Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розгортання платформи Apache Kafka для аналізу даних

Виконав(ла): студент(ка) 4 курсу, групи СНз-41
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

(підпис) Кучер І.М. (прізвище та ініціали)

Керівник (підпис) Гром'як Р.С. (прізвище та ініціали)

Нормоконтроль (підпис) Шимчук Г.В. (прізвище та ініціали)

Завідувач кафедри (підпис) Боднарчук І.О. (прізвище та ініціали)

Рецензент (підпис) (прізвище та ініціали)

Тернопіль
2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«24» січня 2022 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

студенту Кучер Ірина Михайлівна
(прізвище, ім'я, по батькові)

1. Тема роботи **Розгортання платформи Apache Kafka для аналізу даних**

Керівник роботи **к.т.н., доц. Гром'як Р.С.**
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*23*» *березня* 2022 року № *4/7-172*

2. Термін подання студентом завершеної роботи 12 червня 2022 р.

3. Вихідні дані до роботи Літературні джерела з тематики роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
ВСТУП. РОЗДІЛ 1. ОПИС СЕРВІСУ APACHE KAFKA. 1.1 Основи роботи сервісу Apache Kafka. 1.2 Робочий процес обміну повідомленнями Pub-Sub. 1.3 Робочий процес обміну повідомленнями в черзі. Група споживачів. 1.4 Роль Zookeeper. РОЗДІЛ 2. ОСНОВНІ ПРИЙОМИ РОБОТИ З СЕРВІСОМ APACHE KAFKA. 2.1 Базові операції Apache Kafka. 2.2 Програмне створення продюсера та споживача. 2.3 Розробка застосунку реального часу для моніторингу соціальної мережі Twitter. 2.4 Інтеграція із Spark для потокової обробки даних. РОЗДІЛ 3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ. 3.1 Поняття та об'єкт аналізу технічної безпеки. 3.2 Розрахунок захисного заземлення. ВИСНОВОК. ПЕРЕЛІК ПОСИЛАНЬ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Гурик О.Я., к.т.н., доц.		

7. Дата видачі завдання 24 січня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	24.01.22-27.01.22	*Виконано*
2.	Підбір джерел по темі роботи	28.01.22 – 01.04.22	*Виконано*
3.	Оформлення першого розділу	15.04.2022	*Виконано*
4.	Оформлення другого розділу	30.04.2022	*Виконано*
5.	Виконання завдання до підрозділу «Безпека		
	життєдіяльності, основи охорони праці»	15.05.2022	*Виконано*
6.	Оформлення кваліфікаційної роботи	07.06.2022	*Виконано*
7.	Перевірка на плагіат	07.06.2022	*Виконано*
8.	Нормоконтроль	09.06.2022	*Виконано*
9.	Попередній захист кваліфікаційної роботи	11.06.2022	*Виконано*
10.	Захист кваліфікаційної роботи	13.06.2022	

Студент _____ Кучер І.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Гром'як Р.С.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розгортання платформи Apach Kafka для аналізу даних // Кваліфікаційна робота освітнього рівня "Бакалавр" // Кучер Ірина Михайлівна// Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп’ютерних наук, група СНз-41 // Тернопіль, 2022 // с. – 47, рис. – 5, табл. – 6, кресл. – 11, додат. – 3, бібліогр. – 13.

Ключові слова: big data, Apache Kafka, збір даних, messaging system, stream processing.

Кваліфікаційна робота присвячена розгортанню та практичному використанню сервісу потокової обробки повідомлень Apache Kafka.

В першому розділі кваліфікаційної роботи виконано огляд основних принципів потокового оброблення повідомлень та його прикладному застосуванню для збору даних.

В другому розділі розглянуто можливі використання сервісу Apache Kafka на прикладі реального проєкту.

Об’єкт дослідження: процес потокового оброблення повідомлень.

Предмет дослідження: сервіс Apache Kafka як засіб потокового оброблення даних.

Мета роботи: описати процес розгортання сервісу Apache Kafka для задач збору даних на основі системи потокового оброблення повідомлень та навести приклад застосування сервісу.

ANNOTATION

Deploy the Apach Kafka data analysis platform // Qualification work of the educational level "Bachelor" // Iryna Kucher // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science, Group СНз-41 // Ternopil, 2022 // p. – 47 , fig. – 5, references – 13, posters – 10, annexes – 3.

Keywords: big data, Apache Kafka, data collection, messaging system, stream processing.

Qualification work is devoted to the deployment and practical use of the Apache Kafka message streaming service.

The first section of the qualification work reviews the basic principles of streaming message processing and its application to data collection.

The second section discusses the possible uses of the Apache Kafka service on the example of a real project.

Object of research: the process of streaming messages.

Subject of research: Apache Kafka service as a means of data streaming.

Purpose: to describe the process of deploying the Apache Kafka service for data collection tasks based on the instant messaging system and give an example of using the service.

ЗМІСТ

ВСТУП

У технологіях Big Data використовується величезний обсяг даних. Що стосується даних, у нас є дві основні проблеми. Перше завдання полягає в тому, як зібрати великий обсяг даних, а друге – проаналізувати зібрані дані. Щоб подолати ці проблеми, потрібна система обміну повідомленнями.

Kafka розроблена для розподілених високопродуктивних систем. Kafka, як правило, дуже добре працює як заміна більш традиційного брокера повідомлень. У порівнянні з іншими системами обміну повідомленнями, Kafka має кращу пропускну здатність, вбудоване розділення, реплікацію та відмовостійкість, що робить її придатною для великомасштабних застосунків обробки повідомлень.

Система обміну повідомленнями відповідає за передачу даних з однієї програми в іншу, тому програми можуть зосередитися на даних, але не турбуватися про те, як ними обмінюватись. Розподілений обмін повідомленнями заснований на концепції надійної черги повідомлень. Повідомлення розміщуються в черзі асинхронно між клієнтськими програмами та системою обміну повідомленнями. Доступні два типи шаблонів обміну повідомленнями: один – це система обміну повідомленнями «точка-точка», а інша — система обміну повідомленнями "публікація-підписка" (publisher-subscriber) (pub-sub). Більшість шаблонів обміну повідомленнями побудовані за принципом pub-sub.

У системі "точка-точка" повідомлення зберігаються в черзі. Один або кілька споживачів можуть читати повідомлення в черзі, але конкретне повідомлення може використовуватися максимум одним споживачем. Як тільки споживач читає повідомлення в черзі, воно зникає з цієї черги. Типовим прикладом цієї системи є система обробки замовлень, де кожне замовлення оброблятиметься одним обробником замовлень, але одночасно можуть працювати й кілька обробників замовлень.

У системі опублікування-підписки повідомлення зберігаються в темі. На відміну від системи "точка-точка", споживачі можуть підписатися на одну або

кілька тем і отримувати всі повідомлення в цій темі. У системі Publish-Subscribe продюсери повідомлень називаються видавцями, а споживачі повідомлень — передплатниками. Прикладом із реального життя є система передплачуваного кабельного телебачення, яка публікує різні канали, як-от спорт, фільми, музика тощо, і будь-хто може підписатися на власний набір каналів і отримати їх, коли доступні їхні передплачені канали.

Apache Kafka — це розподілена система обміну повідомленнями для публікації та підписки та надійна черга, яка може обробляти великий обсяг даних і дозволяє передавати повідомлення від однієї кінцевої точки до іншої. Kafka підходить як для офлайн-, так і для онлайн-повідомлень. Повідомлення Kafka зберігаються на диску та реплікуються в кластері, щоб запобігти втраті даних. Kafka побудована на основі служби синхронізації Zookeeper. Вона дуже добре інтегрується з Apache Storm і Spark для аналізу потокових даних у реальному часі.

Нижче наведено кілька переваг Kafka.

Надійність — Kafka є розподіленим, розділеним, тиражованим і стійким до відмов.

Масштабованість – система обміну повідомленнями Kafka легко масштабується без простоїв.

Довговічність — Kafka використовує «розподілений журнал фіксації», що означає, що повідомлення зберігаються на диску якомога швидше, отже, вони довговічний.

Продуктивність – Kafka має високу пропускну здатність як для публікації, так і для підписки повідомлень. Він підтримує стабільну роботу, навіть якщо зберігається багато терабайтів повідомлень.

Kafka дуже швидкий і гарантує нульовий час простою та нульову втрату даних. Kafka можна використовувати в багатьох випадках використання. Деякі з них перераховані нижче.

Метрики – Kafka часто використовується для даних оперативного моніторингу. Це включає агрегування статистичних даних із розподілених програм для створення централізованих каналів операційних даних.

Рішення для агрегації журналів — Kafka можна використовувати в організації для збору журналів з кількох служб і надання їх у стандартному форматі для кількох споживачів.

Stream Processing – популярні фреймворки, такі як Storm і Spark Streaming, зчитують дані з теми, обробляють їх і записують оброблені дані в нову тему, де вони стають доступними для користувачів і програм. Сильна довговічність Kafka також дуже корисна в контексті обробки потоків.

Kafka — це уніфікована платформа для обробки всіх каналів даних у реальному часі. Kafka підтримує доставку повідомлень з низькою затримкою та дає гарантію стійкості до відмов у разі збоїв машини. Він здатний працювати з великою кількістю різноманітних споживачів. Кафка дуже швидкий, виконує 2 мільйони записів/сек. Kafka зберігає всі дані на диску, що, по суті, означає, що всі записи спрямовуються в кеш сторінок ОС (RAM). Це робить дуже ефективним передачу даних з кешу сторінок до мережевого сокета.

РОЗДІЛ 1. ОПИС СЕРВІСУ APACHE KAFKA

1.1 Основи роботи сервісу Apache Kafka

Перш ніж заглибитися в Kafka, слід знати про основні терміни, такі як теми, брокери, продюсери та споживачі. Наступна діаграма (див. рис. 1.1) ілюструє основні терміни, а таблиця детально описує компоненти діаграми.

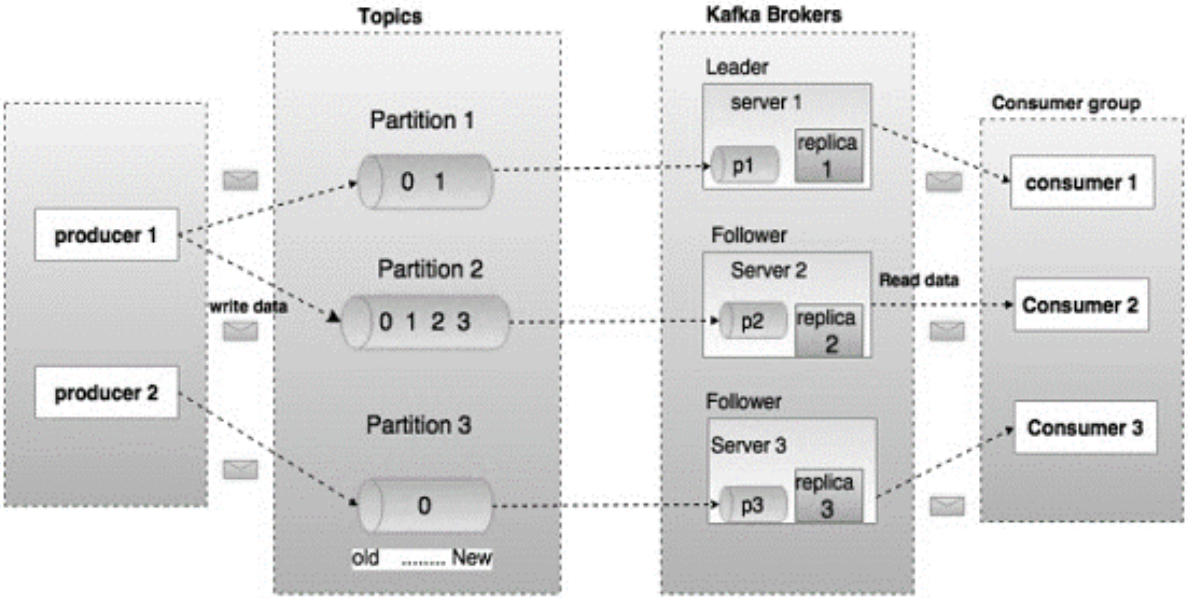

Рисунок 1.1 – Архітектура сервісу Apache Kafka

На діаграмі вище тема налаштована на три розділи (партиції). Розділ 1 має два коефіцієнти зміщення 0 і 1. Розділ 2 має чотири коефіцієнти зміщення 0, 1, 2 і 3. Розділ 3 має один коефіцієнт зміщення 0. Ідентифікатор репліки такий же, як і ідентифікатор сервера, на якому вона розміщена.

Припустимо, якщо коефіцієнт реплікації теми встановлено на 3, тоді Kafka створить 3 ідентичні репліки кожного розділу та помістить їх у кластер, щоб зробити доступними для всіх його операцій. Щоб збалансувати навантаження в кластері, кожен брокер зберігає один або кілька з цих розділів. Декілька продюсерів і споживачів можуть публікувати та отримувати повідомлення одночасно.

Таблиця 1.1 – Компоненти сервісу Apache Kafka

№ з.п.	Компоненти та їх описи
1.	Теми (Topics) Потік повідомлень, що належать до певної категорії, називається темою. Дані зберігаються в темах. Теми розбиті на розділи. Для кожної теми Кафка зберігає мінімальний розмір одного розділу. Кожен такий розділ містить повідомлення в незмінній впорядкованій послідовності. Розділ реалізується як набір файлів сегментів однакового розміру.
2.	Розділи (Partitions) Теми можуть мати багато розділів, тому він може обробляти довільну кількість даних.
3.	Зміщення розділу (Partition offset) Кожне повідомлення в розділі має унікальний ідентифікатор послідовності, який називається "зміщенням".
4.	Репліки розділу (Replicas of partition) Репліки — це не що інше, як «резервні копії» розділу. Репліки ніколи не зчитують і не записують дані. Вони використовуються для запобігання втрати даних.
5.	Брокери (Brokers) – Брокери — це проста система, що відповідає за збереження опублікованих даних. Кожен брокер може мати нуль або більше розділів на тему. Припустимо, якщо в темі N розділів і N кількість брокерів, у кожного брокера буде один розділ. – Припустимо, якщо в темі є N розділів і більше N брокерів (n + m), перший N брокер матиме один розділ, а наступний M брокер не матиме жодного розділу для цієї конкретної теми. – Припустимо, що якщо в темі є N розділів і менше ніж N брокерів (n-m), кожен брокер матиме один або кілька розділів, які спільно використовуються. Цей сценарій не рекомендується через нерівномірний розподіл навантаження між брокерами.
6.	Кластер (Cluster) У Kafka більше одного брокера називають кластером. Кластер Kafka можна розширити без додаткових простоїв. Ці кластери використовуються для керування збереженням та реплікацією даних повідомлень.

Продовження таблиці 1.1

№ з.п.	Компоненти та їх описи
7.	Продюсер (Producer) Продюсери є видавцями повідомлень на одну або кілька тем. Виробники надсилають дані брокерам Kafka. Кожного разу, коли продюсер публікує повідомлення брокеру, брокер просто додає повідомлення до файлу останнього сегмента. Фактично, повідомлення буде додано до розділу. Продюсер також може надсилати повідомлення до розділу на свій вибір.
8.	Споживач (Consumer) Споживачі зчитують дані брокерів. Споживачі підписуються на одну або кілька тем і споживають опубліковані повідомлення, витягуючи дані з брокерів.
9.	Лідер (Leader) "Лідер" – це вузол, відповідальний за всі операції читання і запису для даного розділу. Кожен розділ має один сервер, який виконує роль лідера.
10.	Послідовник (Follower) Вузол, який слідує вказівкам лідера, називається послідовником. Якщо лідер стає недоступний, один із послідовників автоматично стане новим лідером. Послідовник діє, як звичайний споживач, витягує повідомлення та оновлює власне сховище даних.

Розглянемо рисунок 2.2. Він містить зображення архітектури кластеру Kafka.

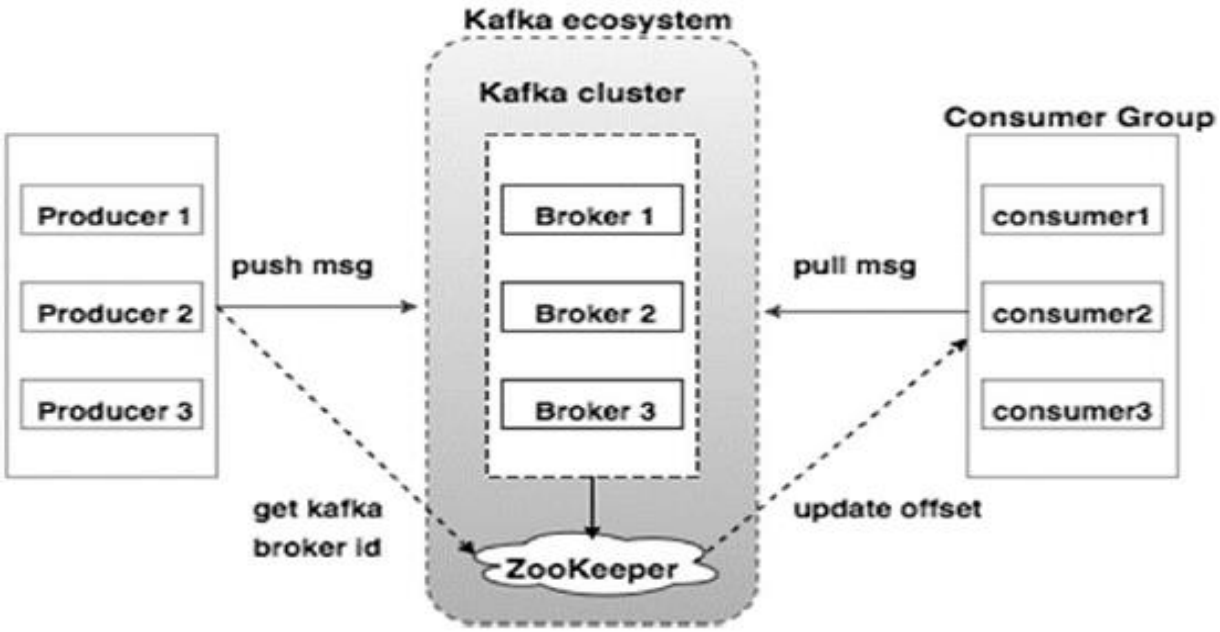

Рисунок 1.2 – Архітектура кластеру Kafka

Таблиця 1.2 містить опис кожного компоненту з рисунку 1.2.

Таблиця 1.2 – Опис компонентів архітектури кластера Kafka

№ з.п.	Назва компоненту та його опис
1.	Брокер Кластер Kafka зазвичай складається з кількох брокерів для підтримки балансу навантаження. Брокери Kafka не мають стану, тому вони використовують Zookeeper для підтримки стану свого кластера. Один екземпляр брокера Kafka може обробляти сотні тисяч читань і записів на секунду, і кожен брокер може обробляти терабайти повідомлень без впливу на продуктивність. Вибір лідера брокера Kafka може бути здійснено в Zookeeper.
2.	Zookeeper Zookeeper використовується для управління та координації брокера Kafka. Служба Zookeeper в основному використовується для сповіщення продюсера та споживача про наявність будь-якого нового брокера в системі Kafka або збій брокера в системі Kafka. Відповідно до сповіщення, отриманого Zookeeper щодо наявності чи збою брокера, продюсер і споживач приймають рішення та починають узгоджувати своє завдання з іншим брокером.
3.	Продюсери Виробники передають дані брокерам. Коли новий брокер запускається, усі виробники шукають його та автоматично надсилають повідомлення цьому новому брокеру. Продюсер Kafka не чекає підтверджень від брокера і надсилає повідомлення так швидко, як брокер може впоратися.
4.	Споживачі Оскільки брокери Kafka не мають громадянства, а це означає, що споживач повинен підтримувати кількість спожитих повідомлень за допомогою зміщення розділу. Якщо споживач визнає конкретне зміщення повідомлення, це означає, що споживач спожив усі попередні повідомлення. Споживач надсилає брокеру асинхронний запит на витяг, щоб мати буфер байтів, готовий до використання. Споживачі можуть перемотати назад або перейти до будь-якої точки розділу, просто вказавши значення зміщення. Значення споживчого зміщення повідомляє Zookeeper.

На даний момент ми обговорювали основні концепції Kafka. Розглянемо тепер робочий процес Kafka.

Kafka — це просто набір тем, розділених на один або кілька розділів. Розділ Kafka — це лінійно впорядкована послідовність повідомлень, де кожне повідомлення ідентифікується за своїм індексом (званим зміщенням). Усі дані в кластері Кафки є роз'єднаним об'єднанням розділів. Вхідні повідомлення записуються в кінці розділу, а повідомлення послідовно читаються споживачами. Довговічність забезпечується копіюванням повідомлень різним брокерам.

Kafka забезпечує швидку, надійну, стійку, відмовостійкість з нульовим простоєм систему обміну повідомленнями на основі черги. В обох випадках продюсери просто надсилають повідомлення на тему, а споживач може вибрати будь-який тип системи обміну повідомленнями залежно від своїх потреб. Проаналізуємо, як споживач може вибрати систему обміну повідомленнями на свій вибір.

1.2 Робочий процес обміну повідомленнями Pub-Sub

Нижче наведено покроковий робочий процес Pub-Sub Messaging.

Продюсери надсилають повідомлення на тему через регулярні проміжки часу. Kafka broker зберігає всі повідомлення в розділах, налаштованих для цієї конкретної теми. Це гарантує, що повідомлення рівномірно розподіляються між розділами. Якщо виробник надсилає два повідомлення і є два розділи, Kafka збереже одне повідомлення в першому розділі, а друге повідомлення в другому розділі.

Споживач підписується на певну тему.

Після того, як споживач підписується на тему, Kafka надасть споживачеві поточне зміщення теми, а також збереже зміщення в ансамблі Zookeeper.

Споживач запитуватиме Kafka через регулярні інтервали (наприклад, 100 мс) для нових повідомлень. Як тільки Кафка отримує повідомлення від виробників, він пересилає ці повідомлення споживачам.

Споживач отримає повідомлення та опрацює його. Після обробки повідомлень споживач надішле брокеру Kafka підтвердження.

Як тільки Kafka отримує підтвердження, він змінює зміщення на нове значення та оновлює його в Zookeeper. Оскільки зміщення зберігаються в Zookeeper, споживач може правильно прочитати наступне повідомлення навіть під час збоїв на сервері.

Цей вищезазначений потік буде повторюватися, доки споживач не зупинить запит. Споживач має можливість у будь-який час перемотати назад/перейти до потрібного зміщення теми та прочитати всі наступні повідомлення.

1.3 Робочий процес обміну повідомленнями в черзі. Група споживачів

У системі обміну повідомленнями в черзі замість одного споживача може бути група споживачів, що мають однаковий «Ідентифікатор групи» і підписана на тему. Простіше кажучи, споживачі, які підписалися на тему з однаковим «Ідентифікатором групи», розглядаються як одна група, і повідомлення розподіляються між ними. Перевіримо реальний робочий процес цієї системи.

Продюсери регулярно надсилають повідомлення в тему. Kafka зберігає всі повідомлення в розділах, налаштованих для цієї конкретної теми, подібно до попереднього сценарію.

Один споживач підписується на певну тему, припустимо, що це "Тема-01" з "Ідентифікатором групи" як "Група-1".

Kafka взаємодіє зі споживачем так само, як Pub-Sub Messaging, доки новий споживач не підписується на ту саму тему, «Тема-01» з тим же «Ідентифікатором групи», що й у «Групи-1».

Після прибуття нового споживача Kafka перемикає свою роботу в режим спільного доступу та ділиться даними між двома споживачами. Цей спільний

доступ триватиме, доки кількість споживачів не досягне кількості розділів, налаштованих для цієї конкретної теми.

Як тільки кількість споживачів перевищує кількість розділів, новий споживач не отримає жодних повідомлень, доки будь-який із існуючих споживачів не скасує підписку. Цей сценарій виникає через те, що кожному споживачеві в Kafka буде присвоєно як мінімум один розділ, і як тільки всі розділи будуть призначені існуючим споживачам, новим споживачам доведеться чекати.

Ця функція також називається «Група споживачів». Таким же чином, Kafka надасть найкраще з обох систем у дуже простий та ефективний спосіб.

1.4 Роль Zookeeper

Критичною залежністю Apache Kafka є Apache Zookeeper, який є розподіленою службою конфігурації та синхронізації. Zookeeper служить координаційним інтерфейсом між брокерами Kafka і споживачами. Сервери Kafka обмінюються інформацією через кластер Zookeeper. Kafka зберігає основні метадані в Zookeeper, такі як інформація про теми, брокерів, зміщення для споживачів (зчитувачів черги) тощо.

Оскільки вся критична інформація зберігається в Zookeeper, і він зазвичай реплікує ці дані по всьому своєму ансамблю, збій брокера / Zookeeper Kafka не впливає на стан кластера Kafka. Кафка відновить стан після перезавантаження Zookeeper. Це дає нульовий час простою для Kafka. Вибір лідера між брокером Kafka також здійснюється за допомогою Zookeeper у разі відмови лідера.

РОЗДІЛ 2. ОСНОВНІ ПРИЙОМИ РОБОТИ З СЕРВІСОМ APACHE KAFKA

2.1 Базові операції Apache Kafka

Спочатку почнемо реалізовувати конфігурацію «один вузол-один брокер», а потім перенесемо наші налаштування до конфігурації з одним вузлом і кількома брокерами.

Установка необхідного програмного забезпечення буде описана далі в цій роботі. Перш ніж перейти до налаштування кластера Kafka, спочатку потрібно запустити Zookeeper, оскільки Kafka Cluster використовує Zookeeper.

Старт Zookeeper.

Відкрийте новий термінал і введіть таку команду

```
bin/Zookeeper-server-start.sh config/Zookeeper.properties
```

Щоб запустити Kafka Broker, слід ввести таку команду

```
bin/kafka-server-start.sh config/server.properties
```

Після запуску Kafka Broker треба виконати команду "jps" на терміналі Zookeeper, і можна буде побачите таку відповідь:

```
821 QuorumPeerMain
928 Kafka
931 Jps
```

Тепер видно побачити два демона, які працюють на терміналі, де QuorumPeerMain — це демон Zookeeper, а інший — демон Kafka.

Конфігурація одного вузла – одного брокера.

У цій конфігурації у є один екземпляр ідентифікатора Zookeeper та брокера. Нижче наведено кроки для його налаштування.

Створення теми Kafka – Kafka надає утиліту командного рядка під назвою "kafka-topics.sh" для створення тем на сервері. Відкрийте новий термінал і введіть приклад нижче:

```
bin/kafka-topics.sh --create --Zookeeper localhost:2181 --
replication-factor 1

--partitions 1 --topic topic-name
```

Приклад:

```
bin/kafka-topics.sh --create --Zookeeper localhost:2181 --
replication-factor 1

--partitions 1 --topic Hello-Kafka
```

Ми щойно створили тему під назвою "Hello-Kafka" з одним розділом і одним фактором репліки. Створений вище вихід буде подібним до наступного –

В результаті — Створена тема «Привіт-Кафка»

Після створення теми ви можете отримати сповіщення у вікні терміналу Kafka broker і журнал для створеної теми, зазначений у “/tmp/kafka-logs/“ у файлі config/server.properties.

Щоб отримати список тем на сервері Kafka, ви можете скористатися такою командою:

```
bin/kafka-topics.sh --list --Zookeeper localhost:2181
```

Результат виводу:

```
Hello-Kafka
```

Оскільки ми створили тему, в ній буде перераховано лише «Hello-Kafka». Припустимо, якщо ви створюєте більше однієї теми, ви отримаєте назви тем у виводі. Старт продюсера для надсилання повідомлень:

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --
topic-topic-name
```

З наведеного вище синтаксису потрібні два основні параметри для клієнта командного рядка продюсера:

1. Список брокерів — список брокерів, яким ми хочемо надіслати повідомлення. У цьому випадку у нас лише один брокер. Файл Config/server.properties містить ідентифікатор порту брокера, оскільки ми знаємо, що наш брокер прослуховує порт 9092, тому ви можете вказати його безпосередньо.
2. Назва теми — в нашому випадку Hello Kafka.

Приклад:

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --
topic Hello-Kafka
```

Продюсер чекатиме введення від stdin і публікує його в кластері Kafka. За замовчуванням кожен новий рядок публікується як нове повідомлення, тоді властивості продюсера за замовчуванням вказуються у файлі "config/producer.properties". Тепер можна ввести кілька рядків повідомлень у терміналі, як показано нижче (лістинг 2.1).

Лістинг 2.1 – Приклад старту продюсера для надсилання повідомлень

```
$ bin/kafka-console-producer.sh --broker-list localhost:9092
--topic Hello-Kafka[2016-01-16 13:50:45,931]
WARN property topic is not valid (kafka.utils.Verifia-
bleProperties)
Hello
My first message

My second message
```

Отримання повідомлень. Як і у продюсера, властивості споживача за замовчуванням вказуються у файлі "config/consumer.proper-ties". Відкрийте новий термінал і введіть наведений нижче синтаксис для читання повідомлень:

```
bin/kafka-console-consumer.sh --Zookeeper localhost:2181 –
topic topic-name
--from-beginning
```

Приклад:

```
bin/kafka-console-consumer.sh --Zookeeper localhost:2181 —
topic Hello-Kafka
--from-beginning
```

Вивід системи:

```
Hello
My first message
My second message
```

Нарешті, можна вводити повідомлення з терміналу продюсера і бачити, як вони з’являються в терміналі споживача. На даний момент у вас є дуже добре уявлення про кластер одного вузла з одним брокером. Тепер перейдемо до конфігурації кількох брокерів.

Конфігурація одного вузла з кількома брокерами.

Перш ніж переходити до налаштування кластера кількох брокерів, спочатку запустіть сервер Zookeeper.

Створіть кілька брокерів Kafka – у нас є один екземпляр брокера Kafka вже в config/server.properties. Тепер нам потрібно кілька екземплярів брокерів, тому скопіюйте наявний файл server.properties у два нові файли конфігурації та перейменуйте їх на server-one.properties та server-two.prop-erties. Потім відредагуйте обидва нові файли та виконайте наступні зміни (лістинг 2.2 та лістинг 2.3):

Лістинг 2.2 Зміни файлу config/server-one.properties

```
# The id of the broker. This must be set to a unique integer
for each broker.
broker.id=1
# The port the socket server listens on
port=9093
# A comma separated list of directories under which to store
log files
log.dirs=/tmp/kafka-logs-1
```

Лістинг 2.3 – Зміни файлу config/server-two.properties

```
# The id of the broker. This must be set to a unique integer
for each broker.
broker.id=2
# The port the socket server listens on
port=9094
# A comma separated list of directories under which to store
log files
log.dirs=/tmp/kafka-logs-2
```

Після того, як всі зміни були зроблені на трьох серверах, відкрийте три нових термінали, щоб запустити кожного брокера по черзі.

```
Broker1
bin/kafka-server-start.sh config/server.properties
Broker2
bin/kafka-server-start.sh config/server-one.properties
Broker3
bin/kafka-server-start.sh config/server-two.properties
```

Тепер у нас на машині працюють три різних брокери. Можна самостійно перевірити всіх демонів, ввівши jps на терміналі Zookeeper, тоді ви побачите вивід системи.

Призначимо значення коефіцієнта реплікації як 3 для цієї теми, оскільки у нас працюють три різні брокери. Якщо у вас два брокери, то призначене значення репліки буде 2:

```
bin/kafka-topics.sh --create --Zookeeper localhost:2181 --
replication-factor 3
-partitions 1 --topic topic-name
```

Приклад:

```
bin/kafka-topics.sh --create --Zookeeper localhost:2181 --
replication-factor 3
-partitions 1 --topic Multibrokerapplication
```

Система виведе повідомлення:

```
created topic "Multibrokerapplication"
```

Команда «Describe» використовується, щоб перевірити, який брокер слухає поточну створену тему, як показано нижче –

```
bin/kafka-topics.sh --describe --Zookeeper localhost:2181
--topic Multibrokerappli-cation
```

Вивід системи:

```
bin/kafka-topics.sh --describe --Zookeeper localhost:2181
--topic Multibrokerappli-cation

Topic:Multibrokerapplication    PartitionCount:1
ReplicationFactor:3 Configs:

Topic:Multibrokerapplication Partition:0 Leader:0
Replicas:0,2,1 Isr:0,2,1
```

З наведеного вище результату ми можемо зробити висновок, що перший рядок дає підсумок усіх розділів, показуючи назву теми, кількість розділів і коефіцієнт реплікації, який ми вже вибрали. У другому рядку кожен вузол буде лідером для випадково обраної частини розділів.

У нашому випадку ми бачимо, що наш перший брокер (з broker.id 0) є лідером. Тоді Replicas:0,2,1 означає, що всі брокери реплікують тему, нарешті, "Isr" - це набір "синхронізованих" реплік. Тобто, це підмножина реплік, які наразі живі і керуються лідером.

Запуск Producer для надсилання повідомлень.

Ця процедура залишається такою ж, як і в налаштуваннях одного брокера.

Приклад:

```
bin/kafka-console-producer.sh --broker-list localhost:9092
--topic Multibrokerapplication
```

Вивід системи:

```
bin/kafka-console-producer.sh --broker-list localhost:9092 --
topic Multibrokerapplication
[2016-01-20 19:27:21,045] WARN Property topic is not valid
(kafka.utils.Verifia-bleProperties)
This is single node-multi broker demo
This is the second message
```

Процедура отримання повідомлень споживачем залишається такою ж, як і в налаштуваннях одного брокера.

Приклад:

```
bin/kafka-console-consumer.sh --zookeeper localhost:2181
—topic Multibrokerapplication --from-beginning
```

Вивід системи:

```
bin/kafka-console-consumer.sh --zookeeper localhost:2181
—topic Multibrokerapplication —from-beginning
This is single node-multi broker demo
This is the second message
```

Основні операції з темами.

Вище описано, як створити тему в кластері Kafka. Тепер подивимось, як змінити створену тему за допомогою наступної команди:

```
bin/kafka-topics.sh —zookeeper localhost:2181 --alter --topic
topic_name
--partitions count
```

Ми вже створили тему «Hello-Kafka» з кількістю розділів і одним фактором репліки. Тепер за допомогою команди «alter» ми змінили кількість розділів.

```
bin/kafka-topics.sh --zookeeper localhost:2181
--alter --topic Hello-kafka --partitions 2
```

Вивід системи:

```
WARNING: If partitions are increased for a topic that has a
key, the partition logic or ordering of the messages will be
affected
Adding partitions succeeded!
```

Щоб видалити тему, можна використовувати наступний синтаксис.

```
bin/kafka-topics.sh --zookeeper localhost:2181 --delete --
topic topic_name
```

Приклад:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --delete --
topic Hello-kafka
```

Вивід системи:

```
> Topic Hello-kafka marked for deletion
```

Примітка. Операція видалення не матиме ефекту, якщо для delete.topic.enable не встановлено значення true.

2.2 Програмне створення продюсера та споживача

2.2.1 Створення продюсера

Створимо додаток для публікації та використання повідомлень за допомогою клієнта Java. Клієнт продюсера Kafka складається з наступних API.

API KafkaProducer.

Розберемося з найважливішим набором API продюсера Kafka у цьому розділі. Центральною частиною API KafkaProducer є клас «KafkaProducer». Клас

KafkaProducer надає можливість підключення брокера Kafka у його конструкторі за допомогою наступних методів.

Клас KafkaProducer надає метод send для асинхронного надсилання повідомлень до теми. Підпис send() виглядає наступним чином:

```
producer.send(new ProducerRecord <byte[],byte[]>(topic,
partition, key1, value1), callback);
```

Параметри методу send: ProducerRecord — виробник керує буфером записів, які очікують на відправку; callback — наданий користувачеві зворотний виклик, який виконується, коли запис підтверджено сервером (нуль означає відсутність зворотного виклику).

Клас KafkaProducer надає метод flush, щоб переконатися, що всі раніше надіслані повідомлення були фактично завершені. Синтаксис методу flush виглядає наступним чином:

```
public void flush()
```

Клас KafkaProducer надає метод partitionFor, який допомагає отримати метадані розділу для даної теми. Його можна використовувати для користувацьких розділів. Спосіб виклику цього методу:

```
public Map metrics()
```

Він повертає карту внутрішніх параметрів, яку підтримує виробник.

public void close() — клас KafkaProducer надає блоки методів close, доки не будуть завершені всі надіслані запити.

API продюсера. Центральною частиною API Producer є клас "Producer". Клас Producer надає можливість підключити брокер Kafka у його конструкторі наступними методами.

Клас продюсера надає метод send для надсилання повідомлень на одну або кілька тем за допомогою синтаксису з лістингу 2.4.

Лістинг 2.4 – Відсилання повідомлень продюсером

```
public void send(KeyedMessaget<k,v> message)
- sends the data to a single topic, partitioned by key using
either sync or async producer.
public void send(List<KeyedMessage<k,v>>messages)
- sends data to multiple topics.
Properties prop = new Properties();
prop.put(producer.type,”async”)
ProducerConfig config = new ProducerConfig(prop);
```

Є два типи продюсерів – Sync та Async.

Така ж конфігурація API також застосовується до продюсера "Sync". Різниця між ними полягає в тому, що продюсер синхронізації надсилає повідомлення безпосередньо, але робить це у фоновому режимі. Асинхронний виробник є кращим, якщо потрібна вища пропускна здатність. У попередніх версіях, наприклад 0.8, асинхронний продюсер не має зворотного виклику для send() для реєстрації обробників помилок. Це доступно лише в версії 0.9 і новіших.

```
public void close()
```

Клас Producer надає метод close, щоб закрити з'єднання пулу продюсерів з усіма брокерами Kafka.

Параметри конфігурації.

Основні параметри конфігурації Producer API наведено в таблиці 2.1

Таблиця 2.1 – Параметри конфігурації Producer API

№ з.п.	Параметр та його опис
1.	client.id містить id продюсера
2.	producer.type sync або async

Продовження таблиці 2.1

№ з.п.	Параметр та його опис
3.	acks Параметр конфігурації acks керує критерієм завершення виконання запиту.
4.	retries Якщо запит виробника не вдається виконати, буде повторне виконання.
5.	bootstrap.servers Завантажується список брокерів
6.	linger.ms Якщо потрібно зменшити кількість запитів, можна збільшити значення linger.ms
7.	key.serializer Ключ для інтерфейсу серіалізатора.
8.	value.serializer Значення для інтерфейсу серіалізатора.
9.	batch.size Розмір буфера.
10.	buffer.memory Контролює загальний обсяг пам'яті, доступний продюсерові для буферизації.

ProducerRecord API.

ProducerRecord — це пара ключ/значення, яка надсилається в кластер Kafka. ProducerRecord конструктор класу для створення запису з парами розділів, ключів і значень за допомогою наступного синтаксису:

```
public ProducerRecord (string topic, int partition, k key, v
value)
```

topic – визначена користувачем назва теми, яка буде додана до запису.

partition – кількість розділів.

key – ключ, який буде включено в запис.

value — вміст запису.

```
public ProducerRecord (string topic, k key, v value)
```

Конструктор класу ProducerRecord використовується для створення запису з парами ключів, значень і без розділу. Зміст параметрів аналогічна до попереднього перевизначеного конструктора.

```
public ProducerRecord (string topic, v value)
```

Клас ProducerRecord створює запис без розділу та ключа. Зміст параметрів такий же, що і в двох попередніх варіантах.

Методи класу ProducerRecord наведено в таблиці 2.2.

Таблиця 2.2 – Методи класу ProducerRecord

№ з.п.	Параметр та його опис
1.	public string topic() Тема буде додана до запису.
2.	public K key() Ключ, що буде включений до запису. Якщо такого ключа немає, повернеться значення null.
3.	public V value() Вміст запису.
4.	partition() Номер партиції (розділу) для запису

2.2.2 Створення програми SimpleProducer (простий продюсер)

Перед створенням програми спочатку потрібно запустити ZooKeeper та Kafka broker, а потім створити власну тему в Kafka broker за допомогою команди create topic. Після цього створити клас java з назвою "SimpleProducer.java" з програмним кодом, представленим у додатку А. Цей же програмний код містить програму, що працює з об'єктами створеного класу.

Для компіляції написаної програми можна ввести команду:

```
javac -cp "/path/to/kafka/kafka_2.11-0.9.0.0/lib/*" *.java
```

Для виконання скомпільованого додатку можна виконати команду:

```
java -cp "/path/to/kafka/kafka_2.11-0.9.0.0/lib/*":.
SimpleProducer <topicname>
```

Виконавши програму, побачимо результат, показаний на лістингу 2.5.

Лістинг 2.5 – Результати виконання створеного продюсера

```
Message sent successfully
To check the above output open new terminal and type Consumer
CLI command to receive messages.
>> bin/kafka-console-consumer.sh --zookeeper localhost:2181 –
topic <topic-name> –from-beginning
1
2
3
4
5
6
7
8
9
10
```

Виведено текст повідомлень, котрі є числами від 1 до 10.

2.2.3 Простий приклад споживача

На даний момент ми створили продюсера для надсилання повідомлень у кластер Kafka. Тепер створимо споживача, який буде читати повідомлення з кластера Kafka. API KafkaConsumer використовується для отримання повідомлень із кластера Kafka. Конструктор класу KafkaConsumer визначено нижче.

```
public KafkaConsumer(java.util.Map<java.lang.String,
java.lang.Object> configs)
```

Тут параметр configs містить мапу налаштувань споживача.

Клас KafkaConsumer містить наступні методи, описані у таблиці 2.4.

Таблиця 2.3 – Методи класу KafkaConsumer

№ з.п.	Параметр та його опис
1.	public java.util.Set<TopicPar-tition> assignment() Отримує множину партицій (розділів) , до котрих асоційований даний споживач
2.	public string subscription() Підписує до даного списку тем для динамічного присвоєння партицій.
3.	public void subscribe(java.util.List<java.lang.String> topics, ConsumerRe-balanceListener listener) Підписує до даного списку тем для динамічного присвоєння партицій.
4.	public void unsubscribe() Відписатися від поточного списку партицій.
5.	public void sub-scribe(java.util.List<java.lang.String> topics) Підписатись на заданий список тем для динамічного отримання асоційований партицій. Якщо список тем порожній, то результат такий же, як для виклику методу unsubscribe().
6.	public void sub-scribe(java.util.regex.Pattern pattern, ConsumerRebalanceLis-tener listener) Екземпляр аргументу відноситься до екземпляру підписки у форматі регулярного виразу, а аргумент слухача отримує сповіщення від екземпляра підписки.
7.	public void assign(java.util.List<TopicPartition> partitions) Асоціювати список партицій о споживача вручну.
8.	poll() Отримати дані для тем або розділів, зазначених за допомогою одного з API підписки/призначення. Метод поверне помилку, якщо теми не підписані до виклику методу.
9.	public void commitSync() Фіксує зміщення, повернуті під час останнього виклику poll() для всіх підписаних списків тем і розділів. Така ж операція застосовується до commitAsync().
10.	public void seek(TopicPartition partition, long offset) Отримати поточне значення зміщення, яке споживач використовуватиме для наступного виклику методу poll().
11.	public void resume() Запуск зупинених на паузу партицій
12.	public void wakeup() Запустити споживача.

API ConsumerRecord використовується для отримання записів із кластера Kafka. Цей API складається з назви теми, номера розділу, з якого отримується запис, і зміщення, яке вказує на запис у розділі Kafka. Клас ConsumerRecord використовується для створення запису споживача з певною назвою теми, кількістю розділів і парами <ключ, значення>. Має наступний вигляд.

```
public ConsumerRecord(string topic, int partition, long
offset, K key, V value)
```

Зміст параметрів пояснень не потребує, оскільки збігається з однойменними параметрами для продюсера.

ConsumerRecords API діє як контейнер для ConsumerRecord. Цей API використовується для збереження списку ConsumerRecord для кожного розділу для певної теми. Його конструктор визначено нижче:

```
public ConsumerRecords(java.util.Map
<TopicPartition,java.util.List
<Consumer-Record>K,V>>> records)
```

Зміст параметрів збігається з аналогічними для провайдера.

Методи класу ConsumerRecords описані в таблиці 2.4.

Таблиця 2.4 – Методи класу ConsumerRecords

№ з.п.	Параметр та його опис
1.	public int count() Число записів для всіх тем.
2.	group.id Приписати окремого споживача до групи з вказаним id.
3.	enable.auto.commit Автоматична фіксація для зміщень, якщо значення true. В іншому випадку фіксації немає.
4.	auto.commit.interval.ms Повертає час, як часто оновлені зміщення записуються в ZooKeeper.
5.	session.timeout.ms Вказує, скільки мілісекунд Kafka буде чекати, поки ZooKeeper відповість на запит (прочитати або записати), перш ніж відмовитися і продовжити отримувати повідомлення.

2.2.4 Створення додатку SimpleConsumer (простий споживач)

Етапи , описані перед створенням продюсера, тут залишаються тими ж. Спочатку треба запустити брокер ZooKeeper і Kafka. Потім створити програму "SimpleConsumer" з класом java з назвою "SimpleConsumer.java" і наповнити їх програмним кодом, наведеним у додатку Б.

Для компіляції та виконання додатку можна скористатись командами, аналогічними для продюсера.

2.3 Розробка застосунку реального часу для моніторингу соціальної мережі Twitter

Давайте проаналізуємо програму в режимі реального часу, щоб отримати найновіші канали Twitter і його хештеги. Раніше ми бачили інтеграцію Storm and Spark з Kafka. В обох сценаріях ми створили Kafka Producer (за допомогою cli) для надсилання повідомлення в екосистему Kafka. Потім інтеграція storm і spark зчитує повідомлення за допомогою споживача Kafka і впроваджує їх в екосистеми storm і spark відповідно. Отже, практично нам потрібно створити продюсера Kafka, який повинен:

- читати канали Twitter за допомогою «Twitter Streaming API»;
- опрацьовувати потоки;
- отримувати хештеги;
- надсилати ці дані на Kafka.

Після того, як Kafka отримує "HashTags", інтеграція Storm/Spark отримує інформацію та надсилає її в екосистему Storm/Spark.

Доступ до «Twitter Streaming API» можна отримати будь-якою мовою програмування. «twitter4j» — це неофіційна бібліотека Java з відкритим вихідним кодом, яка надає модуль на основі Java для легкого доступу до «Twitter Streaming API». «twitter4j» надає структуру на основі прослуховування для

доступу до твітів. Щоб отримати доступ до «Twitter Streaming API», нам потрібно ввійти в обліковий запис розробника Twitter і отримати наведену нижче інформацію про автентифікацію OAuth.

- Customerkey
- CustomerSecret
- AccessToken
- AccessTookenSecret

Після створення облікового запису розробника завантажуємо файли jar «twitter4j» і помістимо їх у шлях класу java.

Повний програмний код продюсера Twitter Kafka (KafkaTwitterProducer.java) наведено у додатку В.

2.4 Інтеграція із Spark для потокової обробки даних

API Spark Streaming забезпечує масштабовану, високопродуктивну, відмовостійку обробку потоків даних в реальному часі. Дані можуть бути отримані з багатьох джерел, таких як Kafka, Flume, Twitter тощо, і можуть бути оброблені за допомогою складних алгоритмів, таких як високорівневі функції, такі як map, reduce, join і window. Нарешті, оброблені дані можуть бути передані до файлових систем, баз даних тощо. Стійкі розподілені набори даних (RDD) є основною структурою даних Spark. Це незмінна розподілена колекція об'єктів. Кожен набір даних у RDD розділений на логічні розділи, які можуть обчислюватися на різних вузлах кластера.

Kafka — це потенційна платформа для обміну повідомленнями та інтеграції для потокової передачі Spark. Kafka виступає як центральний центр для потоків даних у реальному часі та обробляються за допомогою складних алгоритмів у Spark Streaming. Після обробки даних Spark Streaming може публікувати результати в іншій темі Kafka або зберігати їх у HDFS, базах даних чи панелях інструментів. На рисунку 2.1 зображено цей принцип.

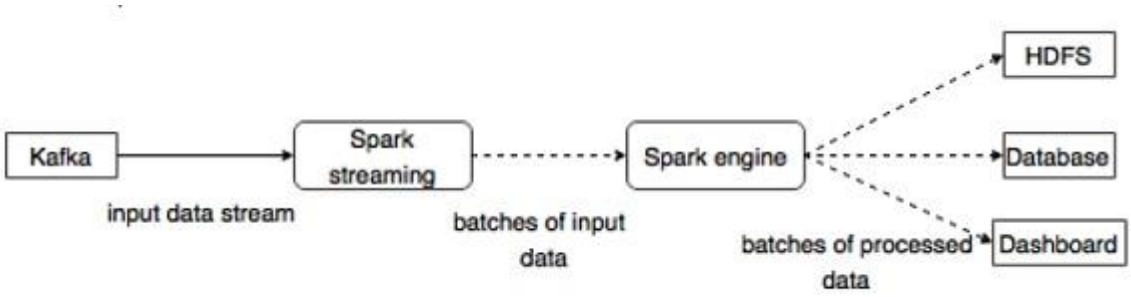

Рисунок 2.1 – Потокове опрацювання даних через інтеграцію Apache Kafka&Spark

Компіляція та виконання програм стандартнs і розглянуте в розділах вище. Коли зібрати весь проєкт та запустити його, то для тестового набору твітів, можна побачити подібний вивід.

```
Popular topics in last 60 seconds (119 total):
#gameinsight (114 tweets)
#iphonegames, (108 tweets)
#iphone, (108 tweets)
#iPhone (63 tweets)
#iphone (32 tweets)
#Apple (22 tweets)
#19) قائداعظم_کا_دشمن_کون tweets)
#GameInsight (18 tweets)
#iPhoneGames (18 tweets)
#mac (15 tweets)

Popular topics in last 10 seconds (119 total):
#gameinsight (114 tweets)
#iphonegames, (108 tweets)
#iphone, (108 tweets)
#iPhone (63 tweets)
#iphone (32 tweets)
#Apple (22 tweets)
#19) قائداعظم_کا_دشمن_کون tweets)
#GameInsight (18 tweets)
#iPhoneGames (18 tweets)
#mac (15 tweets)
-------------------------------------------
Time: 1473602184000 ms
-------------------------------------------
Received 7 kafka messages.
```

Рисунок 2.2 – Вивід програми у вікні терміналу

Безпосередні результати виділення хештегів можна побачити після запуску то результати можна побачити, направивши потік виводу у браузер (див. рис. 2.3). Для цього потрібна модифікація споживача з використанням протоколу HTTP.

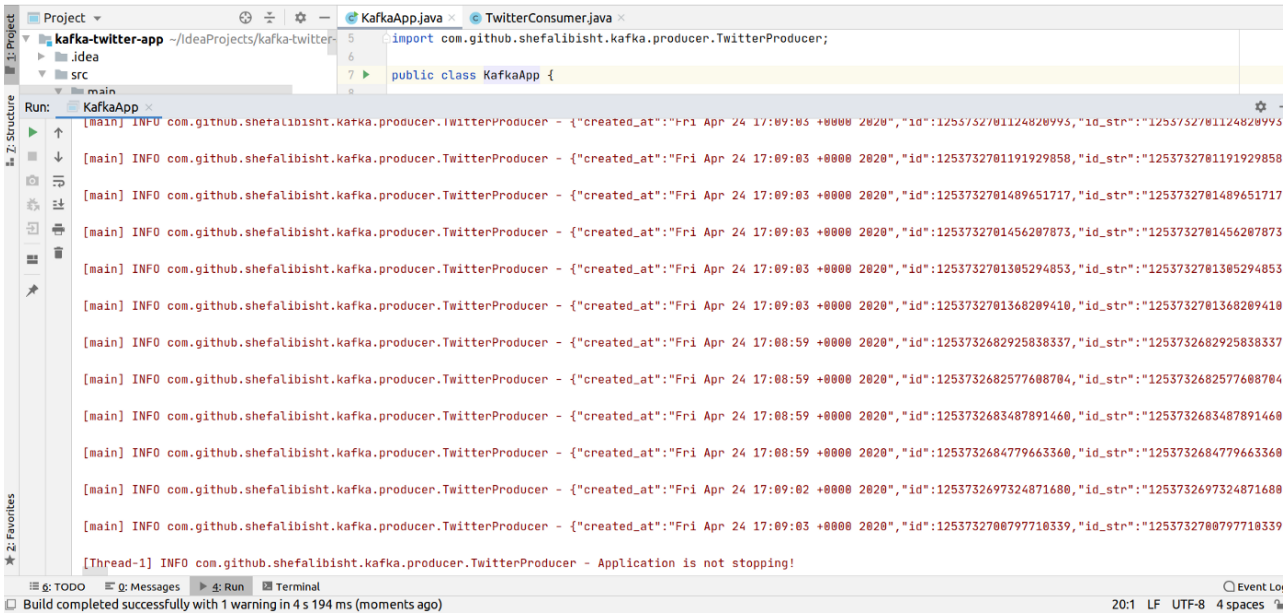

Рисунок 2.3 – Вивід програми в браузері в режимі відладки

РОЗДІЛ 3. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

3.1 Охорона праці та її актуальність в ІТ-сфері

Для підвищення ефективності системи управління охорони праці (СУОП) дуже важлива роль належить формуванню і розвитку інформаційної культури фахівців ІТ-технологій, яка впливає на удосконалення інформаційного контуру сучасних підприємств, дозволяє створювати надійні прогнози щодо стану умов праці, показників здоров'я та працездатності, виробничого травматизму і професійної захворюваності, визначати політику розвитку підприємств, установ та організацій на основі різноманітних стратегій охорони праці (інноваційні, маркетингові, інвестиційні, фінансові, технологічні, диверсифікаційні). Поряд з інформаційною культурою важливо використовувати в рамках СУОП «трикутник» її складових: правову, організаційну, управлінську.

В управлінні охороною праці потрібно реалізувати основні положення, окремі теоретико-методологічні підходи інформаційного менеджменту. Головну роль та відповідальність за стан СУОП мають нести фахівці служби охорони праці сучасного підприємства.

Сучасне суспільство називають постіндустріальним, постекономічним, інформаційним, оскільки йдеться про багатосторонні і кардинальні зміни у розвитку цивілізації.

Інформаційне суспільство передбачає докорінну зміну, яка полягає у перетворенні інформації і знань у головний професійно-виробничий потенціал особистості, соціуму і держави.

На постіндустріальному етапі розвитку суспільства вирішальним фактором стає інформація. Її домінування ініціювала науково-технічна революція, яку ще іменують інформаційною, оскільки нею охоплена будь-яка інтелектуальна діяльність, починаючи з інформаційних образів штучного

інтелекту у нових технологіях, економіки, і продовжуючи інформатизацією суспільства в умова світової глобалізації науки й освіти тощо.

Інформаційні технології розглядаються як потужний важіль економічного зростання України. Для цього необхідні значні стратегічні інвестиції у комп'ютерну та комунікаційну інфраструктуру, програми досліджень і розробок, освітню галузь [42].

Під інформаційною культурою розуміють сукупність, складову НІТ (новітні інформаційні технології), технологічну, правову, психологічну, соціологічну та ергономічну підсистеми, що сприяють спрямованому впливу на протікання соціальних процесів у суспільстві, колективі і вихованню свідомого відношення людини до праці, виконання прав та обов'язків [43].

Поняття інформаційної культури виникло в процесі активізації дослідницької уваги до механізмів інформаційного обміну у зв'язку зі значним підвищення ролі інформації в соціокультурних процесах суспільства, яке розглядають як інформаційне суспільство знань, де в центрі знаходяться інформаційні технології.

Робота з інформацією та інформаційна культура в цілому є одним з найважливіших компонентів спроб компанії управляти змінами. Є три принципові причини, в силу яких сьогодні необхідно дбати про інформаційну культуру компанії.

По-перше, вона все більше і більше стає найважливішою частиною загальної організаційної (корпоративної) культури компанії. Все більше компаній розуміють необхідність перетворень, орієнтованих на задоволення очікувань споживача. Щоб сьогодні впливати на майбутнє, потрібно уявляти собі на що вона буде схожа. А для цього потрібно працювати з різноманітною діловою, професійною, технологічною, соціальною, ринковою та політичною інформацією.

По-друге, інформаційні технології роблять можливим створення в компаніях комп'ютерних мереж, за допомогою яких йде спілкування між

менеджерами, але важливо знати, як люди використовують цю інформацію. Саме по собі створення такої мережі з усіма її робочими станціями і мультимедійними можливостями не гарантує того, що інформація буде використовуватися більш розумно і більш ефективно.

По-третє, для різних функціональних служб, підрозділів та робочих груп сучасних підприємств в сфері охорони праці інформаційна культура різна, а це означає відмінність методологічних підходів до процесів усвідомлення, збору, організації, обробки, поширення і використання інформації. Тому багато менеджерів погодяться з тим, що корпоративна інформаційна культура важлива для вироблення різних стратегій охорони праці та запровадження відповідних заходів з її вдосконалення.

Для деяких галузей, таких як розробка програмного забезпечення, інформаційна культура є необхідною умовою виживання, тому що зміна технологій в розробці програмного забезпечення відбувається кожні 6-8 місяців, а інвестиції на підготовку персоналу і освоєння нової технології величезні і у великих компаніях варіюються від 1,5 до 2 млрд. доларів на рік [46].

Аналіз свідчить, що інформатизація та інтеграція комунікаційного простору України сприяє різкому підвищенню інформаційної та професійної компетентності, ділової активності, стимулюванню конкуренції, створенню інноваційних підприємств та організацій, нових робочих місць, зниженню витрат на утримання управлінського апарату [45].

Поряд із задачами і здобутками окреслилися негативи використання інформаційних технологій:

1) надмірне інформаційне навантаження, суть якого полягає у тому, що кількість корисної інформації, яка надходить до мережі, перевищує психофізіологічні можливості її сприйняття людиною;

2) велика кількість інформації, яка сприймається, але не є корисною для фахівців в даний момент;

3) інформаційний голод, причиною якого є саме надлишок інформації, викликаний інформаційним перенавантаженням;

4) «інформоманія» як хвороба людини, яка робить останню знеособленою, залежною від перебування в інформаційному просторі і роботи з комп’ютером і чому вона віддає перевагу, уникаючи «живого» спілкування з людьми;

5) поява «кіберспільнот», що за своїми соціокультурними характеристиками набагато ближчі до представників інших культур у глобальному інформаційному просторі, ніж до своєї етнонаціональної спільноти чи решти населення, не охопленого Інтернетом;

6) індивідуалізм і дегуманізація способу життя «мешканців» Інтернету – відсутність готовності ділитися своїми знаннями.

Слід розуміти, що комп'ютерні технології, а особливо їх мережі істотно впливають на життєдіяльність людини, припускаючи глобалізацію і технократизацію суспільства. Але в ще більшій мірі цей вплив поширюється безпосередньо на центральну нервову систему, яка звикає працювати в дуже інтенсивному режимі багатозадачності, де вже переважають не тривалі логічні роздуми, а інтуїтивно-реактивні ланцюжки розумових формулювань у зв'язку з величезним обсягом оброблюваної щодня інформації, кількість якої зростає за експоненціальною швидкістю. Виникає припущення, що саме збільшення обсягу інформації та прискорення її обробки людиною може згубно вплинути на розвиток розумових здібностей людини.

Аналіз продуктивності розумової праці в найбільших за чисельністю фахівців IT-фірм показав, що велике значення з точки зору впливу на її результати має організаційна (корпоративна) культура. В цьому напрямі влаштовуються різні тімбілдинги, заходи, тренінги для розвитку персоналу. Також кожен керівник повинен добре розуміти свого співробітника, що саме для нього важливо, що його мотивує. Важливо відвести потрібну роль відповідному співробітнику, щоб він виконував ті завдання, які йому цікаві.

На подібних тренінгах в тому числі повинна розглядатися інформаційна культура працівника, в освоєнні, володінні, мотивуванні, застосуванні, перетворенні інформації із застосуванням сучасних інформаційних технологій і використанням цих умінь в навчанні з охорони праці і в подальшій професійній діяльності. Особливо вони будуть корисні, як доповнення до існуючих інструктажів з охорони праці на підприємстві, або як контроль психологічного стану та взаємовідносин у колективі.

Інформаційна культура як інтегративне утворення абсолютно не зводиться до розрізнених знань, вмінь та навичок роботи за комп'ютером. Вона передбачає інформативну спрямованість цілісної особистості, яка володіє мотивацією до застосування і засвоєння нових даних. Інформаційну культуру можна розглядати, як одну з граней особистісного розвитку промислових робітників. Це шлях універсалізації якостей людини.

Оволодіння інформаційною культурою сприяє реальному розумінню особистістю свого місця, себе і своєї ролі у виробничому колективі. Вона має сприяти формуванню нового покоління фахівців інформаційного суспільства, який повинен володіти наступними навичками: виділення релевантної, значущої інформації, диференціації вихідних даних, розробки інформативних критеріїв її оцінки інформації, вміло використовувати її в рамках СУОП.

Сьогодні продовжує діяти стратегічне правило «Можливості комп’ютерної техніки обмежені тільки нашими уявленнями» [44].

3.2 Шкідлива дія шуту та вібрації і захист від неї

Для запобігання шкідливої дії шуму і вібрації на організм працюючих проводяться технічні, організаційні і медикопрофілактичні заходи.

Одним з основних технічних заходів є зменшення при експлуатації та на стадії проектування, конструювання обладнання причин шуму і вібрації в самому джерелі утворення. Досягають цього завдяки використанню раціональної конструкції обладнання, заміни ударної дії деталей і машин

коливальною, з’єднання елементів гнучкими зв’язками, врівноважування обертових частин механізмів, заміни металевих деталей пластмасовими, забезпечення різних власних частот коливань механізму з частотою збуджуючої сили. Аеродинамічний шум може бути зменшений застосуванням глушників та повітропроводів зі змінним перерізом. Шум трансформаторів (електромагнітний шум) знижується, якщо застосувати листи заліза як складових осердя трансформатора з малою магнітострикцією, серцевини.

Якщо неможливо ізолювати чи знизити шум і вібрацію самого джерела, потрібно:

– ізолювати джерело шуму або вібрації від навколишнього середовища засобами вібро- та звукоізоляції

– раціонально планувати виробничі приміщення, що мають інтенсивні джерела шуму;

– збільшувати звукопоглинання внутрішніх поверхонь приміщення шляхом звукопоглинальних покриттів.

Принцип роботи звукоізоляційних екранів оснований на відбиванні звукової хвилі від різних екранів, стін, кожухів обладнання. Шумливі агрегати слід закривати звукоізоляційними кожухами з виводом назовні органів керування та контрольних приладів. Звукоізоляційні екрани виготовляють з металу, деревини, пластмаси та інших щільних матеріалів. Екрани зсередини покривають звукопоглинаючими матеріалами (скловатою пінополіуретаном), а по периметру кожуха – віброізоляційними підкладками (гума).

Вихідними даними для розрахунку параметрів необхідного екрану є спектр шуму, який необхідно ослабити, кількість екранів, через які проходить шум, їх площа, акустичні характеристики приміщення.

За розрахованими значеннями необхідної звукової ізоляційної здатності екрану підбирається матеріал конструкції й екрану.

Принцип звукопоглинання оснований на явищі трансформації коливальної енергії звуку в теплову через втрати при терті. Найбільші втрати при терті мають

пористі, волокнисті і перфоровані матеріали: поролон, пемзолітові і деревоволокнисті плити тощо.

Енергія звукової хвилі переходить у теплову енергію, причому, ефект звукоізоляції збільшується з ростом частоти звукової хвилі. Звукопоглинаючими матеріалами оббивають стелі, стіни. Щоб одержати ефективну звукоізоляцію, найбільш доцільно застосовувати багатошарові огородження з м'якими прошарками (мінеральна вата).

Важливим технічним рішенням у забезпеченні виробничих умов є вдосконалення ручних віброінструментів. Для цього використовують віброгасіння, змінюють ударний вузол, проводять балансування частин, що обертаються.

Послаблення локальної вібрації і передачі вібрації на підлогу і сидіння досягається засобами віброізоляції і вібропоглинання, застосуванням пружинних і гумових амортизаторів, прокладок тощо. Для обмеження поширення вібрацій через ґрунт, між фундаментом і ґрунтом залишають повітряні проміжки, які називаються акустичними розривами.

В останні роки знаходять застосування динамічні віброгасники, в яких створюються вібрації, що співпадають по частоті і протилежні по фазі вібрації машини, коливання якої необхідно зменшити.

До організаційних заходів по боротьбі з шумом та вібрацією на виробництві відносяться: впровадження раціонального режиму праці і відпочинку, обмеження часу роботи при використанні ручного інструменту, який створює вібрацію.

Глушники звуку застосовуються для зменшення шуму аеродинамічних установок (вентиляторів, пневмоінструментів, газотурбінних, дизельних, компресорних установок). Вони поділяються на активні, які поглинають звукову енергію, що на них поступила, і реактивні, які відбивають цю енергію. Потужні джерела шуму як правило розміщують в окремих приміщеннях, які віддалені від постійних робочих місць.

Ізоляційні кабіни або екрани застосовують як екрани робочих місць для зменшення зовнішніх шумів.

Якщо не вдається зменшити рівень шуму і вібрації на робочому місці до нормативних значень та необхідно використовувати засоби індивідуального захисту: рукавиці, взуття, навушники, м'які шоломи, які зменшують рівень звукового тиску на 40-50 дБ.

У процесі виробництв, експлуатації і зберігання радіоелектронних засобів можуть виникати механічні і динамічні дії, що характеризуються широким діапазоном частот коливань, а також амплітудою, прискоренням і часом дії. Рівень механічних дій визначається умовами транспортування й експлуатації.

Необхідно розрізняти два види механічних дій: удари і вібрації. Удар виникає, коли апаратура отримує швидку зміну прискорення (піддаються удару входи кабелів, джгути, резистори, конденсатори, напівпровідникові діоди і тріоди, силові трансформатори, дроселі тощо). Вібрації – довготривалі знакозмінні процеси, які впливають на роботу апаратури при безпосередньому контакті з джерелом коливань або через повітряне середовище.

У результаті дії вібрацій і удару можуть бути наступні ушкодження апаратури: порушення герметичності через псування паяльних, зварних і клеєних швів і появи тріщин у метало-скляних спаях; повне руйнування корпусів або окремих їх частин через механічний резонанс або циклічну втому; обривання монтажних зв'язків, відшарування багатошарових друкованих плат, руйнування підставок; вихід з ладу електричних контактів; модуляція розмірів хвилеводних трактів; коаксіальних кабелів, конденсаторів змінної ємності, коливальних контурів, електровакуумних приладів, зміщення положення органів настроювання і управління.

Під впливом вібрацій може статись зміна параметрів напівпровідникових приладів, вольт амперних характеристик діодів, транзисторів. Все це призводить до руйнування конструкцій за рахунок явищ втоми.

Радіоелектронна апаратура (РЕА) повинна мати віброміцність, вібростійкість, ударостійкість.

Захист РЕА здійснюється наступними групами методів:

– зменшується інтенсивність джерел вібрації шляхом балансування, зменшення зазорів, віброізоляції джерела вібрацій;

– зменшується величина дій, що передається апаратом шляхом віброізоляції, демпфірування, виключення резонансів, активного віброзахисту за допомогою ексцентриків, маятників, гіроскопів;

– використання найбільш добротні і жорсткі компоненти і вузли;

– застосовуються амортизатори.

Захист часом, захист віддалю, усунення джерела тепловиділення, теплоізоляція, охолодження гарячої поверхні, забезпечення тепловіддачі тіла людини та індивідуальні засоби захисту.

Захист часом передбачає обмеження часу перебування робітника в зоні дії інфрачервоного випромінювання. Потужність випромінювання можна знизити за рахунок конструкторських і технологічних рішень (зміною нагрівання виробів у нагрівальних пічках індукційним нагріванням та ін.) і за рахунок покриття поверхні, яка нагрівається, тепло ізолювальним матеріалом.

Якщо теплоізоляція неможлива, тоді захист від прямої дії інфрачервоного випромінювання здійснюється екрануванням.

Екрани можуть бути прозорими, напівпрозорими і непрозорими.

У свою чергу вони поділяються на тепловідбивальні, тепловідвідні та теплопоглинальні; стаціонарні і нестаціонарні.

Застосовують також прозору водяну завісу у вигляді суцільної тонкої водяної плівки. Вода є активним поглиначем інфрачервоного випромінювання.

Перегрівання людини попереджують раціональним режимом пиття, режимом праці та гідро процедурами. Спецодяг виготовляється з незаймистого, стійкого до інфрачервоного випромінювання, м’якого і повітронепроникного

матеріалу (тканина з металевим покриттям відбиває 90 % інфрачервоного випромінювання).

Для захисту очей застосовують світлофільтри зі спеціального жовто-зеленого або синього скла.

Першочергові заходи – це конструкторські і технологічні рішення, які виключають генерацію або понижують інтенсивність випромінювання. Спеціальні засоби захисту (екранування джерел випромінювання, фарбування стін у світлі кольори) попереджують розповсюдження і знижують інтенсивність цих випромінювань у виробничих приміщеннях. Очі захищають окулярами або щитками зі склом – світлофільтром. Для захисту шкіри використовують мазі з речовинами – світлофільтрами для цих променів (салол, саліцилово-метиловий ефір та ін.), а також спецодяг з бавовняних тканин і грубововняного сукна. Руки захищають рукавицями.

ВИСНОВОК

Завданням кваліфікаційно роботи було виконати огляд сервісу Apache Kafka для потокової обробки даних через механізм обробки повідомлень.

В ході виконання роботи було виконано:

1. Призначення та архітектуру сервісу Apache Kafka.
2. Установку сервісу на комп’ютер та його налаштування для розробки програми, що ілюструє його роботу
3. Огляд програмного інтерфейсу для створення продюсерів та споживачів повідомлень в брокері Apache Kafka.
4. Для перевірки роботи сервісу було розроблено прикладну програму мовою Java для програмного опрацювання твітів соціальної мережі з використанням API доступу до цієї соціальної мережі та сервісу Apache Spark, який реалізує власне опрацювання даних. В нашому прикладі це було звичайне виведення даних в потік по протоколу HTTP.

В розділі з охорони праці та безпеки в надзвичайних ситуаціях виконано огляд відповідних питань згідно завдання консультанта з розділу та описано аспекти використання обчислювальної техніки з контексті забезпечення вимог безпеки.

ПЕРЕЛІК ПОСИЛАНЬ

1. Kumar, Manish, and Chanchal Singh. Building Data Streaming Applications with Apache Kafka. Packt Publishing Ltd, 2017.

2. Estrada, Raul. Apache Kafka 1.0 Cookbook: Over 100 practical recipes on using distributed enterprise messaging to handle real-time data. Packt Publishing Ltd, 2017.

3. Narkhede, Neha, Gwen Shapira, and Todd Palino. Kafka: the definitive guide: real-time data and stream processing at scale. " O'Reilly Media, Inc.", 2017.

4. Bejeck, Bill. Kafka Streams in Action: Real-time apps and microservices with the Kafka Streams API. Simon and Schuster, 2018.

5. Koutanov, Emil. Effective Kafka: A Hands-On Guide to Building Robust and Scalable Event-Driven Applications with Code Examples in Java. Obsidian Dynamics, 2020.

6. Scott, Dylan, Viktor Gamov, and Dave Klein. Kafka in Action. Simon and Schuster, 2022.

7. Kumar, Manish, and Chanchal Singh. Building Data Streaming Applications with Apache Kafka. Packt Publishing Ltd, 2017.

8. Kreps, Jay. I heart logs: Event data, stream processing, and data integration. " O'Reilly Media, Inc.", 2014.

9. Dunning, Ted, and Ellen Friedman. Streaming architecture: new designs using Apache Kafka and MapR streams. " O'Reilly Media, Inc.", 2016.

10. Dean, Alexander, and Valentin Crettaz. Event Streams in Action: Real-time Event Systems with Kafka and Kinesis. Manning Publications, 2019.

11. Akidau, Tyler, Slava Chernyak, and Reuven Lax. Streaming systems: the what, where, when, and how of large-scale data processing. " O'Reilly Media, Inc.", 2018.

12. Житецький В.Ц. Охорона праці користувачів комп’ютерів. Навчальний посібник. - Вид. 2-ге, доп. - Львів: Афіша, 2000. - 176с.

13. Навакатіян О.О., Кальниш В.В., Стрюков С.М. Охорона праці користувачів комп’ютерних відеодисплейних терміналів. - К.:1997. - 400с.

ДОДАТКИ

ДОДАТОК А

Програмний код класу продюсера

```
//import util.properties packages
import java.util.Properties;

//import simple producer packages
import org.apache.kafka.clients.producer.Producer;

//import KafkaProducer packages
import org.apache.kafka.clients.producer.KafkaProducer;

//import ProducerRecord packages
import org.apache.kafka.clients.producer.ProducerRecord;

//Create java class named “SimpleProducer”
public class SimpleProducer {

   public static void main(String[] args) throws Exception{

      // Check arguments length value
      if(args.length == 0){
         System.out.println("Enter topic name");
         return;
      }

      //Assign topicName to string variable
      String topicName = args[0].toString();

      // create instance for properties to access producer configs
      Properties props = new Properties();

      //Assign localhost id
      props.put("bootstrap.servers", “localhost:9092");

      //Set acknowledgements for producer requests.
      props.put("acks", “all");

      //If the request fails, the producer can automatically
retry,
      props.put("retries", 0);

      //Specify buffer size in config
      props.put("batch.size", 16384);

      //Reduce the no of requests less than 0
      props.put("linger.ms", 1);
```

```
      //The buffer.memory controls the total amount of memory
available to the producer for buffering.
      props.put("buffer.memory", 33554432);

      props.put("key.serializer",

"org.apache.kafka.common.serialization.StringSerializer");

      props.put("value.serializer",

"org.apache.kafka.common.serialization.StringSerializer");

      Producer<String, String> producer = new KafkaProducer
         <String, String>(props);

      for(int i = 0; i < 10; i++)
         producer.send(new ProducerRecord<String,
String>(topicName,
            Integer.toString(i), Integer.toString(i)));
               System.out.println("Message sent successfully");
               producer.close();
   }
}
```

ДОДАТОК Б

Програмний код додатку простого споживача

```
import java.util.Properties;
import java.util.Arrays;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.ConsumerRecord;

public class SimpleConsumer {
   public static void main(String[] args) throws Exception {
      if(args.length == 0){
         System.out.println("Enter topic name");
         return;
      }
      //Kafka consumer configuration settings
      String topicName = args[0].toString();
      Properties props = new Properties();
      props.put("bootstrap.servers", "localhost:9092");
      props.put("group.id", "test");
      props.put("enable.auto.commit", "true");
      props.put("auto.commit.interval.ms", "1000");
      props.put("session.timeout.ms", "30000");
      props.put("key.deserializer",

"org.apache.kafka.common.serialization.StringDeserializer");
      props.put("value.deserializer",

"org.apache.kafka.common.serialization.StringDeserializer");
      KafkaConsumer<String, String> consumer = new KafkaConsumer
         <String, String>(props);
      //Kafka Consumer subscribes list of topics here.
      consumer.subscribe(Arrays.asList(topicName))
      //print the topic name
      System.out.println("Subscribed to topic " + topicName);
      int i = 0;
      while (true) {
         ConsumerRecords<String, String> records = con-
sumer.poll(100);
         for (ConsumerRecord<String, String> record : records)

         // print the offset,key and value for the consumer
records.
         System.out.printf("offset = %d, key = %s, value = %s\n",
            record.offset(), record.key(), record.value());
      }
   }
}
```

ДОДАТОК В

Програмний код TwitterKafkaProducer

```
import java.util.Arrays;
import java.util.Properties;
import java.util.concurrent.LinkedBlockingQueue;

import twitter4j.*;
import twitter4j.conf.*;

import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerRecord;

public class KafkaTwitterProducer {
   public static void main(String[] args) throws Exception {
      LinkedBlockingQueue<Status> queue = new
LinkedBlockingQueue<Sta-tus>(1000);

      if(args.length < 5){
         System.out.println(
            "Usage: KafkaTwitterProducer <twitter-consumer-key>
            <twitter-consumer-secret> <twitter-access-token>
            <twitter-access-token-secret>
            <topic-name> <twitter-search-keywords>");
         return;
      }

      String consumerKey = args[0].toString();
      String consumerSecret = args[1].toString();
      String accessToken = args[2].toString();
      String accessTokenSecret = args[3].toString();
      String topicName = args[4].toString();
      String[] arguments = args.clone();
      String[] keyWords = Arrays.copyOfRange(arguments, 5,
arguments.length);

      ConfigurationBuilder cb = new ConfigurationBuilder();
      cb.setDebugEnabled(true)
         .setOAuthConsumerKey(consumerKey)
         .setOAuthConsumerSecret(consumerSecret)
         .setOAuthAccessToken(accessToken)
         .setOAuthAccessTokenSecret(accessTokenSecret);

      TwitterStream twitterStream = new
TwitterStreamFactory(cb.build()).get-Instance();
      StatusListener listener = new StatusListener() {

         @Override
```

```
        public void onStatus(Status status) {
            queue.offer(status);

            // System.out.println("@" +
status.getUser().getScreenName()
                + " - " + status.getText());
            // System.out.println("@" +
status.getUser().getScreen-Name());

            /*for(URLEntity urle : status.getURLEntities()) {
                System.out.println(urle.getDisplayURL());
            }*/

            /*for(HashtagEntity hashtage :
status.getHashtagEntities()) {
                System.out.println(hashtage.getText());
            }*/
        }

        @Override
        public void onDeletionNotice(StatusDeletionNotice
statusDeletion-Notice) {
            // System.out.println("Got a status deletion notice
id:"
                + statusDeletionNotice.getStatusId());
        }

        @Override
        public void onTrackLimitationNotice(int
numberOfLimitedStatuses) {
            // System.out.println("Got track limitation notice:" +
                num-berOfLimitedStatuses);
        }

        @Override
        public void onScrubGeo(long userId, long upToStatusId) {
            // System.out.println("Got scrub_geo event userId:" +
userId +
            "upToStatusId:" + upToStatusId);
        }

        @Override
        public void onStallWarning(StallWarning warning) {
            // System.out.println("Got stall warning:" + warning);
        }

        @Override
        public void onException(Exception ex) {
            ex.printStackTrace();
        }
    };
    twitterStream.addListener(listener);
```

```
      FilterQuery query = new FilterQuery().track(keyWords);
      twitterStream.filter(query);

      Thread.sleep(5000);

      //Add Kafka producer config settings
      Properties props = new Properties();
      props.put("bootstrap.servers", "localhost:9092");
      props.put("acks", "all");
      props.put("retries", 0);
      props.put("batch.size", 16384);
      props.put("linger.ms", 1);
      props.put("buffer.memory", 33554432);

      props.put("key.serializer",
         "org.apache.kafka.common.serializa-
tion.StringSerializer");
      props.put("value.serializer",
         "org.apache.kafka.common.serializa-
tion.StringSerializer");

      Producer<String, String> producer = new
KafkaProducer<String, String>(props);
      int i = 0;
      int j = 0;

      while(i < 10) {
         Status ret = queue.poll();

         if (ret == null) {
            Thread.sleep(100);
            i++;
         }else {
            for(HashtagEntity hashtage : ret.getHashtagEntities())
{
               System.out.println("Hashtag: " +
hashtage.getText());
               producer.send(new ProducerRecord<String, String>(
                  top-icName, Integer.toString(j++),
hashtage.getText()));
            }
         }
      }
      producer.close();
      Thread.sleep(5000);
      twitterStream.shutdown();
   }
}
```