Doina R. Zmaranda, Cristian I. Moisi. An Analysis of the Performance and Configuration Features of MySQL Document Store and Elasticsearch as an Alternative Backend in a Data Replication Solution. MDPI. URL: https://www.mdpi.com/2076-3417/11/24/11590 (дата звернення: 26.09.2024).

5. Jiahao Cheng. Real-Time Construction Algorithm of Co-Occurrence Network Based on Inverted Index. ArXiv.org. URL: https://arxiv.org/abs/2308.08756 (дата звернення: 26.09.2024).

6. Toni Linnusmaki. Increasing e-commerce conversion rate with relevant search results using Elasticsearch. Trepo. URL: https://trepo.tuni.fi/handle/10024/121439 (дата звернення: 11.10.2024).

7. Nikita Kathare, O. Vinati Reddy. A Comprehensive Study of Elastic Search. Bryan House Publications. URL: https://web.archive.org/web/20221203011409id_/https://www.bryanhousepub.org/src/static/pdf/JRSE-2022-4-11_7.pdf (дата звернення: 11.10.2024).

8. Architecting the OpenSearch service at CERN for openwebsearch.eu. DLR eLibrary. URL: https://elib.dlr.de/210749/1/172-176-PB.pdf#page=7 (дата звернення: 13.10.2024).

9. Субраманіан К. Practical Splunk Search Processing Language. – 1-е вид. – Нью-Йорк: APress, 2020. – 268 с.

10. Lu Han, Ligu Zhu. Design and Implementation of Elasticsearch for Media Data. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/9103825 (дата звернення: 29.10.2024).

11. Saravanan Kuppusamy. Mastering Elasticsearch: A Comprehensive Guide. Google Books. URL: https://books.google.com.ua/books?hl=uk&lr=&id=GGkJEQAAQBAJ&oi=fnd&pg=PT12 (дата звернення: 11.11.2024).

12. Taranjot Singh. The Effect of Amazon Web Services (AWS) on Cloud-Computing. ResearchGate. URL: https://www.researchgate.net/profile/Taranjot-Singh-4/publication/356809704_The_effect_of_Amazon_Web_Services_AWS_on_Cloud-Computing (дата звернення: 11.11.2024).

13. Abdulla Kalandar Mohideen, Shikharesh Majumdar. A Data Indexing Technique to Improve the Search Latency of AND Queries for Large Scale Textual Documents. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/9302542 (дата звернення: 27.11.2024).

14. Chunpeng Cai. Design of Corpus Distributed Retrieval System Based on Elasticsearch Search Engine. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/10796551 (дата звернення: 28.11.2024).

15. Юханссон Л. RabbitMQ Essentials – Second Edition: Створення розподілених і масштабованих додатків з використанням RabbitMQ. – 2-е вид. – Бірмінгем: Packt Publishing, 2020. – 154 с.

16. Praveen Borra. Comprehensive Survey of Amazon Web Services (AWS) Techniques, Tools, and Best Practices for Cloud Solutions. ResearchGate. URL: https://www.researchgate.net/profile/Praveen-Borra/publication/381956331_Comprehensive_Survey_of_Amazon_Web_Services_AWS_Techniques_Tools_and_Best_Practices_for_Cloud_Solutions (дата звернення: 07.12.2024).

17. Eniedson Fabiano Pereira da Silva Junior. On Searching Product Catalog: Relational Database Versus Search Engine approaches. ACM Digital Library. URL: https://dl.acm.org/doi/abs/10.1145/3658271.3658299 (дата звернення: 08.12.2024).

18. Siamak Solat. Sharding Distributed Databases: A Critical Review. ResearchGate. URL: https://www.researchgate.net/publication/379753203_Sharding_Distributed_Databases_A_Critical_Review (дата звернення: 14.12.2024).

19. Bizhong Wei, Jian Dai. An Optimization Method for Elasticsearch Index Shard Number. IEEE Xplore. URL: https://ieeexplore.ieee.org/abstract/document/9407368 (дата звернення: 20.12.2024).

20. Praveen M Dhulavvagol. Performance Analysis of Distributed Processing System using Shard Selection Techniques on Elasticsearch. ScienceDirect. URL: https://www.sciencedirect.com/science/article/pii/S1877050920308395 (дата звернення: 20.12.2024).