Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
(повна назва)

Кафедра програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження моделей збору інформації у системі електронної комерції
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-1

Маркевич А.В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник доц. Каук В.І.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерних наук
Кафедра програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Маркевичу Артуру Вінченцовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Дослідження моделей збору інформації у системі електронної комерції»

Затверджена наказом по університету від 29.03.2024р. № 250 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18.06.2024

3. Вихідні дані до роботи опис досліджуваних систем збору та обробки журнальних даних, галузі електронної комерції

4. Перелік питань, що потрібно опрацювати в роботі

аналіз та порівняння наявних систем збору інформації, розробка схеми даних для планування та виконання експериментальних досліджень, створення програмних засобів, проведення експериментів і аналізування отриманих результатів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	24.01 – 15.02.24	*виконано*
2	Аналіз та вибір API для дослідження	16.02 – 25.02.24	*виконано*
3	Аналіз та моделювання предметної області	18.02 – 28.02.24	*виконано*
4	Планування експериментів	24.02 – 28.02.24	*виконано*
5	Програмна реалізація кожного з обраних для дослідження API	25.02 – 01.05.24	*виконано*
6	Експериментальні дослідження	03.05 – 21.05.24	*виконано*
7	Аналіз результатів експериментальних досліджень та розробка рекомендацій	21.05 – 24.05.24	*виконано*
8	Написання та оформлення статті та тез доповіді	18.05 – 24.05.24	*виконано*
9	Підготовка пояснювальної записки	01.05 – 26.05.24	*виконано*
10	Підготовка презентації та доповіді	07.06.2024	*виконано*
11	Нормоконтроль	10.06.2024	*виконано*
12	Рецензування	15.06.2024	*виконано*
13	Занесення диплома в електронний архів	17.06.2024	*виконано*
14	Попередній захист	17.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	18.06.2024	*виконано*

Дата видачі завдання 20 січня 2024р.

Студент (ка) _____ Маркевич А.В.
(підпис)

Керівник роботи _____ доц. Каук В.І.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до кваліфікаційної роботи, 91 сторінки, 17 рисунків, 4 таблиць, 5 додатки, 22 джерела.

ЗБІР ІНФОРМАЦІЇ, СИСТЕМИ ЖУРНАЛЮВАННЯ, ЕЛЕКТРОННА КОМЕРЦІЯ, ЖУРНАЛЮВАННЯ, JAVA.

Об’єктом дослідження є системи збору інформації в контексті електронної комерції.

Метою роботи є визначення ефективності використання систем моделей збору інформації для систем електронної комерції.

У результаті роботи було досліджено ефективність використання систем моделей збору інформації для систем електронної комерції; розроблено системи генерації та відправки інформації в системи збору.

INFORMATION COLLECTION, LOGGING SYSTEMS, E-COMMERCE, LOGGING, JAVA.

The object of research is information collection systems in the context of e-commerce.

The purpose of the study is to determine the effectiveness of using information collection model systems for e-commerce systems.

As a result of the study, the efficiency of using information collection model systems for e-commerce systems was investigated.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Маркевич Артур Вінченцович, студент(ка) гр. ІПЗм-22-1, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження моделей збору інформації у системі електронної комерції», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений(на) з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

В сучасному світі інформаційні технології мають велике значення для розвитку бізнесу і забезпечення підтримки інфраструктури. Електронна комерція, яка є одним з ключових аспектів цієї інфраструктури, відіграє важливу роль у розвитку економіки. За даними Statista, у 2023 році глобальні продажі в електронній комерції досягли $5.2 трильйона, що підкреслює її значення та швидке зростання (Statista, 2023) [1].

Електронна комерція - це бізнес-модель, яка дозволяє купувати і продавати товари та послуги через Інтернет. Вона охоплює широкий спектр даних, систем та інструментів і може здійснюватися через комп'ютери, планшети, смартфони та інші "розумні" пристрої. Сьогодні за допомогою електронної комерції можна придбати майже все, що завгодно, що робить електронну комерцію висококонкурентною. Електронна комерція працює в декількох сегментах ринку, включаючи бізнес-бізнес, бізнес-споживач, споживач-споживач і споживач-бізнес.

Ведення журналів у системах електронної комерції важливо з кількох причин. По-перше, це допомагає виявити закономірності та тенденції в поведінці клієнтів, які можуть бути використані для покращення продуктивності систем електронної комерції. По-друге, це допомагає забезпечити безпеку та конфіденційність даних клієнтів, що має вирішальне значення для побудови довіри та лояльності серед клієнтів. По-третє, це допомагає забезпечити кращий користувацький досвід для клієнтів шляхом виявлення та усунення проблем у системі [2].

Таким чином, ведення журналів у системах електронної комерції є важливим аспектом діяльності підприємств електронної комерції. Важливо проводити дослідження для виявлення найкращих практик ведення журналів та аналізу даних, зібраних з них, щоб покращити ефективність, безпеку та якість обслуговування клієнтів.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Проблема що досліджується

У сучасному динамічному інформаційному ландшафті, де вирішення великих завдань електронної комерції визначається не тільки технологічними революціями, але й постійно зростаючими очікуваннями споживачів, проблематика управління журналами стає насущною. З кожним днем компанії все більше відчувають необхідність вдосконалення систем журналювання, які стали ключовим елементом розвитку електронної комерції.

Сучасний пейзаж електронної комерції зазнає впливу не тільки стрімкого технологічного прогресу, але й викликів, пов'язаних із зростанням кількості цифрових транзакцій, потреби впровадження високих стандартів безпеки, а також очікуванням клієнтів на надійність та швидкість обслуговування [3]. Ці аспекти формують актуальні питання, які висуває дослідження продуктивності розподілених систем журналювання з відкритим вихідним кодом.

У цьому контексті, розгляд проблем управління журналами набуває особливого значення, враховуючи не тільки сучасні реалії електронної комерції, але й високі вимоги до ефективності, безпеки та користувацького досвіду. Таким чином, дослідження спрямоване на розуміння та вирішення викликів, які постають перед підприємствами у сучасному цифровому середовищі [4].

1.2 Розробка технічного завдання кваліфікаційної роботи

Основною метою дослідження – є порівняльний аналіз продуктивності та ефективності платформ журналювання з відкритим вихідним кодом - Elasticsearch, Splunk та Grafana Loki. Ключовими критеріями оцінювання обрано масштабованість рішень, швидкодію індексації та пошукових запитів, а також ефективність використання інфраструктурних ресурсів для зберігання все зростаючих обсягів журнальних даних. Метою є виявлення можливостей та

обмежень даних платформ для побудови високопродуктивних систем централізованого збору, моніторингу та аналізу журналів подій в інформаційних системах різного масштабу.

Розглянемо завдання, розбивши його на конкретні етапи, які необхідно виконати під час проведення магістерського дослідження, виключаючи аспекти теоретичного матеріалу:

– визначення критеріїв оцінювання що відповідає певним вимогам до проведення магістерського дослідження;
– створити аккаунти на обраних системах збору інформації;
– підготовка даних, а саме збір і підготовка великих обсягів журнальних даних для подальшого використання в тестуванні платформ;
– занотувати процес розробки системи у текстовій записці магістерського дослідження;
– розгортання платформ - встановлення та налаштування Elasticsearch, Splunk та Grafana Loki в тестовому середовищі та перевірка сумісності з різними інфраструктурними середовищами;
– тестування продуктивності запису журналів, тестування пошукових запитів: - визначення часу виконання пошукових запитів різної складності;
– тестування можливостей візуалізації та моніторингу кожної з досліджуваних систем;
– обґрунтувати висновки, які виникають на основі проведеного магістерського дослідження.

Після формулювання технічного завдання для магістерського дослідження, переходимо до вибору інструментів для його виконання, аналізу предметної області та підготовки до самого дослідження.

1.3 Засоби що буде використано для проведення дослідження

Проведення даного дослідження виконуватиметься на мові програмування Java [5] з використанням середовища розробки IntelliJ IDEA [6]. Для реалізації тестового стенду будуть задіяні наступні ключові бібліотеки:

SLF4J – відкрита бібліотека журналювання для запису журналів подій з програм Java [7]. Використовуватиметься для генерації тестових.

Lombok - це бібліотека для Java, яка автоматизує підключення стандартного коду до проекту за допомогою анотацій [8]. Використовуватиметься для спрощення роботи з POJO (Plain Old Java Objects) та зменшення масштабу коду, що підтримується.

JMX (Java Management Extensions) – технологія моніторингу та керування Java додатками [9]. Застосовуватиметься для збору метрик продуктивності систем обробки журналів.

Elasticsearch – система для зберігання та аналізу великих обсягів даних [10]. Використовуватиметься як одна з систем для збору та обробки журналів.

Splunk – платформа для аналізу та візуалізації даних, включаючи журнали [11]. Використовуватиметься для обробки журналів у досліджуваній системі.

Grafana Loki – система для збору та обробки журналів з використанням мікросервісів [12]. Використовуватиметься як інша система для збору та аналізу журналів.

Java Microbenchmarking Harness (JMH) – бібліотека для вимірювання продуктивності Java-коду [13]. Використовуватиметься для оцінки продуктивності розробленого програмного стенду та аналізу вимірювань.

Docker - це відкрите програмне забезпечення для автоматизації розгортання, управління та запуску програм у контейнерах [14]. Використовуватиметься для локального розгортання сервісів які досліджуються.

Docker Compose - це інструмент для оркестрації багатоконтейнерних Docker додатків. Він дозволяє визначати та запускати комплексні додатки, які складаються з декількох контейнерів, в одному файлі конфігурації. Docker Compose спрощує управління додатками, що складаються з різних компонентів, таких як веб-сервери, бази даних, служби кешування тощо, та дозволяє запускати їх разом з одного місця за допомогою одного командного рядка [15]. Використовуватиметься для оркестрації досліджувальних сервісів.

2 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

2.1 Аналіз сфери моделей збору інформації

Збільшення обсягів даних журналів подій, що генеруються в сучасних інформаційних системах, вимагає ефективних рішень для їх збору, зберігання та аналізу. Це має велике значення для моніторингу роботи систем, виявлення та усунення неполадок, забезпечення безперервності бізнесу та відповідності вимогам нормативних документів. Проте реалізація систем централізованого журналювання може бути складним та довготривалим процесом.

Існують різні підходи та моделі організації збору журнальних даних для ефективного збору та обробки даних, що є критично важливими для моніторингу роботи сучасних інформаційних систем та додатків

Централізована модель передбачає збір усієї інформації в єдине сховище. Переваги - простота організації, можливість кореляції подій з усієї IT-інфраструктури. Недоліки - обмеження масштабованості, ускладнення забезпечення надійності.

Децентралізована модель базується на локальних сховищах журналів для кожного додатку або сервісу. Плюси - гнучкість, масштабованість, стійкість до збоїв. Мінуси - складність аналізу даних з різних джерел.

Гібридна модель поєднує центральне сховище журналів та локальні бази подій. Локальні журнали використовуються для моніторингу окремих сервісів, а глобальна БД - для системного аналізу та аудиту.

Для великих розподілених інфраструктур доцільно застосовувати мікросервісну модель збору журналів. Кожен сервіс виконує локальне журналювання з наступною консолідацією даних за допомогою спеціалізованої підсистеми централізованої обробки журналів.

Отже, існує компроміс між надійністю та масштабованістю локальних сховищ журналів і зручністю централізованої моделі для аналітики. Вибір

оптимальної стратегії залежить від особливостей інформаційної системи та поставлених цілей моніторингу.

2.2 Головні відомості про системи збору інформації

Основними компонентами сучасних систем збору і аналізу журнальних даних є агенти журналювання, системи централізованого збору журналів, сховища даних журналів, засоби аналітичної обробки даних.

Агенти журналювання – це програмні модулі, що вбудовуються в додатки та інфраструктурні компоненти для генерування журналів подій та помилок в стандартизованих форматах (JSON, XML, CSV тощо).

Системами централізованого збору журналів є спеціалізоване ПЗ для агрегації журнальних записів з багатьох джерел, їх обробки, зберігання та подальшого аналізу. Приклади: ELK Stack, Splunk, Graylog, Datadog.

Сховища даних журналів – це системи керування базами даних, оптимізовані для прийому, зберігання та обробки великих обсягів слабко-структурованих даних - журналів подій та повідомлень. Одними з найбільш відомих є Elasticsearch, InfluxDB, Cassandra [16].

Засоби аналітичної обробки даних. Програмне забезпечення для візуалізації, аналізу журналів, виявлення аномалій та причинно-наслідкових зв'язків між подіями. Приклади: Kibana, Grafana [17].

2.3 Аналіз ключових функцій основних інструментів збору інформації

Існує велика кількість рішень з відкритим вихідним кодом. Вибір оптимального варіанту залежить від масштабованості, продуктивності та вартості системи. Ці чинники впливають на загальну ефективність та цінність розгорнутої системи журналювання для організації.

Взявши основні інструменти збору інформації ми проаналізували які ключові функції в них є найбільш загальні і які оригінальні. Для аналізу були розглянуті наступні інструменти [18]:

- Sematext Logs;
- Splunk;
- Sumo Logic;
- SolarWinds PaperTrail;
- SolarWinds Loggly;
- ManageEngine EventLog Analyzer;
- Datadog;
- Dynatrace;
- Mezmo (Formerly LogDNA);
- Logz.io;
- Logentries (now Rapid7 InsightOps);
- Scalyr;
- Elasticsearch, Logstash and Kibana (ELK stack or Elastic Stack);
- Graylog;
- GoAccess;
- Grafana Loki;
- Systemd Journal;
- Logstash;
- rsyslog;
- syslog-ng;
- Fluentd;
- Filebeat;
- Logagent.

Були визначені загальні та оригінальні ключові функції.

Загальні ключові функції:

– багато інструментів підтримують відкриті стандарти, такі як syslog для збору журналів;
– більшість інструментів пропонують можливості збору, розбору/обробки, зберігання, пошуку та візуалізації журналів;
– зберігання журналів зазвичай базується на Elasticsearch або власному сховищі, створеному для журналів;
– ціноутворення часто включає безкоштовний рівень, а потім платні рівні, що залежать від обсягу та періоду зберігання.

Оригінальні ключові функції:

– Sematext Logs пропонує автоматичне виявлення журналів і сервісів, доступ до API Elasticsearch і контроль доступу на основі ролей;
– SolarWinds Papertrail має простий інтерфейс для пошуку, вбудоване архівування;
– Datadog пропонує необмежені можливості збору та архівування;
– Logz.io надає готові панелі інструментів та додатки Kibana;
– Scalyr використовує власне колоночне сховище даних замість індексації.

У цьому дослідженні проводиться порівняльний аналіз трьох провідних інструментів управління журналами з відкритим вихідним кодом - Elasticsearch, Splunk і Grafana Loki. Ці три рішення представляють різні підходи до управління зберіганням та аналізом журналів в масштабах. Оцінюючи ключові показники, пов'язані з методами зберігання, можливостями масштабування та тестами продуктивності, це дослідження має на меті надати рекомендації щодо вибору оптимальної архітектури управління журналами на основі вимог додатків та інфраструктурних обмежень.

Elasticsearch забезпечує розподілений документ-орієнтований підхід, який використовує інвертовані індекси Lucene та шардинг для масштабованого повнотекстового пошуку даних журналів. Splunk використовує власну схему з

різними рівнями зберігання для стиснення та оптимізації зберігання і пошуку журналів. Loki пропонує унікальну архітектуру на основі міток, що використовує обробку в пам'яті та зберігання об'єктів. Порівняння цих систем може виявити компроміси у вартості, складності, можливостях зберігання та гнучкості запитів.

Дослідження порівняльної ефективності даних платформ дозволить обрати найбільш підходящу з урахування особливостей інформаційних систем та інфраструктури. Важливими критеріями оцінки є продуктивність запису та пошуку в журналах, масштабованість нарощення ресурсів, латентність доступу до даних різноманітної давнини. Також актуальним є аналіз гнучкості систем та можливостей інтеграції з хмарними сервісами та контейнеризованими середовищами.

Інший важливий аспект – це інструменти та технології, що використовуються для збору та обробки журнальних даних. На ринку існує широкий вибір засобів, таких як Beats, Logstash, rsyslog, Fluentd та інші. Вибір оптимального інструментарію залежить від масштабу системи, складності інфраструктури, форматів журналів, вимог до латентності тощо.

Також важливу роль відіграє інтеграція з обраною централізованою системою журналювання, наприклад, Elasticsearch, Splunk чи Loki. Необхідно забезпечити сумісність у плані форматів, протоколів, методів парсингу та зберігання даних. Якість налаштування конвеєрів обробки журналів має значення для ефективного аналізу та моніторингу.

Для різних класів інформаційних систем можуть бути різні вимоги до систем журналювання. Наприклад, хмарні додатки потребують масштабованості та низької латентності. Критичні для бізнесу системи вимагатимуть високої надійності та стійкості. А для систем машинного навчання важливі можливості глибинного аналізу даних.

На додаток, важливим аспектом є масштабованість обраних для дослідження платформ - Elasticsearch, Splunk та Grafana Loki. Системи повинні бути здатні ефективно обробляти зростаючі обсяги даних журналів без втрати продуктивності.

Аналіз має включати оцінку пропускної спроможності запису даних, можливостей нарощування ресурсів кластерів, стійкості до збільшення навантаження при обробці аналітичних запитів в реальному часі.

Також варто дослідити ефективність використання інфраструктурних ресурсів та методи стиснення даних обраними системами журналювання.

Комплексне тестування дозволить виявити можливості та обмеження платформ при масштабуванні для потреб великих організацій.

2.3.1 Elasticsearch

Elasticsearch - це розподілена пошуково-аналітична система, яка є частиною Elastic Stack, що також включає Logstash та Kibana. Він забезпечує пошук та аналітику різних типів даних, таких як структурований або неструктурований текст, числові дані та геопросторові дані, майже в режимі реального часу. Elasticsearch має високу масштабованість і може паралельно обробляти великі обсяги даних, що робить його придатним для різноманітних випадків використання [19]. Elasticsearch зазвичай використовується для повнотекстового пошуку в журналах, аналітики безпеки, бізнес-аналітики та оперативної розвідки. Він пропонує RESTful API для взаємодії з пошуковою системою і підтримує різні функції, такі як повнотекстовий пошук, фразовий пошук і агрегації. Крім того, його можна використовувати для пошуку додатків, аналітики безпеки та бізнес-аналітики. Elasticsearch - це розподілена пошуково-аналітична система з відкритим вихідним кодом, призначена для обробки великих наборів даних у режимі, близькому до реального часу. Він побудований на основі бібліотеки Apache Lucene і є сховищем даних без схем, орієнтованим на документи. Elasticsearch організовує дані в документи, які групуються в індекси. Він використовує інвертовані індекси

для ефективного пошуку і має розподілену архітектуру, що дозволяє швидко здійснювати пошук і аналіз великих обсягів даних. Elasticsearch є центральним компонентом Elastic Stack - набору інструментів з відкритим вихідним кодом для збору, збагачення, зберігання, аналізу та візуалізації даних. Інвертовані індекси зображено на рисунку 1.

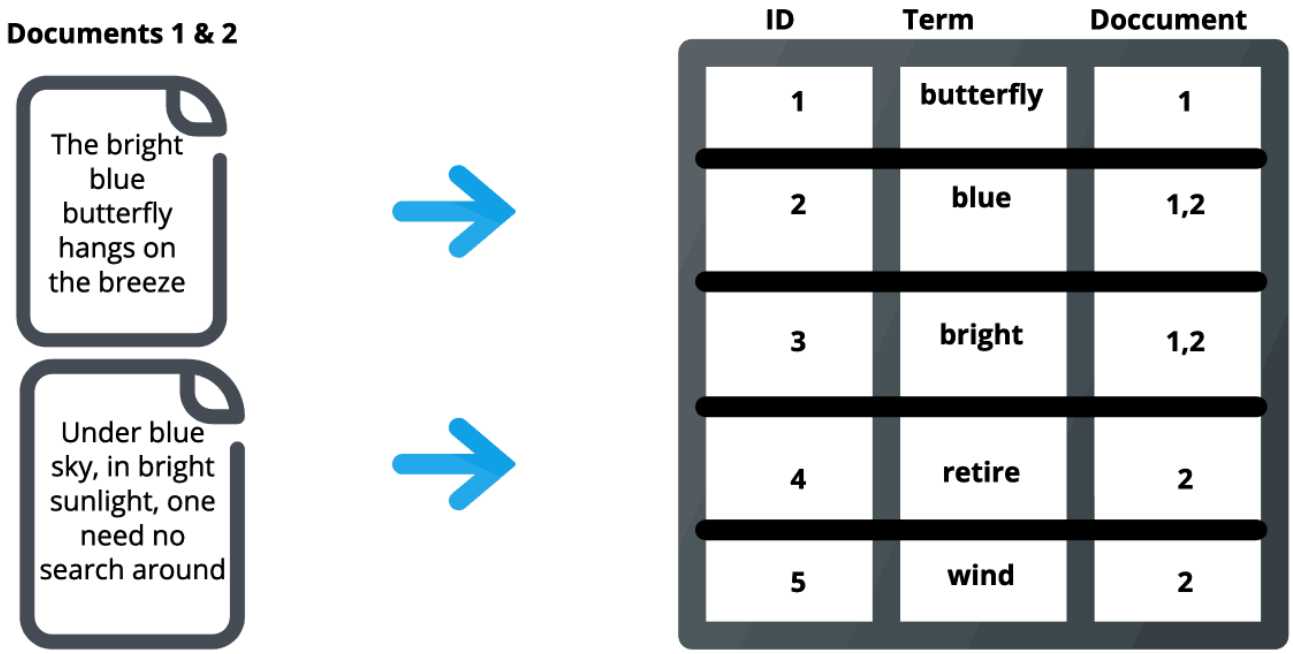


Рисунок 1. – Візуальне представлення інвертованого індексу (за даними [19])

Стек ELK, до якого входять Elasticsearch, Logstash і Kibana, використовується для аналізу журналів, пошуку документів, управління інформацією та подіями безпеки (SIEM) і спостережливості. Elasticsearch, як частина стеку ELK, дозволяє агрегувати журнали з різних систем і додатків, аналізувати ці журнали і створювати візуалізації для моніторингу та усунення дефектів. Стек ELK використовується для вирішення широкого спектру завдань, включаючи аналітику журналів, пошук документів, управління інформацією та подіями безпеки (SIEM) і спостережливість. Він забезпечує механізм пошуку та аналітики, отримання даних і візуалізацію.

Elasticsearch дуже корисний для додатків електронної комерції. Він забезпечує швидкі та ефективні результати пошуку, що може покращити користувацький досвід та підвищити рівень задоволеності клієнтів. За допомогою Elasticsearch веб-сайти електронної комерції можуть швидко надавати релевантні результати пошуку навіть у великих обсягах даних. Elasticsearch можна використовувати для персоналізації результатів пошуку на основі історії пошуку, вподобань і поведінки клієнта. Це може допомогти веб-сайтам електронної комерції забезпечити більш персоналізований досвід для своїх клієнтів. Elasticsearch також пропонує персоналізацію на основі машинного навчання та семантичний пошук, що може ще більше покращити досвід пошуку на веб-сайтах електронної комерції [20].

Elasticsearch пропонує широкий спектр функцій і переваг, включаючи високу продуктивність, масштабованість, безкоштовні інструменти та плагіни, роботу в режимі, близькому до реального часу, і легку розробку додатків. Elasticsearch є безкоштовним і з відкритим вихідним кодом, що робить його доступним для всіх. Elasticsearch використовується в широкому спектрі додатків, включаючи метрики інфраструктури і моніторинг контейнерів, ведення журналів і аналіз журналів, моніторинг продуктивності додатків, аналіз і візуалізацію геопросторових даних, безпеку і бізнес-аналітику, а також вилучення і об'єднання публічних даних.

2.3.2 Splunk

Splunk - це потужна платформа для аналізу даних, яка широко використовується для аналізу журналів, розвідки безпеки, бізнес-аналітики та оперативної розвідки. Вона пропонує цілий ряд функцій, включаючи збір даних, пошук і аналіз, візуалізацію та машинне навчання. Splunk широко використовується в різних галузях, включаючи електронну комерцію, де він допомагає поліпшити показники безвідмовної роботи, підвищити рівень безпеки і збільшити продажі. Він забезпечує багатоканальний моніторинг досвіду, активно покращуючи час безвідмовної роботи та продуктивність веб-сторінок, API та

транзакцій, щоб забезпечити кращий цифровий досвід при кожній взаємодії з клієнтом. Splunk також допомагає підтримувати кібербезпеку в роздрібній торгівлі, покращуючи загальну продуктивність і працездатність інфраструктури магазинів, а також забезпечуючи надійний і безпечний досвід роботи на всіх каналах для клієнтів і співробітників [21].

У секторі електронної комерції Splunk використовується для підтримки 100% безвідмовної роботи, забезпечення бездоганного досвіду покупок і усунення проблем набагато швидше, ніж раніше. Він допомагає підтримувати безперебійну роботу електронної комерції, забезпечуючи продуктивність, стабільність і гнучкість, особливо під час міграції в хмару. Splunk також забезпечує видимість у реальному часі, прискорює усунення несправностей, дозволяє швидко виявити першопричину проблем та усунути їх у реальному часі [21].

Основні можливості. Splunk пропонує цілий ряд функцій і можливостей, серед яких:

- збір даних – може збирати дані з різних джерел, таких як файли журналів, пристрої Інтернету речей та журнали додатків;
- пошук і аналіз – дозволяє користувачам шукати, фільтрувати та аналізувати дані за допомогою потужної мови обробки пошуку (SPL);
- візуалізація – дозволяє користувачам створювати візуально привабливі дашборди та звіти, щоб отримувати інформацію та приймати обґрунтовані рішення;
- машинне навчання – підтримує аналітику на основі машинного навчання, що дозволяє організаціям прогнозувати результати та відкривати нові можливості;
- безпека – пропонує розширені функції безпеки, такі як виявлення загроз, управління інцидентами та моніторинг відповідності нормативним вимогам.

Варіанти використання Splunk в електронній комерції включають отримання інформації про поведінку споживачів, формулювання бізнес-стратегій та аналіз поведінки споживачів, щоб зрозуміти, як змінюється реакція на пропозицію та дохід від замовлень. Він також використовується для A/B-тестування, аналізу реакції клієнтів на різні пропозиції та розуміння вподобань клієнтів щодо покупок в магазині чи онлайн.

2.3.3 Grafana Loki

Grafana Loki - це горизонтально масштабована, високодоступна, багатокористувацька система агрегації журналів, натхненна Prometheus. Вона відрізняється від Prometheus тим, що фокусується на журналах, а не на метриках, і збирає журнали за допомогою push, а не pull. Loki розроблена, щоб бути дуже економічно ефективною та масштабованою. На відміну від інших систем журналювання, Loki не індексує вміст журналів, а лише індексує метадані про журнали у вигляді набору міток для кожного потоку журналів. Цей унікальний підхід робить Loki дуже економічно ефективною та масштабованою системою [22]. На рисунку 2 зображений стек журналів Grafana Loki.

Рисунок 2. – Стек журналів Loki (за даними [22])

Основні можливості:

– поглинання даних – збирає журнали за допомогою push, перетворюючи їх на потоки, додаючи мітки, і надсилає потоки до Loki через HTTP API;
– пошук та аналіз – Loki дозволяє користувачам запитувати журнали з командного рядка, за допомогою LogCLI або безпосередньо через API Loki;
– візуалізація – інтегрується з Grafana, Mimir та Tempo, забезпечуючи повний стек спостережуваності та безперешкодну кореляцію між журналами, метриками та трасуванням.

Loki пропонує кілька нетехнічних переваг, таких як зниження витрат, оптимізація операцій та створення кращих команд. Це дозволяє організаціям уникати роботи з гарячими та холодними індексами, управлінням життєвим циклом та одноразовими некромантичними процесами для реанімації старих даних [22]. Це може призвести до значної економії коштів і підвищення операційної ефективності

У секторі електронної комерції Grafana Loki відіграє вирішальну роль у підтримці безвідмовної роботи, усуненні дефектів і отриманні цінної інформації про поведінку споживачів для поліпшення загального клієнтського досвіду і збільшення продажів. Вона допомагає підтримувати безперебійну роботу електронної комерції, забезпечуючи продуктивність, стабільність і гнучкість, особливо під час міграції в хмару. Крім того, Grafana Loki забезпечує видимість в режимі реального часу, прискорює пошук і усунення дефектів, дозволяє швидко виявити першопричину проблем і усунути їх в режимі реального часу.

3 ПІДГОТОВКА ДО ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

3.1 Визначення тестових сценаріїв збору інформації

Вибір моделей збору інформації має вирішальне значення для успіху експерименту. Моделі збору інформації повинні бути розроблені для збору даних, які є репрезентативними для даних журналів. Моделі повинні бути розроблені для збору даних різного розміру, починаючи від малих і закінчуючи великими наборами даних.

3.1.2 Згенерувати різні типи подій

Потрібно згенерувати типи подій типові для електронної комерції, такими подіями є покупки, перегляди товарів, додавання до кошика, та оплати. Для цього нам потрібно створити моделі тестових даних, такі як користувач, продукт, та модель події журналу. На рисунках 3, 4 та 5 зображені ці моделі.

```
6   @Data  8 usages
7   @AllArgsConstructor
8   public class User {
9       String username;
10  }
```

Рисунок 3. – Кодова репрезентація моделі користувача (виконано самостійно)

```java
@Data  8 usages
@AllArgsConstructor
public class Product {
    String name;
    double price;
}
```

Рисунок 4. – Кодова репрезентація моделі продукту (виконано самостійно)

```java
@Data  10 usages
public class LogEvent {

    Date timestamp;
    String eventType;
    User user;
    Product product;

    public LogEvent(String eventType, User user, Product product) {
        this.timestamp = new Date();
        this.eventType = eventType;
        this.user = user;
        this.product = product;
    }

}
```

Рисунок 5. – Кодова репрезентація моделі події журналу (виконано самостійно)

Далі нам знадобиться сервіс для створювання моделей подій, таких як «Перегляд товару», «Додавання товару до кошика», «Покупка товару» та «Оплата

товару». Це полегшить подальшу генерацію подій журналювання. На рисунку 6 зображено такий сервіс.

```
@Service  no usages
public class LogCreatorService {

    public static LogEvent generateProductViewEvent(User user, Product product) {
        return new LogEvent( eventType: "ProductView", user, product);
    }

    public static LogEvent generateAddToCartEvent(User user, Product product) {
        return new LogEvent( eventType: "AddToCart", user, product);
    }

    public static LogEvent generatePurchaseEvent(User user, Product product) {
        return new LogEvent( eventType: "Purchase", user, product);
    }

    public static LogEvent generatePaymentEvent(User user, Product product) {
        return new LogEvent( eventType: "Payment", user, product);
    }

}
```

Рисунок 6. – Кодова репрезентація сервіс для створювання моделей подій (виконано самостійно)

3.1.3 Збереження з використанням різних методів

Нам потрібно зберегти події журналів в Elasticsearch, Splunk і Grafana Loki, використовуючи різні методи зберігання та індексації. Для того щоб застосунки систем журналювання не конфліктували між собою, та при цьому дати їм змогу користуватися одними даними, було створено модульну архітектуру на Java з використанням Gradle. Який включає в себе наступні модулі:

– головний модуль – доступний модуль для всіх підмодулів;
– модуль для Elasticsearch;
– модуль для Splunk;
– модуль для Grafana Loki.

На рисунку 7 зображено структуру проекту.

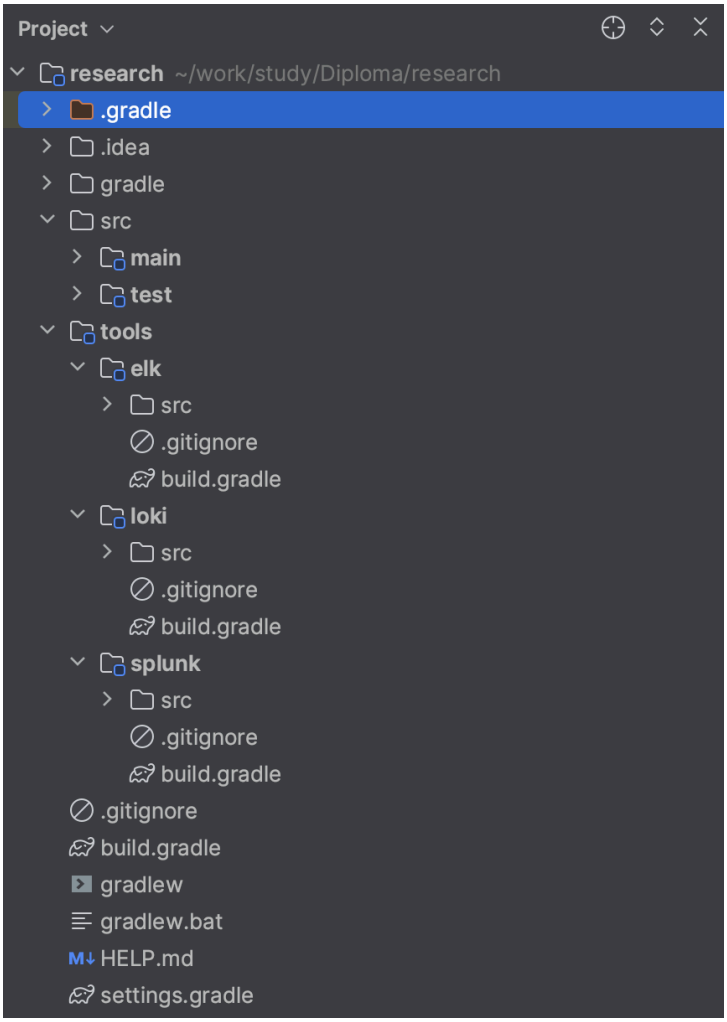
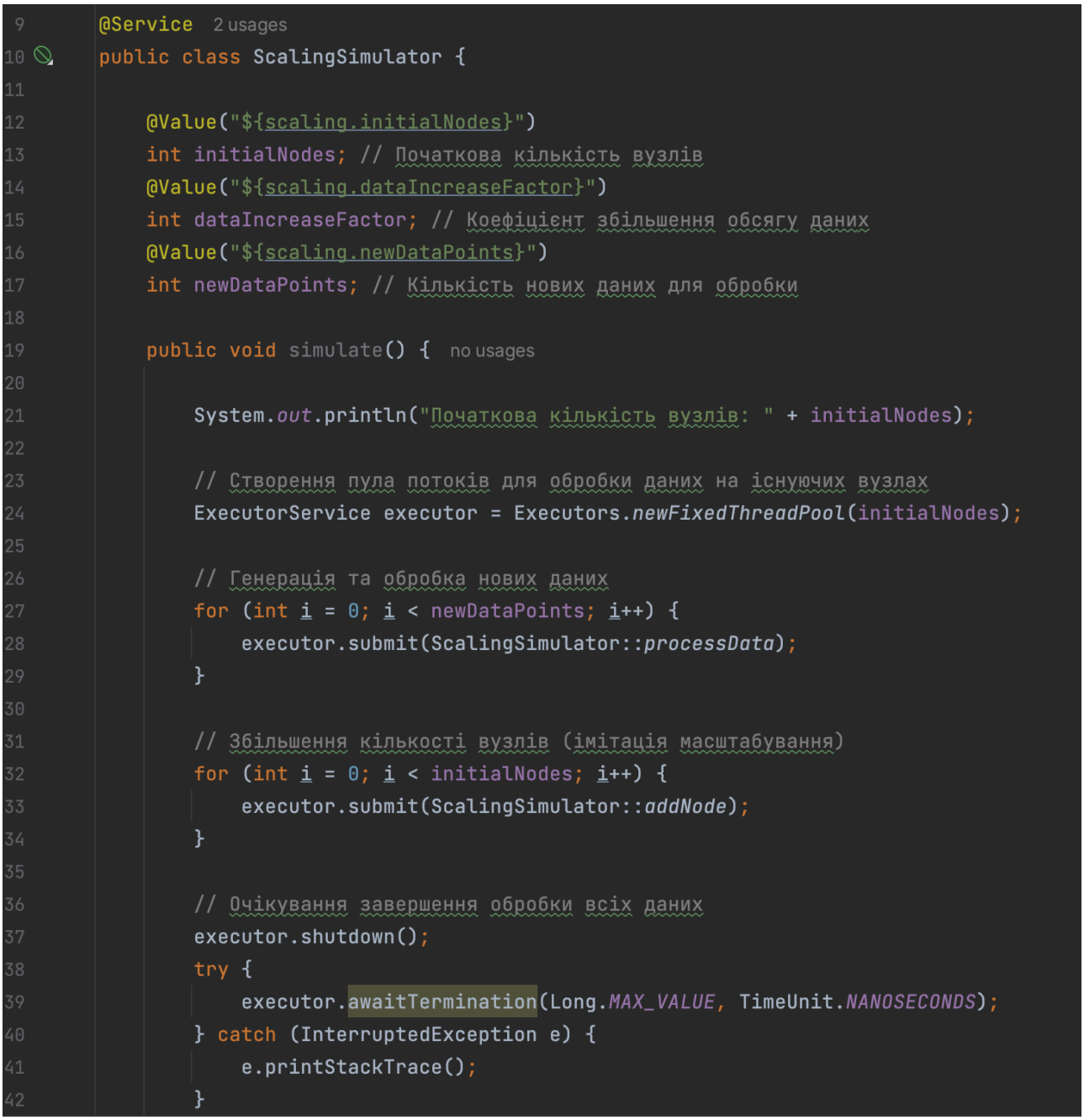


Рисунок 7. – Структура проекту (виконано самостійно)

3.1.4 Тестування можливостями масштабування

Для імітування масштабування ми можемо використати багатопоточність та додавання нових ресурсів (вузлів) для обробки збільшеного обсягу даних в Java, можна використовувати многозадачність та многонитковість. На рисунку 8 зображено приклад реалізації такого імітатора.



```java
@Service  2 usages
public class ScalingSimulator {

    @Value("${scaling.initialNodes}")
    int initialNodes; // Початкова кількість вузлів
    @Value("${scaling.dataIncreaseFactor}")
    int dataIncreaseFactor; // Коефіцієнт збільшення обсягу даних
    @Value("${scaling.newDataPoints}")
    int newDataPoints; // Кількість нових даних для обробки

    public void simulate() {  no usages

        System.out.println("Початкова кількість вузлів: " + initialNodes);

        // Створення пула потоків для обробки даних на існуючих вузлах
        ExecutorService executor = Executors.newFixedThreadPool(initialNodes);

        // Генерація та обробка нових даних
        for (int i = 0; i < newDataPoints; i++) {
            executor.submit(ScalingSimulator::processData);
        }

        // Збільшення кількості вузлів (імітація масштабування)
        for (int i = 0; i < initialNodes; i++) {
            executor.submit(ScalingSimulator::addNode);
        }

        // Очікування завершення обробки всіх даних
        executor.shutdown();
        try {
            executor.awaitTermination(Long.MAX_VALUE, TimeUnit.NANOSECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

Рисунок 8. – Приклад реалізації імітації масштабування (виконано самостійно)

3.1.5 Вимірювання час роботи та продуктивність систем журналювання

Для того щоб виміряти час роботи та продуктивність логерів, ми можемо скористуватися інструментами для профілювання та вимірювання часу виконання

в Java. Для вимірювання часу виконання, може слугувати Java Microbenchmarking Harness (JMH). На рисунку 9 зображено приклад реалізації такого коду.

```java
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
@State(Scope.Benchmark)
public class LoggerBenchmark {

    private static final int ITERATIONS = 100000;  no usages

    @Benchmark  no usages
    public void measureLoggerPerformance() {
        // Виклик методів журналювання
    }

    public static void main(String[] args) throws RunnerException {
        Options options = new OptionsBuilder()
                .include(LoggerBenchmark.class.getSimpleName())
                .forks( value: 1)
                .build();

        new Runner(options).run();
    }
}
```

Рисунок 9. – Приклад реалізації JMH (виконано самостійно)

3.2 Кінцеве бачення реалізації задач дослідження

Наше дослідження буде спрямоване на створення та аналіз трьох унікальних реалізацій системи управління журналами з відкритим вихідним кодом, зокрема Elasticsearch, Splunk і Grafana Loki.

Кожна із цих реалізацій відображатиме відмінний підхід до проблеми зберігання та аналізу журналів в контексті масштабованості. Ми спроектуємо та

реалізуємо кожну систему, акцентуючи увагу на їхній архітектурі, методах зберігання та швидкодії обробки даних.

Під час дослідження розглядались ключові аспекти, такі як ефективність зберігання, можливості горизонтального масштабування та продуктивність у реальному часі. Це дозволить нам зрозуміти переваги та обмеження кожної системи та, в кінцевому підсумку, надати рекомендації щодо вибору оптимальної архітектури управління журналами відповідно до унікальних вимог додатків та інфраструктурних особливостей.

4 СТВОРЕННЯ ПРОГРАМНОЇ СИСТЕМИ ДЛЯ ДОСЛІДЖЕННЯ

4.1 Налаштування систем журналювання

У рамках дослідження було проведено налаштування та розгортання систем журналювання ELK Stack (Elasticsearch, Logstash, Kibana), Grafana Loki та Splunk, на платформі MacOS. Для спрощення розгортання та підвищення портативності було прийнято рішення використовувати технологію контейнеризації Docker.

Слід зазначити, що для системи Splunk офіційно не підтримується Docker-версія, сумісна з операційною системою MacOS. У зв'язку з цим, в рамках дослідження було використано Splunk Enterprise застосунок, рекомендоване розробниками рішення для розгортання Splunk на платформі MacOS.

4.1.1 Налаштування ELK Stack

Налаштування ELK Stack (Elasticsearch, Logstash, Kibana) було здійснено з використанням Docker-контейнерів, що забезпечило швидке та стандартизоване розгортання.

Першим кроком було розгортання Elasticsearch - розподіленої пошукової та аналітичної системи, яка забезпечує високу продуктивність та масштабованість при обробці великих обсягів даних. Для цього було використано офіційний Docker-образ Elasticsearch, налаштований за допомогою відповідних змінних середовища та конфігураційних файлів.

Наступним етапом було налаштування Logstash - потужного інструменту для збирання, трансформації та транспортування даних журналів. Логіка обробки журналів була визначена у конфігураційному файлі Logstash, який включав вказівки щодо вхідних джерел даних, фільтрів для їх трансформації та виходу для передачі оброблених даних в Elasticsearch.

Для візуалізації та аналізу зібраних журналів було розгорнуто Kibana - гнучку веб-платформу з відкритим вихідним кодом. Kibana була інтегрована з Elasticsearch

та дозволила створювати різноманітні візуалізації, діаграми та динамічні панелі моніторингу на основі зібраних даних журналів.

Для швидкого розгортання ELK Stack було створено docker-compose.yml файл з налаштуванням сервісів та їх залежностей.

```
version: "3"
services:
  setup:
    image: docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
    user: "0"
    command: >
      ...
    healthcheck:
      ...

  es01:
    depends_on:
      setup:
        condition: service_healthy
    image: docker.elastic.co/elasticsearch/elasticsearch:${STACK_VERSION}
    volumes:
      - certs:/usr/share/elasticsearch/config/certs
      - esdata01:/usr/share/elasticsearch/data
    ports:
      - ${ES_PORT}:9200
    environment:
      - node.name=es01
      - ...
    mem_limit: ${MEM_LIMIT}
    ulimits:
      memlock:
        soft: -1
        hard: -1
    healthcheck:
      ...

  kibana:
    depends_on:
      es01:
        condition: service_healthy
    image: docker.elastic.co/kibana/kibana:${STACK_VERSION}
    volumes:
      - certs:/usr/share/kibana/config/certs
      - kibanadata:/usr/share/kibana/data
```

```
    ports:
      - ${KIBANA_PORT}:5601
    environment:
      - SERVERNAME=kibana
      - ...
    mem_limit: ${MEM_LIMIT}
    healthcheck:
      ...

  logstash01:
    ...
    image: docker.elastic.co/logstash/logstash:${STACK_VERSION}
    labels:
      co.elastic.logs/module: logstash
    user: root
    volumes:
      ...
    environment:
      ...
    ports:
      ...

volumes:
  ...
```

Також, для налаштування logstash було створено logstash.conf файл де зазначені данні для отримання та відображення журналювання.

```
input {
  tcp {
    port => 8044
    codec => json_lines
  }
}
output {
  elasticsearch {
    index => "elkdemoindex"
    hosts => "${ELASTIC_HOSTS}"
```

```
    user => "${ELASTIC_USER}"
    password => "${ELASTIC_PASSWORD}"
    cacert=> "certs/ca/ca.crt"
  }
}
```

Після виконання команди **docker-compose up -d** ми отримуємо локальну версію ELK Stack.

```
docker % docker-compose up -d
[+] Running 1/1
[+] Running 5/5er_default  Created
0.1s
 ✔ Network docker_default         Created
0.1s
 ✔ Container docker-setup-1       Healthy
3.2s
 ✔ Container docker-es01-1        Healthy
25.7s
 ✔ Container docker-kibana-1      Healthy
57.1s
 ✔ Container docker-logstash01-1  Started
```

Тепер ми можемо перевірити сервіси відкривши їх в системному браузері.

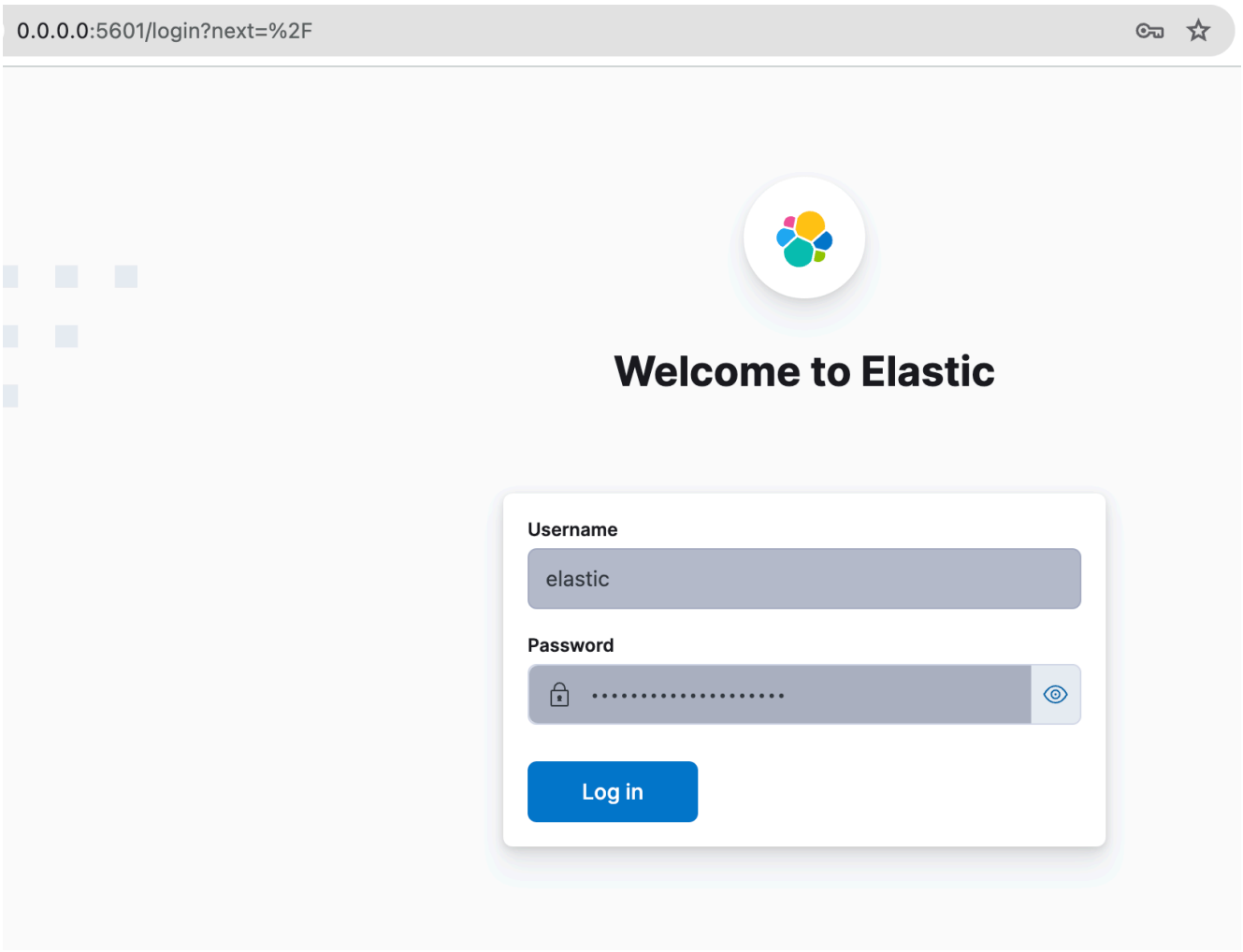


Рисунок 10. – Вхід в інтерфейс Kibana (виконано самостійно)

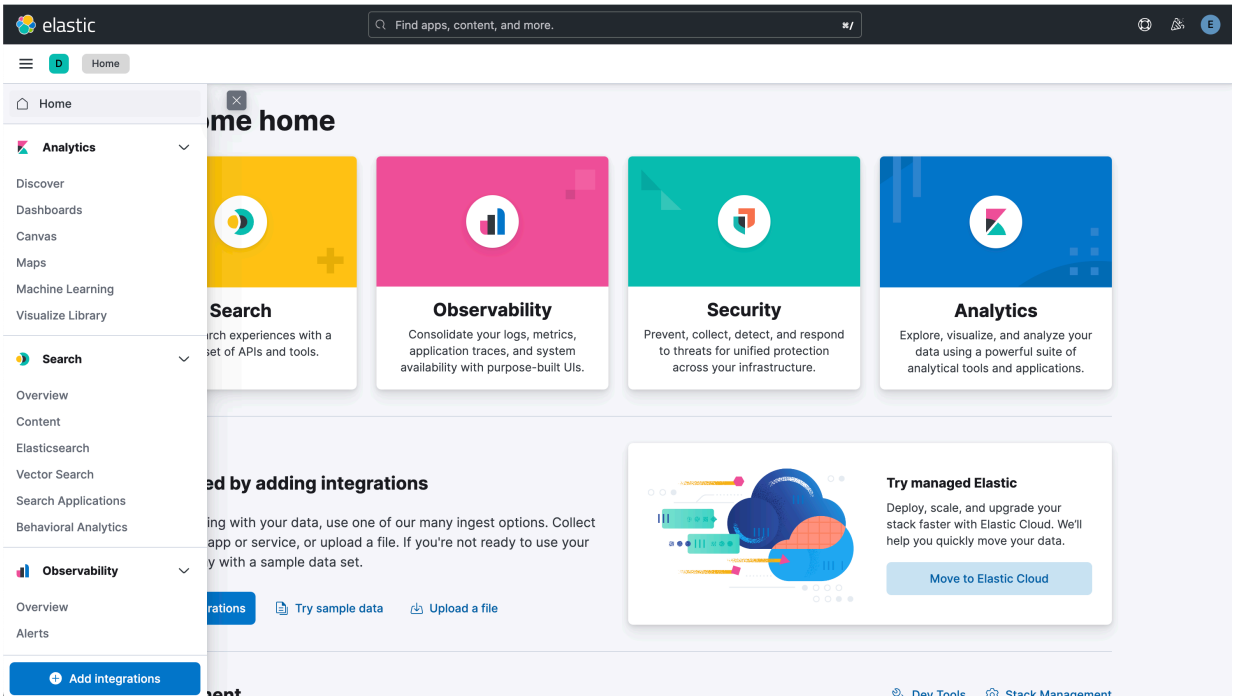


Рисунок 11. – Головна сторінка Kibana (виконано самостійно)

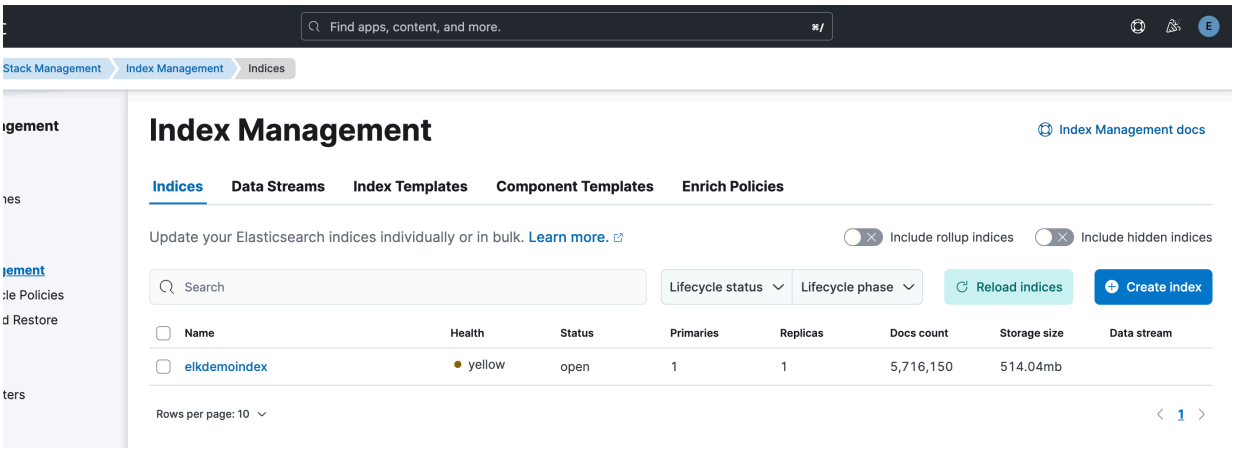


Рисунок 12. – Відображення налаштованого індексу в інтерфейсі Kibana (виконано самостійно)

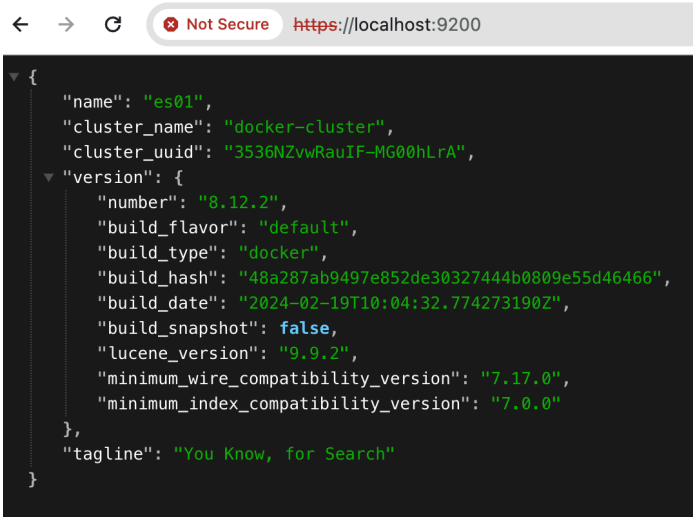


Рисунок 13. – Відкриття сервісу Elasticsearch (виконано самостійно)

Після успішного розгортання компонентів ELK Stack за допомогою Docker-контейнерів було підтверджено коректну взаємодію між цими сервісами, а також їх здатність взаємодіяти з зовнішніми сервісами та джерелами даних журналів. Таким чином, було забезпечено функціональність централізованого збирання, трансформації, зберігання та візуалізації даних журналів в рамках розгорнутої системи.

4.1.2 Налаштування Grafana Loki

Наступним етапом дослідження було налаштування та розгортання Grafana Loki.

На початковому етапі було розгорнуто Loki - систему журналювання, призначену для збирання, індексації та зберігання журналів. Офіційний Docker-образ Loki був налаштований за допомогою відповідних змінних середовища та конфігураційних файлів, які визначали параметри функціонування сервісу.

Після успішного розгортання Loki, було виконано налаштування та інтеграцію з Grafana - веб-платформою візуалізації даних. Офіційний Docker-образ Grafana був розгорнутий та інтегрований з Loki за допомогою спеціального плагіну, що дозволило візуалізувати дані журналів, отриманих з Loki.

Використання підходу з Docker-контейнерами забезпечило стандартизований та портативний процес розгортання системи централізованого журналювання Grafana Loki на платформі MacOS, уникаючи складнощів, пов'язаних з налаштуванням окремих компонентів вручну.

Для спрощення розгортання компонентів системи також було створено docker-compose.yml файл з налаштуванням сервісів.

```
version: "3"
networks:
  loki:

services:
  loki:
    image: grafana/loki:2.8.2
    ports:
      - "3100:3100"
    command: -config.file=/etc/loki/custom-config.yaml
    volumes:
      - ./custom-config.yaml:/etc/loki/custom-config.yaml
    networks:
      - loki

  grafana:
    environment:
      - GF_PATHS_PROVISIONING=/etc/grafana/provisioning
      - GF_AUTH_ANONYMOUS_ENABLED=true
      - GF_AUTH_ANONYMOUS_ORG_ROLE=Admin
    entrypoint:
      - sh
      - -euc
      - |
        mkdir -p /etc/grafana/provisioning/datasources
        cat <<EOF > /etc/grafana/provisioning/datasources/ds.yaml
        apiVersion: 1
```

```
        datasources:
        - name: Loki
          type: loki
          access: proxy
          orgId: 1
          url: http://loki:3100
          basicAuth: false
          isDefault: true
          version: 1
          editable: false
        EOF
        /run.sh
    image: grafana/grafana:latest
    ports:
      - "3000:3000"
    networks:
      - loki
```

Після виконання команди **docker-compose up -d** ми отримуємо локальну версію сервісів Grafana Loki.

```
docker % docker-compose up -d
[+] Running 0/0
[+] Running 3/3er_loki  Creating
0.0s
 ✔ Network docker_loki         Created
0.0s
 ✔ Container docker-grafana-1  Started
1.5s
 ✔ Container docker-loki-1     Started
```

Тепер ми можемо перевірити сервіси відкривши їх в системному браузері перейшовши за посиланням http://localhost:3000/.

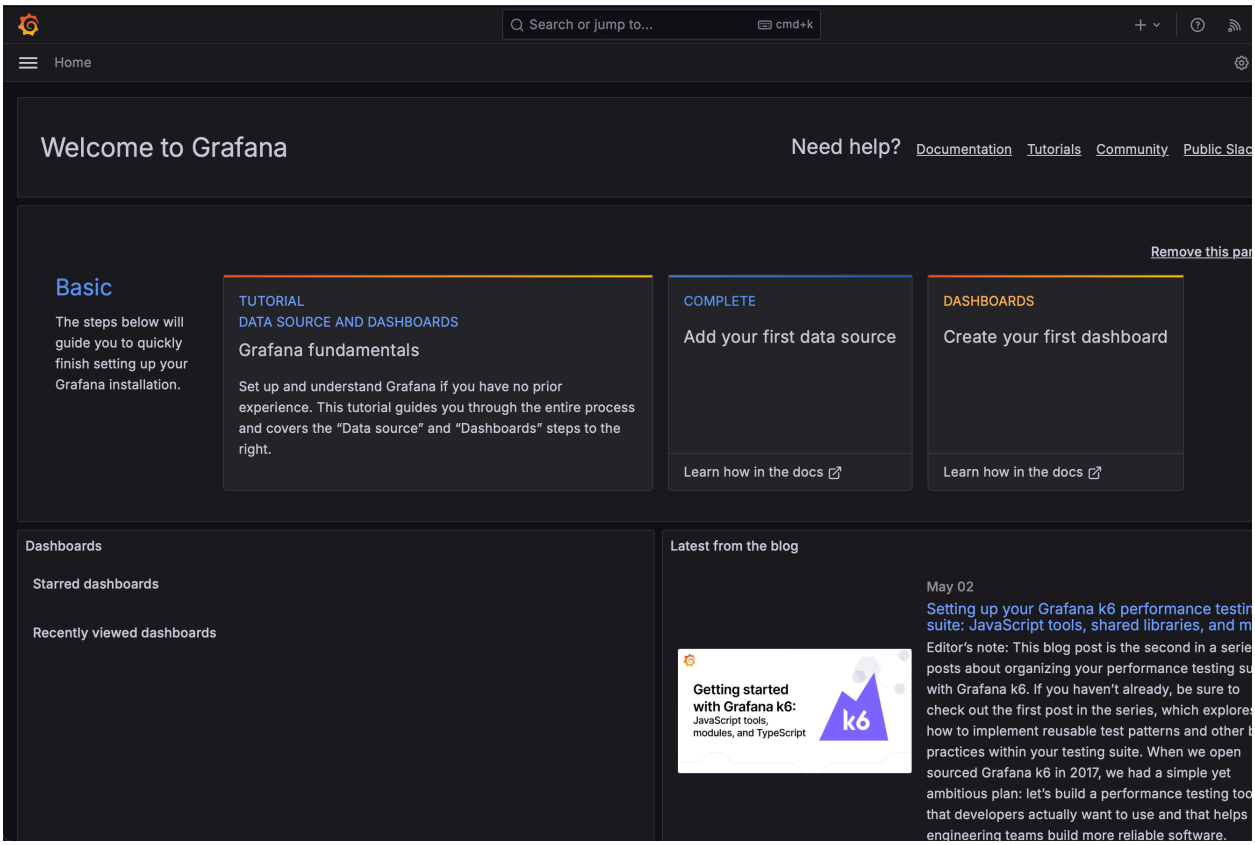


Рисунок 14. – Відкриття локального сервісу Grafana (виконано самостійно)

Перейшовши в меню Home → Explorer ми бачимо інтеграцію з Loki.

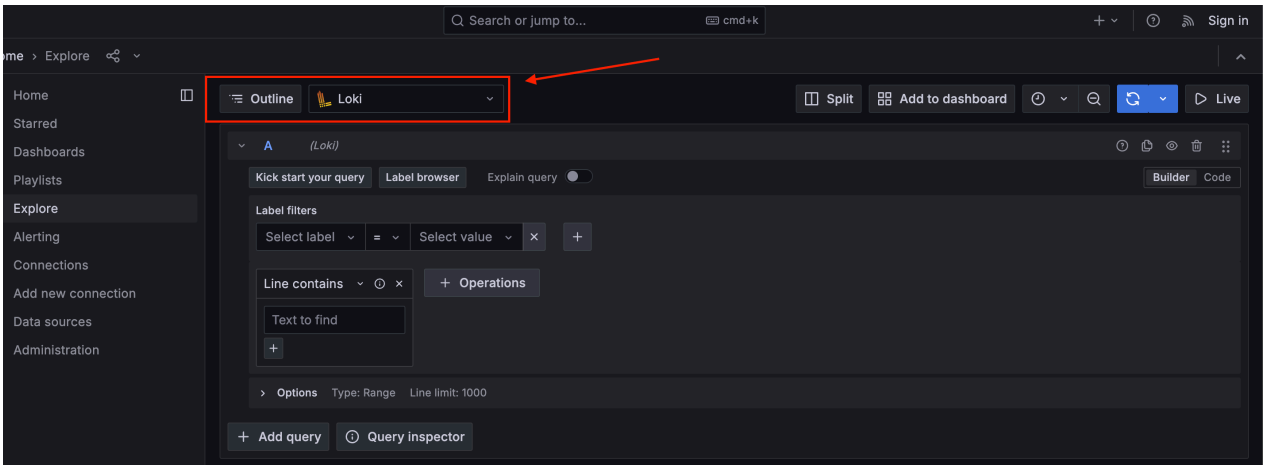


Рисунок 15. – Інтеграція з Loki (виконано самостійно)

Після успішного розгортання Grafana Loki за допомогою Docker-контейнерів було підтверджено коректну взаємодію між компонентами Loki та Grafana. Loki продемонстрував здатність збирати, індексувати та зберігати журнали з різних джерел даних, а Grafana забезпечила функціональність візуалізації та аналізу цих

журналів. Таким чином, розгорнута система Grafana Loki повністю забезпечила необхідні можливості для централізованого збирання, зберігання та візуалізації даних журналів в рамках єдиного рішення.

4.1.3 Налаштування Splunk

Для налаштування Splunk було використано рекомендований розробниками підхід із застосуванням Splunk застосунку, оскільки офіційної Docker-версії, сумісної з MacOS, не існує.

На початковому етапі було виконано завантаження та встановлення Splunk застосунку відповідно до інструкцій розробника. Після успішної інсталяції було проведено початкове налаштування Splunk, включно з створенням облікового запису адміністратора та визначенням основних параметрів функціонування системи.

Перейшовши за посиланням http://localhost:8000/ ми бачимо сторінку входу Splunk Enterprise.

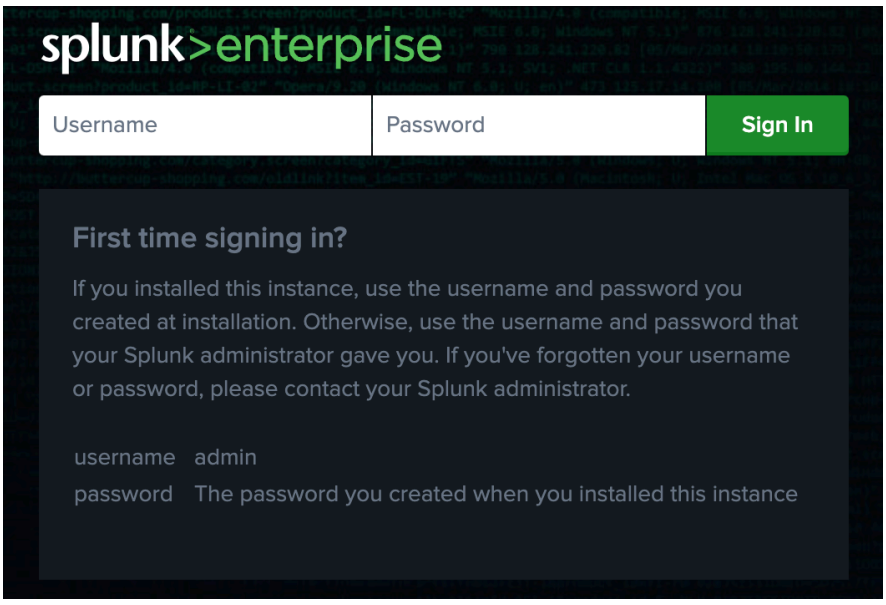


Рисунок 16. – Сторінка входу Splunk Enterprise (виконано самостійно)

Наступним кроком було налаштування джерел даних журналів, з яких Splunk повинен збирати та обробляти дані. Це включало налаштування моніторингу

певних каталогів файлової системи, мережевих портів, а також інтеграцію з різними сервісами та додатками за допомогою спеціальних модулів збору даних (data inputs).

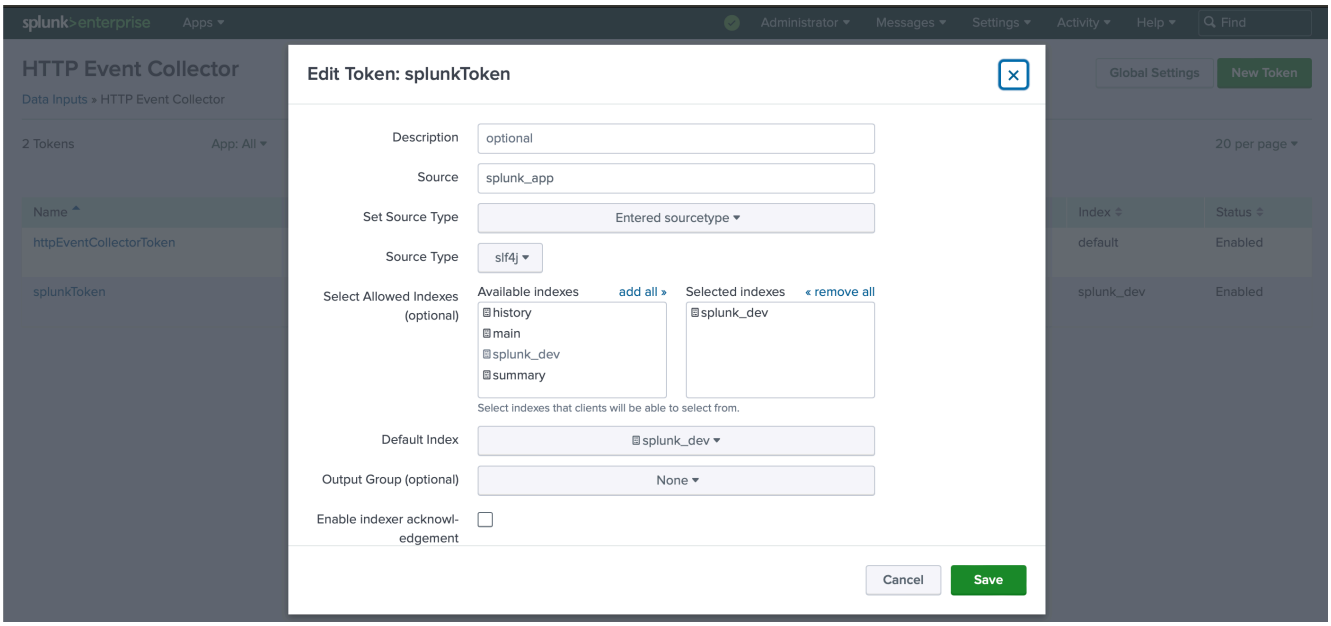


Рисунок 17. – Налаштування HTTP Event Collector токену (виконано самостійно)

Незважаючи на відсутність офіційної підтримки Docker-версії Splunk для MacOS, використання застосунку Splunk забезпечило розгортання повнофункціональної системи централізованого журналювання та моніторингу на цій платформі.

4.2 Створення та налаштування Java застосунку

Для проведення дослідження було розроблено мультимодульне Java-застосування на базі Spring Boot. Основною метою даного застосування було генерування журналів у різних форматах та їх відправка до досліджуваних систем централізованого журналювання.

Структура застосування складалася з одного головного модуля та трьох підмодулів, призначених для взаємодії з ELK Stack, Grafana Loki та Splunk відповідно.

У головному модулі були зібрані загальні залежності та утиліти, необхідні для роботи всього застосування. Зокрема, тут розміщувалися класи для генерування тестових даних та класи для проведення бенчмаркінгу продуктивності журналювання.

Кожен з підмодулів містить специфічну логіку для взаємодії з відповідною системою журналювання. Наприклад, підмодуль для ELK Stack включав налаштування Logstash для відправки журналів в Elasticsearch, а підмодуль для Grafana Loki – налаштування для збору журналів та їх передачі до Loki.

Дане застосування було розроблене з використанням найкращих практик Java та Spring Boot, що забезпечило гнучкість, модульність та легкість підтримки коду. Завдяки цьому було можливо швидко змінювати конфігурацію та налаштування для кожної системи журналювання, а також проводити тестування продуктивності в різних сценаріях навантаження.

4.2.1 Основний модуль (common)

Спочатку було розроблено файл Gradle збірника – build.gradle. Визначено основні плагіни.

```
lugins {
    id 'org.springframework.boot' version '3.3.0-SNAPSHOT'
    id 'io.spring.dependency-management' version '1.1.4'
    id 'java'
}
```

Наступним кроком було визначено основні залежності.

```
dependencies {
```

```
        implementation 'org.springframework.boot:spring-boot-starter'
        compileOnly 'org.projectlombok:lombok'
        annotationProcessor 'org.projectlombok:lombok'

    //    jmh
        implementation 'org.openjdk.jmh:jmh-core:1.37'
        annotationProcessor                'org.openjdk.jmh:jmh-generator-
annprocess:1.37'
        testImplementation 'org.openjdk.jmh:jmh-runner:1.37'
    }
```

Було створено основні моделі для відображення в журналах подій – *User*, *Product* та *LogEvent*, що включає в себе данні створення події, тип події, данні події та буде відображати в журналі деталі згенерованих подій.

```
    @ToString
    public class LogEvent {
        Date timestamp;
        String eventType;
        User user;
        Product product;

        public LogEvent(String eventType, User user, Product product) {
            this.timestamp = new Date();
            this.eventType = eventType;
            this.user = user;
            this.product = product;
        }
    }
```

LogCreatorServer включає в себе чотири методи для генерації простих подій, таких як *generateProductViewEvent*, *generateAddToCartEvent*, *generatePurchaseEvent*, *generatePaymentEvent*.

```
    public class LogCreatorService {
        public static LogEvent generateProductViewEvent(User user, Product
product) {
```

```
            return new LogEvent("ProductView", user, product);
        }

        public static LogEvent generateAddToCartEvent(User user, Product
product) {
            return new LogEvent("AddToCart", user, product);
        }

        public static LogEvent generatePurchaseEvent(User user, Product
product) {
            return new LogEvent("Purchase", user, product);
        }

        public static LogEvent generatePaymentEvent(User user, Product
product) {
            return new LogEvent("Payment", user, product);
        }
    }
```

StaticCommonRunner розроблений для спрощеного створення різних подій відштовхуючись від індексу ітерації та фіксації події за допомогою бібліотеки *slf4j*.

```
    public class StaticCommonRunner {

        public static Logger log =
LoggerFactory.getLogger(StaticCommonRunner.class);

        public static void createInfoLog(int index) {
            log.info(createLogEvent(index).toString());
        }

        private static LogEvent createLogEvent(int index) {
            if (index % 2 == 0) {
                return
LogCreatorService.generateAddToCartEvent(createUser(index),
createProduct(index));
            }
            if (index % 3 == 0) {
                return
LogCreatorService.generatePurchaseEvent(createUser(index),
createProduct(index));
```

```
            }
            if (index % 5 == 0) {
                return
LogCreatorService.generatePaymentEvent(createUser(index),
createProduct(index));
            }
            return
LogCreatorService.generateProductViewEvent(createUser(index),
createProduct(index));
        }

        private static Product createProduct(int index) {
            return Product.builder()
                    .name("Product " + index)
                    .price(90.15 + index)
                    .build();
        }

        private static User createUser(int index) {
            return User.builder()
                    .username("User [" + index + "]")
                    .build();
        }
    }
```

Для тестування продуктивності у реальному часі та ефективності зберігання було використано бібліотеку JMH за допомогою якої були створені Benchmark тести, за формулою 4.1:

$$T = \frac{1}{n}\sum_{i=1}^{n} t_i \tag{4.1}$$

Але після отриманих помилок пам’яті при виконані реалізації тестів, було вирішено додати невелику затримку після кожного виконання тесту.

Результат відображено формулою 4.2:

$$T = \frac{1}{n}\sum_{i=1}^{n} (t_i + c) \tag{4.2}$$

де T – середній час виконання методу журналювання у мілісекундах,

n – загальна кількість ітерацій, на яку розрахований бенчмарк,

t_i – час виконання однієї ітерації методу журналювання у мілісекундах,

c – константа затримки, яка дорівнює 1 мілісекунді.

Код класу бенчмарку:

```
@BenchmarkMode(Mode.AverageTime)
@OutputTimeUnit(TimeUnit.MILLISECONDS)
@State(Scope.Benchmark)
public class LoggerBenchmark {

    private static final int ITERATIONS = 10000;
    private static final int RANDOM_LIMIT = 10000;

    @Benchmark
    @Measurement(iterations = ITERATIONS, time = 1, timeUnit =
TimeUnit.MILLISECONDS)
    public void measureLoggerPerformance()throws InterruptedException
  {
        StaticCommonRunner.createInfoLog(new
Random().nextInt(RANDOM_LIMIT));
        Thread.sleep(1);
    }

    public static void main(String[] args) throws RunnerException {
        Options options = new OptionsBuilder()
                .include(LoggerBenchmark.class.getSimpleName())
                .forks(1)
                .build();

        new Runner(options).run();
    }

}
```

Можливості горизонтального масштабування були протестовані за допомогою присутніх в бібліотеці JMH інструментів – анотації *@Threads*, яка визначає кількість виконання потоків.

Проведемо розрахунки за формулою 4.3:

$$T = \frac{1}{m \times n} \sum_{j=1}^{m} \sum_{i=1}^{n} (t_{ij} + c) \tag{4.3}$$

де T – середній час виконання методу журналювання у мілісекундах,

m – кількість потоків,

n – кількість ітерацій в одному потоці,

t_{ij} – час виконання i − ї ітерації в j − му потоці методу (без затримки),

c – константа затримки, яка дорівнює 1 мілісекунді.

Враховуючи специфіку апаратного та програмного забезпечення експериментального середовища, а також накладені системні обмеження на створення паралельних потоків виконання, де граничне значення кількості створюваних потоків детерміновано кількістю наявних обчислювальних ядер, максимальна кількість паралельних потоків виконання, що можуть бути ініційовані в рамках даного дослідження, обмежується величиною 10.

4.2.2 Модуль elk

Для забезпечення функціональної інтеграції розподіленої системи моніторингу та аналізу журналів модуля elk в програмний комплекс, ключовими точками налаштування виступали: файл конфігурації збірки build.gradle, в якому визначалися залежності від необхідних бібліотек та компонентів, а також налаштування logback.xml, де задавалися параметри маршрутизації, форматування та передачі журналів в ELK-систему.

До основних змін файлу конфігурації збірки увійшов блок залежностей.

```
dependencies {
    implementation(project(":common"))

    // Logstash
```

```
        implementation 'net.logstash.logback:logstash-logback-encoder:6.6'
    }
```

Для взаємодії java застосунку з системою ELK Stack було створено та налаштовано logback.xml.

```
    <configuration>
        <appender name="LOGSTASH"
class="net.logstash.logback.appender.LogstashTcpSocketAppender">
            <destination>localhost:8044</destination>
            <encoder
class="net.logstash.logback.encoder.LogstashEncoder">
                <fieldNames>
                    <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-
5level %logger{36} - %msg%n</pattern>
                </fieldNames>
            </encoder>
        </appender>

        <root level="INFO">
            <appender-ref ref="LOGSTASH"/>
        </root>
    </configuration>
```

Параметр *destination* вказує URL для відправки журналів в систему журналювання.

4.2.3 Модуль loki

Налаштування інтеграції для модуля loki було зроблено за допомогою внесення аналогічних змін в файлах build.gradle та logback.xml.

До змін build.gradle увійшло додавання залежності *com.github.loki4j* для можливості взаємодії з системою Grafana Loki.

```
    dependencies {
        implementation(project(":common"))
        // Loki
        implementation group: 'com.github.loki4j', name: 'loki-logback-
appender', version: '1.5.1'}
```

Та налаштування файлу logback.xml.

```
<configuration>
    <appender name="LOKI"
class="com.github.loki4j.logback.Loki4jAppender">
        <http>
            <url>http://localhost:3100/loki/api/v1/push</url>
        </http>
        <format>
            <label>
                <pattern>app=loki-app,host=${HOSTNAME}</pattern>
            </label>
            <message>
                <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} [%thread] %-
5level %logger{36} - %msg%n</pattern>
            </message>
        </format>
    </appender>

    <root level="INFO">
        <appender-ref ref="LOKI" />
    </root>
</configuration>
```

Параметр *url* вказує URL для відправки журналів в систему журналювання.

4.2.4 Модуль splunk

Для налаштування модуля splunk знадобилися подібні до попередніх зміни. До змін build.gradle увійшло додавання залежності *com.splunk.logging* для можливості взаємодії з системою Splunk та налагодження репозиторію з якого завантажується ця залежність.

```
repositories {
    maven {
        url 'https://splunk.jfrog.io/splunk/ext-releases-local'
        name 'Splunk Releases'
    }
}

dependencies {
    implementation(project(":common"))
```

```
    // splunk
    implementation 'com.splunk.logging:splunk-library-javalogging:1.8.0'
}
```

До налаштування logback.xml увійшли також такі параметри як *url* та *token* для визначення можливості взаємодії з Splunk інтеграцією, та основні конфігурації індексації та збереження журналів.

```
    <configuration>
        <include
resource="org/springframework/boot/logging/logback/defaults.xml"/>

        <appender                                                    name="SPLUNK"
class="com.splunk.logging.HttpEventCollectorLogbackAppender">
            <url>http://localhost:8088</url>
            <token>${token}</token>
            <source>splunk_app</source>
            <index>splunk_dev</index>
            <sourcetype>slf4j</sourcetype>
            <host>localhost</host>
            <type>raw</type>
            <messageFormat>text</messageFormat>
            <batch_size_count>1</batch_size_count>

<disableCertificateValidation>true</disableCertificateValidation>
            <layout class="ch.qos.logback.classic.PatternLayout">
                <pattern>%d{yyyy-MM-dd  HH:mm:ss.SSS}  [%thread]  %-5level
%logger{36} - %msg%n</pattern>
            </layout>
        </appender>

        <root level="INFO">
            <appender-ref ref="SPLUNK"/>
        </root>
    </configuration>
```

Розроблене Java-застосування стало ключовим інструментом для генерації журналів та проведення порівняльного аналізу ELK Stack, Grafana Loki та Splunk в рамках дослідження.

4.3 Налаштування запуску JMH тестів

Для проведення бенчмаркінгу та дослідження продуктивності було застосовано Java Microbenchmark Harness (JMH). Інтеграція JMH в систему збірки Gradle була реалізована шляхом реєстрації спеціальної задачі (*task*) з визначенням відповідної конфігурації.

```
tasks.register('jmh', JavaExec) {
    group = 'benchmark'
    description = 'Runs the JMH benchmarks'

    mainClass = 'org.openjdk.jmh.Main'
    classpath = sourceSets.main.runtimeClasspath

    args = [
            'ua.nure.research.performance.LoggerBenchmark',
//        'ua.nure.research.scaling.ScalingSimulatorBenchmark',
            '-f', '1',
            '-wi', '5',
            '-jvmArgsPrepend', '-server',
            '-jvmArgs', '-Xms2G -Xmx2G'
    ]
}
```

Зокрема, у конфігураційному файлі було зареєстровано задачу *jmh* з групою *benchmark* та описом *Runs the JMH benchmarks*. В якості головного класу зазначено *org.openjdk.jmh.Main* з classpath, що відповідає runtime-classpath головного модуля проекту. Аргументами запуску визначено класи бенчмарку *LoggerBenchmark* та *ScalingSimulatorBenchmark* (за для запуску бенчмарків окремо, неактивний бенчмарк закоментований), параметри *warmup/measurement* ітерацій, кількість завантажень та потоків, а також *JVM*-аргументи для виділення розміру стека та купи пам'яті.

Дана конфігурація дозволила інтегрувати JMH в систему збірки та надала можливість запускати бенчмарк-тести через Gradle-команди для аналізу продуктивності компонентів системи.

5 ОПИС ПРОВЕДЕНИХ ДОСЛІДЖЕНЬ

У цьому дослідженні було здійснено три види вимірювань:

- вимірювання продуктивності у реальному часі;
- вимірювання ефективності зберігання;
- вимірювання можливості та ефективності горизонтального масштабування.

Для вимірювання продуктивності та продуктивності з масштабуванням у реальному часі було використано бібліотеку Java Microbenchmark Harness. Вимірювання для кожної з систем проводились з чотирма кількостями ітерацій (100, 1000, 10000, 100000).

Результати вимірювання продуктивності наведено у таблицях 5.1 та 5.3.

Результати вимірювання ефективності зберігання наведено у таблицях 5.2 та 5.4.

Таблиця 5.1 – Вимірювання продуктивності у реальному часі (таблиця виконана самостійно)

№	Кількість ітерацій	ELK Stack (мілісекунд)	Grafana Loki (мілісекунд)	Splunk (мілісекунд)
1	100	1.274 ± 0.061	1.290 ± 0.035	1.336 ± 0.072
2	1000	1.260 ± 0.010	1.284 ± 0.021	1.294 ± 0.005
3	10000	1.294 ± 0.009	1.322 ± 0.012	1.333 ± 0.008
4	100000	1.321 ± 0.004	1.354 ± 0.005	1.380 ± 0.011

Таблиця 5.2 – Вимірювання ефективності зберігання (таблиця виконана самостійно)

№	Кількість ітерацій	ELK Stack (kb)	Grafana Loki (kb)	Splunk (kb)
1	100	4577	8	36
2	1000	10393	1328	56

Кінець таблиці 5.2

3	10000	23613	2632	76
4	100000	67523	7860	108

Для вимірювання продуктивності з масштабуванням була використана можливість бібліотеки JMH - @Threads, завдяки якої можна симулювати багато потокову середу. Для дослідження було симульовано 10 потоків

Таблиця 5.3 – Вимірювання продуктивності з масштабуванням у реальному часі (таблиця виконана самостійно)

№	Кількість ітерацій	ELK Stack (мілісекунд)	Grafana Loki (мілісекунд)	Splunk (мілісекунд)
1	100	1.216 ± 0.015	1.251 ± 0.009	1.284 ± 0.009
2	1000	1.234 ± 0.051	1.269 ± 0.025	1.304 ± 0.016
3	10000	1.282 ± 0.013	1.362 ± 0.029	1.405 ± 0.035
4	100000	1.395 ± 0.005	1.491 ± 0.010	1.522 ± 0.014

Таблиця 5.4 – Вимірювання ефективності зберігання з масштабуванням (таблиця виконана самостійно)

№	Кількість ітерацій	ELK Stack (kb)	Grafana Loki (kb)	Splunk (kb)
1	100	34980 (34.16Mb)	6568 (6.4Mb)	52
2	1000	116336 (113.61Mb)	15752 (15.4Mb)	72
3	10000	169697 (165.72Mb)	28852 (28.2Mb)	88

Кінець таблиці 5.4

4	100000	505630 (493.78Mb)	97020 (94.7Mb)	136

Вимірювання продуктивності у реальному часі (Таблиці 1 та 3) продемонстрували, що ELK Stack забезпечує найкращі показники затримки як для одиночних потоків, так і при горизонтальному масштабуванні до 10 потоків. Це свідчить про високу швидкодію та здатність ефективно обробляти великі обсяги даних. Splunk виявився найменш продуктивною системою, демонструючи найвищі значення затримки в усіх випадках.

Результати вимірювання ефективності зберігання (Таблиці 2 та 4) виявили цікаву закономірність. Для невеликих наборів даних (до 100 ітерацій) найбільш оптимальним рішенням з точки зору використання дискового простору є Grafana Loki. Однак, при збільшенні обсягу даних від 1000 ітерацій і більше, кращі показники демонструє Splunk. Натомість ELK Stack виявився найменш ефективною системою в контексті зберігання даних для всіх розмірів наборів.

Беручи до уваги отримані результати, можна зробити такі висновки та рекомендації. Для додатків із високими вимогами до продуктивності та обробки даних у реальному часі, незалежно від обсягу навантаження, оптимальним рішенням є ELK Stack завдяки його перевагам у швидкодії.

У випадках, коли пріоритетом є ефективне використання дискового простору для невеликих наборів журнальних даних, доцільно обрати Grafana Loki як найбільш оптимальну систему, але така мала кількість наборів недоцільна для справжніх застосунків.

Для додатків, що генерують великі обсяги журнальних даних (від 1000 ітерацій і більше), і де ключовою вимогою є оптимізація зберігання, Splunk стає найкращим вибором завдяки своїй високій ефективності використання дискового простору.

Splunk може не бути оптимальним рішенням для додатків із жорсткими вимогами до продуктивності в реальному часі через гірші показники затримки порівняно з іншими розглянутими системами.

ELK Stack демонструє збалансовану продуктивність та здатність масштабуватися, водночас поступаючись Grafana Loki та Splunk у питаннях ефективності зберігання даних.

6 АНАЛІЗ ПРОВЕДЕНОГО ДОСЛІДЖЕННЯ

Три сервіси для генерації та відправки журналів систем збору інформації створених за допомогою Java, порівнювалися з точки зору швидкодії. Для моделювання реальних умов експлуатації досліджуваних систем збору інформації були розгорнуті їхні локальні версії у вигляді Docker-контейнерів, що імітували повнофункціональні застосунки.

Під час аналізу предметної області була розглянута та розкрита тема сфери моделей збору інформації. Основною метою аналітичного огляду було висвітлення проблематики збору інформації подій в e-commerce веб-застосунках. Було розглянуто ключові функцій популярніших інструментів збору інформації та виділені більш загальні та оригінальні функції. Також були розглянуті архітектурні принципи роботи обраних, для дослідження, систем.

В аналітичному розділі дослідження були проаналізовані принципи функціонування обраних систем Elasticsearch (ELK Stack), Grafana Loki та Splunk, а також визначені цілі, що дозволили дослідженню повністю розкритися. Були поставлені задачі дослідження швидкодії та ефективності зберігання даних.

В ході проведення дослідження було створено сервісний додаток за допомогою мови програмування Java та фреймворку Spring-Boot що розгортався за допомогою інструменту побудови – Gradle. Було створено файли docker-compose.yml які дозволяли здійснювати розгортання ELK Stack та Grafana Loki за допомогою Docker. А також було встановлено Splunk Web застосунок. Були налагоджені системи для збереження даних та комунікація між Java застосунком та системами. Також були визначені тестові сценарії збору інформації.

У експериментальній частині дослідження було заплановано дослідити три типи метрик:

- ефективність зберігання;
- можливості горизонтального масштабування;
- продуктивність у реальному часі.

Оцінка продуктивності у реальному часі полягає у вимірюванні швидкості роботи застосунків в порівнянні між собою та в залежності від кількості ітерацій. Ефективність зберігання вимірюється занятим дисковим простором.

Було проведено серію тестів, що дозволили виміряти час обробки запитів на різних об'ємах даних.

Результати показали, що ELK Stack має найкращу продуктивність при зростанні кількості запитів, тоді як Grafana Loki та Splunk демонструють дещо гірші показники. Це свідчить про те, що ELK Stack може бути кращим вибором для систем з великим об'ємом запитів.

Оцінка ефективності зберігання даних включала вимірювання використання дискового простору для кожної системи при різних об'ємах даних.

Splunk продемонстрував найефективніше використання дискового простору, що є важливим фактором для зменшення вартості зберігання даних. Grafana Loki також показав добрі результати, хоча і з трохи більшим використанням простору. ELK Stack потребував найбільше місця для зберігання даних, що може бути важливим фактором при виборі системи для великих обсягів даних.

Здатність до горизонтального масштабування була перевірена шляхом додавання нових вузлів до кластерів кожної системи і оцінки впливу на продуктивність.

ELK Stack продемонстрував найкращі можливості масштабування, забезпечуючи ефективне додавання нових вузлів без значних втрат у продуктивності. Grafana Loki також добре впорався із завданням, але потребував більше часу для перебалансування даних між вузлами. Splunk показав гірші результати у порівнянні з іншими системами, що може бути пов'язано з архітектурними особливостями цієї системи.

На основі отриманих показників проведеного дослідження, варто зазначити, що кожна з розглянутих систем управління та збору інформації має свої переваги

та недоліки. ELK Stack рекомендується для високонавантажених e-commerce систем завдяки його високій продуктивності та добрій масштабованості. У випадку систем з обмеженим бюджетом на зберігання даних, де важлива ефективність використання дискового простору, слід звернути увагу на Splunk. Grafana Loki можна розглядати як альтернативу для систем середнього рівня навантаження або для специфічних задач, де його функції можуть бути більш доречними. Вибір конкретної системи повинен враховувати специфічні вимоги до функціональності, архітектури конкретного застосунку, а також планований обсяг даних та навантаження. Таким чином, кожна з розглянутих систем має свої переваги та недоліки, і вибір конкретної системи повинен бути заснований на конкретних потребах проекту.

ВИСНОВКИ

В умовах постійного збільшення обсягів даних журналів подій, що генеруються сучасними інформаційними системами, ефективні рішення для їх збору, зберігання та аналізу стають надзвичайно важливими. У цьому дослідженні було проведено порівняння трьох популярних сервісів для збору та аналізу журналів: Elasticsearch (ELK Stack), Grafana Loki та Splunk, створених за допомогою Java та Docker. Порівняння здійснювалося з точки зору швидкодії, ефективності зберігання даних та можливостей горизонтального масштабування.

Основною метою аналітичного огляду було висвітлення проблематики збору інформації подій в e-commerce веб-застосунках. Було розглянуто ключові функції популярних інструментів збору інформації та виділені їх загальні та унікальні особливості. Також були проаналізовані архітектурні принципи роботи обраних систем.

Для моделювання реальних умов експлуатації досліджуваних систем збору інформації були розгорнуті їхні локальні версії у вигляді Docker-контейнерів, що імітували повнофункціональні застосунки. Було створено сервісний додаток за допомогою мови програмування Java та фреймворку Spring-Boot, що розгортався за допомогою інструменту побудови – Gradle. Створені файли docker-compose.yml дозволяли здійснювати розгортання ELK Stack та Grafana Loki за допомогою Docker. Також було встановлено Splunk Web застосунок. Налагоджені системи забезпечували збереження даних та комунікацію між Java застосунком та системами.

У експериментальній частині дослідження було заплановано дослідити три типи метрик:

- ефективність зберігання;
- можливості горизонтального масштабування;
- продуктивність у реальному часі.

Оцінка продуктивності у реальному часі полягала у вимірюванні швидкості роботи застосунків в порівнянні між собою та в залежності від кількості ітерацій. Ефективність зберігання вимірювалася занятим дисковим простором.

Результати дослідження показали, що ELK Stack має найкращу продуктивність при зростанні кількості запитів, тоді як Grafana Loki та Splunk демонструють дещо гірші показники. Однак, Splunk продемонстрував найефективніше використання дискового простору, що є важливим фактором для зменшення вартості зберігання даних. Grafana Loki також показав добрі результати в цьому плані, хоча і з трохи більшим використанням простору. ELK Stack потребував найбільше місця для зберігання даних.

Здатність до горизонтального масштабування була перевірена шляхом додавання нових вузлів до кластерів кожної системи і оцінки впливу на продуктивність. ELK Stack продемонстрував найкращі можливості масштабування, забезпечуючи ефективне додавання нових вузлів без значних втрат у продуктивності. Grafana Loki також добре впорався із завданням, але потребував більше часу для перебалансування даних між вузлами. Splunk показав гірші результати у порівнянні з іншими системами, що може бути пов'язано з архітектурними особливостями цієї системи.

На основі отриманих показників проведеного дослідження, варто зазначити, що кожна з розглянутих систем управління та збору інформації має свої переваги та недоліки. ELK Stack рекомендується для високонавантажених e-commerce систем завдяки його високій продуктивності та добрій масштабованості. У випадку систем з обмеженим бюджетом на зберігання даних, де важлива ефективність використання дискового простору, слід звернути увагу на Splunk. Grafana Loki можна розглядати як альтернативу для систем середнього рівня навантаження або для специфічних задач, де його функції можуть бути більш доречними.

Вибір конкретної системи повинен враховувати специфічні вимоги до функціональності, архітектури конкретного застосунку, а також планований обсяг

даних та навантаження. Таким чином, кожна з розглянутих систем має свої переваги та недоліки, і вибір конкретної системи повинен бути заснований на конкретних потребах проекту.

- розроблено технічну задачу щодо магістерського дослідження;
- зібрано та розписано теоретичний матеріал щодо теми та галузі дослідження – систем журналювання інформації;
- проведено та розписано основні відомості щодо систем журналювання інформації;
- проведено та розписано основні відомості щодо їх можливостей;
- проведено та розписано основні відомості щодо їх відмінних функцій;
- обрано засоби щодо проведення дослідження;
- Обрано три системи збору інформації, які будуть використовуватись у ході дослідження, а саме – Elasticsearch, Splunk, Grafana Loki.

Результати дослідження свідчать, що вибір оптимальної системи збору та аналізу журналів залежить від конкретних вимог та особливостей інформаційної системи. Під час прийняття рішення слід враховувати масштабованість, продуктивність, вартість, а також можливості інтеграції з існуючими інфраструктурними компонентами.

Подальші дослідження можуть бути спрямовані на розширення спектру аналізованих інструментів та платформ, вивчення ефективності обраних рішень у різних сценаріях використання та дослідити вплив масштабованості та продуктивності на загальну ефективність систем збору інформації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Retail e-commerce sales worldwide from 2014 to 2027 - https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales (дата звернення 29.05.2024).

2. E-commerce Defined: Types, History, and Examples - https://www.investopedia.com/terms/e/ecommerce.asp (дата звернення 29.05.2024).

3. Analysis of Models Usability Methods Used on Design Stage to Increase Site Optimization CEUR Workshop Proceedings, 2023, 3403, pp. 387–409.

4. 27 Latest Ecommerce Trends To Boost Your Revenue in 2023 - https://www.cloudways.com/blog/latest-ecommerce-trends (дата звернення 29.05.2024).

5. What is Java technology and why do I need it? - https://www.java.com/en/download/help/whatis_java.html (дата звернення 29.05.2024).

6. IntelliJ IDEA overview - https://www.jetbrains.com/help/idea/discover-intellij-idea.html (дата звернення 29.05.2024).

7. Simple Logging Facade for Java (SLF4J) - https://slf4j.org (дата звернення 29.05.2024).

8. Project Lombok - https://projectlombok.org (дата звернення 29.05.2024).

9. Java Management Extensions (JMX) - https://www.oracle.com/java/technologies/javase/javamanagement.html (дата звернення 29.05.2024).

10. The heart of the free and open Elastic Stack - https://www.elastic.co/elasticsearch (дата звернення 29.05.2024).

11. What Is Splunk & What Does It Do? A Splunk Intro - https://www.splunk.com/en_us/blog/learn/what-splunk-does.html (дата звернення 29.05.2024).

12. Grafana Loki - https://grafana.com/docs/loki/latest (дата звернення

29.05.2024).

13. Introduction to Java Microbenchmarking with JMH (Java Microbenchmark Harness) - https://medium.com/@AlexanderObregon/introduction-to-java-microbenchmarking-with-jmh-java-microbenchmark-harness-55af74b2fd38 (дата звернення 29.05.2024).

14. Docker overview - https://docs.docker.com/get-started/overview (дата звернення 29.05.2024).

15. Docker Compose overview - https://docs.docker.com/compose (дата звернення 29.05.2024).

16. What is log management - https://www.dynatrace.com/news/blog/what-is-log-management (дата звернення 29.05.2024).

17. Choosing the Right Data Logger: Essential Tips - https://www.hioki.com/sg-en/learning/test-tools/data-loggers-explained (дата звернення 29.05.2024).

18. 20+ Best Log Management Tools for Monitoring, Analytics & More: Pros & Cons Comparison [2023] - https://sematext.com/blog/best-log-management-tools (дата звернення 29.05.2024).

19. Elasticsearch: What It Is, How It Works, And What It’s Used For - https://www.knowi.com/blog/what-is-elastic-search (дата звернення 29.05.2024).

20. What is Elastic Stack (ELK Stack) – https://www.techtarget.com/searchitoperations/definition/Elastic-Stack (дата звернення 29.05.2024).

21. Splunk is the key to enterprise resilience - https://www.splunk.com/en_us/about-us/why-splunk.html (дата звернення 29.05.2024).

22. Loki overview - https://grafana.com/docs/loki/latest/get-started/overview/ (дата звернення 29.05.2024).

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

3. Gruzdo, I., Kyrychenko, I., Tereshchenko, G., Shanidze, O. Analysis of Models Usability Methods Used on Design Stage to Increase Site Optimization CEUR Workshop Proceedings, 2023, 3403, pp. 387–409