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 систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

Магістр
(назва освітнього ступеня)

на тему: **Методи та засоби розробки гнучкої та маштабованої ETL-системи для обробки великих обсягів даних**

Виконав: студент(ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

(підпис) Луговий А.С. (прізвище та ініціали)

Керівник (підпис) Тиш Є.В. (прізвище та ініціали)

Нормоконтроль (підпис) Луцик Н.С. (прізвище та ініціали)

Завідувач кафедри (підпис) Осухівська Г.М. (прізвище та ініціали)

Рецензент (підпис) Цуприк Г.Б. (прізвище та ініціали)

Тернопіль
2024

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 систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г.М.*
(підпис) (прізвище та ініціали)
«11» листопада 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Луговому Андрію Сергійовичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Методи за засоби розробка гнучкої та масштабованої ETL- системи для обробки великих обсягів даних*

Керівник роботи Тиш Євгенія Володимирівна, кандитат технічних наук
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « *29* » *листопада 2024* року № 4/7-1144

2. Термін подання студентом завершеної роботи *20 грудня 2024 року*

3. Вихідні дані до роботи *Технологія Apache Airflow, різновиди ETL-систем, Python, відкриті Api Kaggle*

4. Зміст роботи (перелік питань, які потрібно розробити)

Вступ
1 Основи та сучасні підходи до створення ETL-систем
2 Методи та інструменти розробки гнучких і масштабованих ETL-систем
3 Реалізація та тестування розробленого рішення
4 Охорона праці та безпека в надзвичайних ситуаціях
Висновки

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

1. Актуальність і мета дослідження.
2. Завдання, об'єкт і предмет, наукова новизна і практична цінність дослідження.
3. Архітектура програмної системи
4. Блок-схема алгоритму ETL-системи
5. Діаграма процесу створення запиту до відкритої API Kaggle
6. ER-діаграма бази даних та скрипт підключення та запису до PostgreSQL
7.Методи порівняння підходів до зберігання даних, автоматизації ETL-процесів
8. Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Осухівська Г.М., зав.каф. КС*	*05.12.2024*	
Безпека в надзвичайних ситуаціях	*Стручок В.С., старший викладач*	*10.12.2024*	

7. Дата видачі завдання *11.11.2024*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Аналіз існуючих методів та засобів розв'язання науково-технічних проблем, що відповідають тематиці кваліфікаційної роботи.*	*20.11.2024р. – 30.11.2024р.*	*Викон.*
2.	*Теоретичні аспекти та актуальні підходи до розробки ETL-систем*	*02.12.2024р.*	*Викон.*
3.	*Методологія та інструментарій для створення гнучких і масштабованих ETL-рішень*	*09.12.2024р.*	*Викон.*
4.	*Практична реалізація та тестування розробленого ETL-рішення*	*17.12.2023р.*	*Викон.*
5.	*Охорона праці та безпека в надзвичайних ситуаціях*	*18.12.2024р.*	*Викон.*
6.	*Оформлення пояснювальної записки і графічного матеріалу*	*19.12.2024р.*	*Викон.*
7.	*Попередній захист кваліфікаційної роботи магістра*	*20.12.2024р.*	*Викон.*
8.	*Захист кваліфікаційної роботи магістра*	*26.12.2024р.*	*Викон.*

Студент _____ *Луговий А.С.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Тиш Є.В.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Луговий А.С. Методи та засоби розробки гнучкої ETL-системи для обробки великих обсягів даних. Робота на здобуття кваліфікаційного ступеня магістра: спец. 123 — комп'ютерна інженерія / наук.кер. Тиш Є.В. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. — 69 с.

Ключові слова: ETL-система, обробка даних, архітектурна модель, масштабованість, алгоритм, платформа, Apache Airflow, PostgreSQL візуалізація даних, API, інтеграція даних.

Кваліфікайна робота присвячена присвячена розробленню автоматизованих ETL-процесів для збору, обробки та збереження даних із використанням Docker-контейнерів та бази даних PostgreSQL.

У першому розділі проведено аналіз сучасних підходів до реалізації ETL-процесів, розглянуто інструменти для екстракції, трансформації та завантаження даних. Оцінено можливості використання Apache Airflow для підвищення ефективності обробки даних.

У другому розділі описано вибір інструментів, таких як FastAPI, Docker і PostgreSQL, та розроблено архітектуру ETL-системи. Система автоматично збирає дані з зовнішніх API, трансформує їх і завантажує у базу даних.

Третій розділ присвячено реалізації системи. Деталізовано створення Docker-контейнера для виконання ETL-процесу, інтеграцію FastAPI для запитів користувача та підключення до PostgreSQL. Описано розробку програмного забезпечення для збору, обробки та збереження даних.

У останньому розділі детально розглянуто аспекти охорони праці та безпеки в надзвичайних ситуаціях під час розробки і експлуатації автоматизованих ETL-процесів.

ANNOTATION

Luhovyi A.S. Methods and tools for developing a flexible and scalable ETL system for processing large volumes of data. Master's qualification thesis: specialty 123 – Computer Engineering / Scientific advisor: Tysh Ye.V. Ternopil: Ternopil Ivan Puluj National Technical University, 2024. — 69 p.

Keywords**:** ETL system, data processing, architectural model, scalability, algorithm, platform, Apache Airflow, PostgreSQL, data visualization, API, data integration

The qualification work is dedicated to the development of automated ETL processes for data collection, processing, and storage using Docker containers and a PostgreSQL database.

In the first chapter, an analysis of modern approaches to implementing ETL processes is conducted, and tools for data extraction, transformation, and loading are reviewed. The potential of using Apache Airflow to enhance data processing efficiency is evaluated.

The second chapter describes the selection of tools, including FastAPI, Docker, and PostgreSQL, and the development of the ETL system architecture. The system automatically collects data from external APIs, transforms it, and loads it into the database.

The third chapter focuses on the implementation of the system. It details the creation of a Docker container for executing the ETL process, the integration of FastAPI for user queries, and the connection to PostgreSQL. The chapter also outlines the development of software for data collection, processing, and storage.

The final section thoroughly examines aspects of occupational safety and emergency security during the development and operation of automated ETL processes.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасному світі даних, що швидко розвивається, компанії та організації стикаються з серйозними проблемами в ефективному зборі, обробці, аналізі та зберіганні даних.

За таких умов компанія повинна приймати оперативні та добре сплановані рішення для ефективного управління. Цьому може сприяти лише встановлення систем, які гарантуватимуть своєчасну обробку даних та швидку доставку їх на аналітичну платформу. Цю проблему можуть вирішити системи ETL, які здійснюють автоматизацію процесу вилучення, перетворення та завантаження даних у сховище.

Система ETL, перш за все, повинна бути дуже гнучкою і масштабованою, оскільки дані мають мінливий характер і їх обсяг постійно збільшується. Гнучка система ETL може враховувати зміни в потоках даних, різних форматах і джерелах інформації. Це допомагає фірмам швидко реагувати на нові виклики, змінюючи або додаючи вимоги до інформації. Масштабовані рішення забезпечують стабільну роботу системи, навіть якщо обсяг інформації зростає.

Сучасні ETL-рішення зосереджені на спрощенні інтеграції різних джерел даних, таких як бази даних, файли, API або дані в режимі реального часу. Це дозволяє ефективно інтегрувати різнорідні дані в єдину аналітичну базу, що прискоює роботу і скорочує час, необхідний для роботи з даними.

Іншим важливим питанням є безпека та надійність обробки даних: система ETL повинна забезпечувати безпечний доступ до джерел даних, підтримку резервного копіювання та відновлення інформації, а також контроль доступу до даних на всіх етапах їх обробки.

Розробка таких систем, що поєднує в собі автоматизацію, маштабованість і гнучкість, необхідну компаніям, які прагнуть отримати конкурентну перевагу завдяки обробці та аналізу об’ємних обсягів інформації.

Мета кваліфікаційної роботи. Розроблення методів та інструментів для створення гнучкої та масштабованої ETL-системи, здатної обробляти великі обсяги даних з відкритих API та надавати доступ до обробленої інформації.

Завдання кваліфікаційної роботи:

– провести огляд існуючих рішень та технологій для розробки ETL-систем, включаючи інструменти для автоматизації та оркестрації процесів;

– розробити архітектуру гнучкої та масштабованої ETL-системи для обробки великих обсягів даних із використанням Apache Airflow;

– реалізувати автоматизовану систему очищення даних перед їх збереженням у базі даних;

– розробити REST API на основі FastAPI для забезпечення доступу до оброблених даних;

– контейнеризувати систему за допомогою Docker для забезпечення гнучкого розгортання та масштабованості;

– проведення експериментів щодо продуктивності та масштабованості системи в умовах зростання обсягів даних.

Об'єкт дослідження – процеси розробки ETL-систем для обробки великих обсягів даних.

Предмет дослідження – методи та технології автоматизації ETL-процесів із використанням сучасних інструментів, таких як Apache Airflow, PostgreSQL, Docker, а також API для забезпечення доступу до оброблених даних.

Методи дослідження: Для виконання поставлених завдань у роботі використано такі методи:

– аналіз і узагальнення – при дослідженні існуючих рішень та технологій для розробки ETL-систем, зокрема інструментів автоматизації та оркестрації процесів.

– формалізація – при розробці архітектури гнучкої та масштабованої ETL-системи, здатної обробляти великі обсяги даних.

– моделювання, проектування та програмування – реалізації автоматизованої системи очищення даних та контейнеризації системи за допомогою Docker.

– експериментальні методи – при проведенні тестування продуктивності та масштабованості системи в умовах зростання обсягів даних, а також при оцінці її гнучкості в різних середовищах.

Наукова новизна одержаних результатів:

– запропоновано архітектуру гнучкої та масштабованої ETL-системи, яка дозволяє автоматизувати процеси збору, трансформації та завантаження даних про COVID-19, забезпечуючи високу продуктивність і масштабованість;

– набув подальшого розвитку у застосуванні підходу до інтеграції різних етапів роботи з даними, використовуючи сучасні інструменти і технології, такі як Apache Airflow для оркестрації ETL-процесів та Docker для контейнеризації системи, що забезпечило гнучке розгортання на різних середовищах;

– розроблено автоматизовану систему трансформації даних, яка дозволяє ефективно обробляти великі обсяги інформації та зберігати її у PostgreSQL для подальшого аналізу та використання.

Практичне значення результатів кваліфікаційної роботи полягає у реалізації спеціалізованої системи ETL для трансформації інформації та побудована архітектура з автоматизованими процесами завантаження та трансформації даних до власної локально бази даних, що зменшуючи витрати часу та ресурсів на їх обробку.

Публікації. Результати дослідження апробовано на XII науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології», XII міжнародній науково-технічній конференція молодих учених та студентів «Актуальні задачі сучасних технологій» у вигляді тез конференцій.

1. Луговий А.С., Тиш Є.В. Методи та засоби роботи ETL-процесів в умовах високих навантажень. Матеріали XII міжнародної науково-практичної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (11-12 грудня 2023 року). Тернопіль: ТНТУ. 2024. С. 420.

2. Луговий А.С., Тиш Є.В. Методи оптимізації продуктивності etl-систем у багаторівневих аналітичних платформах. Матеріали XII науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (18-19 грудня 2024 року). Тернопіль: ТНТУ. 2024. С. 162.

Структура роботи. Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається із вступу, 4 розділів, висновків, списку використаних джерел та додатку (-ів). Обсяг роботи: пояснювальна записка – 69 арк. формату А4, графічна частина – 6 аркушів формату А1

РОЗДІЛ 1
ОСНОВИ ТА СУЧАСНІ ПІДХОДИ ДО СТВОРЕННЯ ETL-СИСТЕМ

1.1. Визначення та роль ETL-систем у сучасному середовищі обробки даних

З появою великих обсягів даних (Big Data) та необхідністю ефективної їх обробки, ETL-системи стали невід'ємною частиною сучасного середовища управління інформацією. ETL (Extract, Transform, Load) є основним процесом у побудові сховищ даних, аналітичних платформ та інтеграції даних із різних джерел [1]. У цій роботі детально розглядаються історія виникнення, ключові концепції, завдання, які вирішують ETL-системи, а також їх роль у сучасних умовах.

До появи ETL-систем у 1970-1980-х роках переважно використовувалися ручні підходи до інтеграції даних. Дані оброблялися програмістами, які писали спеціальні скрипти для збирання інформації з різних джерел, її консолідації та завантаження у сховища. Ці процеси були громіздкими, схильними до помилок і не могли ефективно масштабуватися. З ростом обсягів даних та розвитком концепції реляційних баз даних виникла потреба в автоматизованих системах, які могли б обробляти дані систематично та стандартизовано [2].

ETL-системи були створені через кілька факторів, таких як зростання обсягів даних, необхідність інтеграції численних джерел інформації, потреба в очищенні даних і вимога до реального часу для прийняття рішень. Вони автоматизують процеси збору, трансформації та завантаження даних, усуваючи дублікати, помилки та стандартизуючи інформацію для подальшого аналізу.

Ці системи допомагають інтегрувати дані з різних джерел, підвищують ефективність роботи завдяки автоматизації та забезпечують масштабованість для роботи з великими обсягами даних у хмарних і розподілених середовищах.

У сучасному середовищі ETL-системи є критичними для підготовки даних для аналітики, реалізації Data Lake та Data Warehouse, а також для роботи з великими даними та потоковою обробкою в реальному часі, що дозволяє оперативно забезпечувати можливість ухвалення рішень на основі актуальних даних

Основними компонентами сучасних ETL-систем є:

1) Збирання даних – етап, на якому система автоматично отримує дані з різних джерел (бази даних, API, файли тощо) для подальшої обробки;

2) Трансформації даних – процес очищення, нормалізації та перетворення даних для їх подальшого використання у аналітичних системах;

3) Завантаження даних – перенесення оброблених даних у центральне сховище для подальшого аналізу та використання.

Основний принцип ETL — автоматизація рутинних завдань інтеграції даних з мінімальним втручанням людини (рис. 1.1). Це дозволяє суттєво зменшити витрати часу й ресурсів.

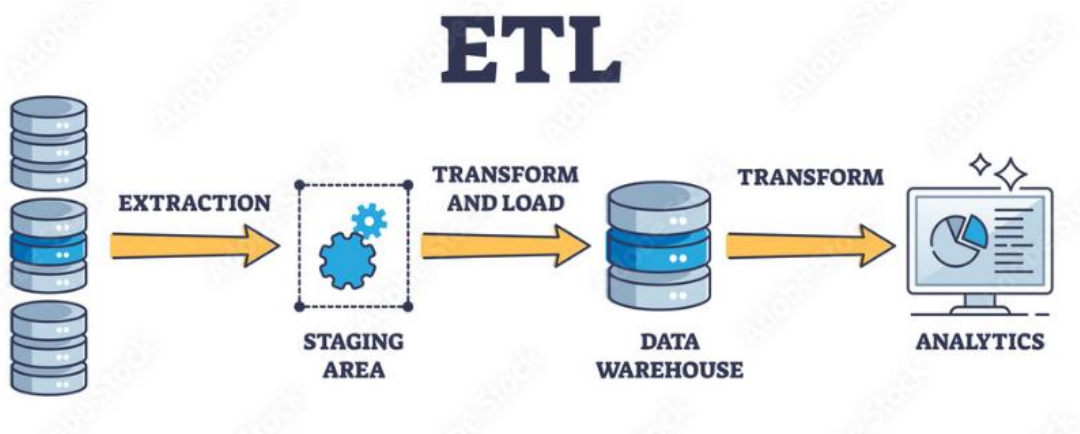

Рис. 1.1. Схема роботи ETL-системи, що включає етапи збирання, трансформації та завантаження даних до сховища.

Зважаючи на тенденції росту даних, ETL-системи стають ключовим елементом побудови інфраструктури даних [3]. Їх еволюція йде в напрямку ELT (Extract, Load, Transform), де обробка здійснюється вже у сховищах, а також у використанні штучного інтелекту для автоматичного налаштування процесів.

1.2. Огляд сучасних підходів до розробки ETL-систем

Сучасний ландшафт розробки ETL (екстракція, трансформація, завантаження) систем базується на різноманітних підходах, які відрізняються за технологічними

принципами, методологією дизайну та способами доставки даних. Еволюція обробки даних, зростання їхніх обсягів та нові бізнес-вимоги спонукали до значних змін у методах розробки ETL-систем. Від класичних підходів до інноваційних інструментів штучного інтелекту — спектр можливостей став надзвичайно широким, а системи гнучкішими, продуктивнішими та масштабованими.

Класичні або традиційні підходи до розробки ETL-систем базуються на використанні спеціалізованих програмних продуктів [4]. Ці системи працюють за принципом батч-обробки, що включає три основні етапи: вилучення даних із різних джерел (бази даних, файли, API), трансформацію даних у проміжному середовищі та завантаження їх у цільові сховища. Основною перевагою таких систем є їхня надійність, стабільність та відповідність вимогам великих корпоративних середовищ (рис. 1.2). Проте вони мають певні недоліки: високу вартість ліцензування, обмежену масштабованість та складність інтеграції із сучасними потоковими джерелами даних.

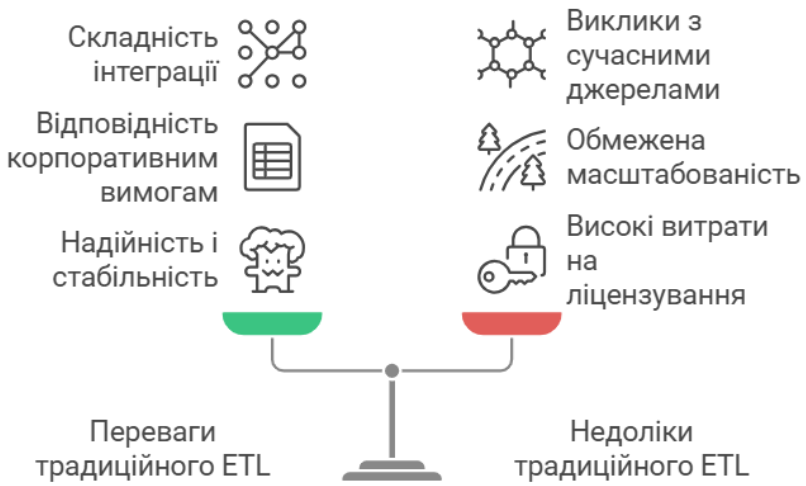

Рис. 1.2. Оцінка традиційного ETL-підходу: переваги та недоліки

З розвитком хмарних технологій на зміну традиційним підходам прийшли хмарні ETL-системи, такі як AWS Glue, Google Dataflow і Azure Data Factory. Ці інструменти призначені для обробки даних безпосередньо в хмарному середовищі, забезпечуючи гнучкість операцій і швидке масштабування обчислювальних ресурсів. Модель «оплата за фактом» дає змогу компаніям оптимізувати витрати, оскільки вони

платять тільки за використовувані ресурси. Крім того, хмарна платформа підтримує інтеграцію з різними джерелами, що значно спрощує процес розробки. Однак основними проблемами такого підходу залишаються залежність від підключення до Інтернету та проблеми безпеки даних у хмарі.

Розподілені системи, такі як Apache Hadoop та Apache Spark, представляють наступний етап у розробці ETL-систем [5]. Ці платформи використовують паралельну обробку даних на багатьох вузлах, що дозволяє ефективно працювати з великими обсягами інформації. Особливістю розподілених систем є їхня здатність інтегруватися з потоковими інструментами, такими як Apache Kafka, що дозволяє обробляти дані в реальному часі (табл. 1.1). Цей підхід особливо корисний для обробки великих даних, однак його недоліками є складність налаштування та адміністрування, а також потреба у високій кваліфікації команди розробників.

Таблиця 1.1

Порівняльна характеристика підходів

Підхід	Переваги	Недоліки
Традиційні системи	Надійність, багатофункціональність	Висока вартість, складність масштабування
Хмарні платформи	Гнучкість, масштабованість	Залежність від хмари, питання безпеки
Розподілені системи	Висока продуктивність, реальна паралельність	Складність налаштування та адміністрування

1.3. Вимоги до платформ для обробки даних

Системи ETL важливі для великих організацій, оскільки дають змогу об'єднувати й обробляти великі обсяги даних із різних джерел, а сучасні технології, як-от Hadoop, Spark та інші інструменти для роботи з великими даними, допомагають підвищити ефективність і масштабованість таких процесів. Використання інструментів оркестровки, таких як Apache Airflow, значно спрощує автоматизацію складних робочих процесів.

Дослідження спрямовані на розробку систем, розташованих у галузі сучасних інформаційних технологій, зокрема в галузі управління та обробки даних. ETL-системи є важливим компонентом великих інформаційних систем, оскільки забезпечують автоматичний збір, перетворення та завантаження даних із різних джерел у центральне сховище. Ці системи є важливим аспектом розвитку сучасних інформаційних технологій, оскільки забезпечують ефективну інтеграцію великих даних, їх подальший аналіз і використання в аналітичних процесах.

Для створення ефективних ETL-систем важливо враховувати низку вимог до платформ обробки даних. Основними характеристиками таких платформ є висока продуктивність, масштабованість, відмовостійкість і можливість інтеграції з різними джерелами даних. Продуктивність платформи визначається її здатністю обробляти великі обсяги інформації в найкоротші терміни. Це дуже важливо в реальних умовах, коли затримки в обробці можуть вплинути на прийняття бізнес-рішень (рис. 1.3).

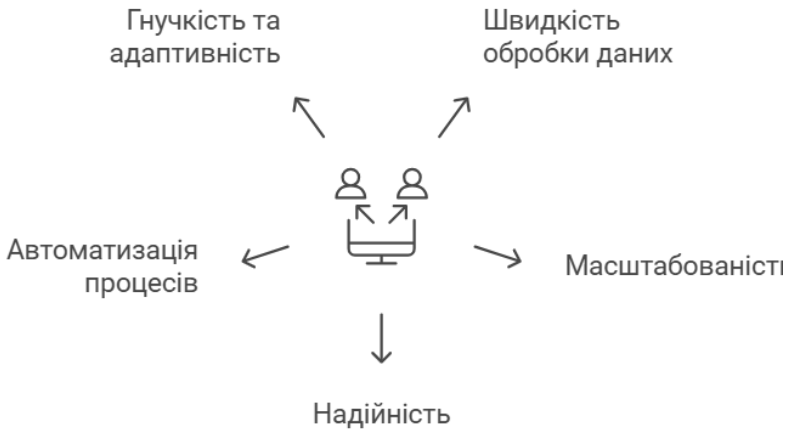

Рис. 1.3. Критерії ефективності системи ETL

Масштабованість забезпечує адаптацію системи до змінних обсягів даних без значної модернізації, а стійкість — гарантує безперервність процесів при можливих збоях або збої в роботі окремих компонентів. Інтеграційні можливості передбачають роботу з різними типами даних: структурованими, напівструктурованими та неструктурованими, що вимагає гнучких підходів до ETL-процесів та підтримки широкого спектру форматів даних. Додаткові вимоги включають безпеку обробки даних, зокрема підтримку шифрування, аутентифікації та контролю доступу.

До ключових аспектів оцінювання продуктивності ETL-систем слід віднести темпи обробки інформації. Така система повинна обробляти дані в режимі реального часу або близькому до реального, е дозволить забезпечувати актуальність інформації для користувачів.

Наступний ключ до оцінки є масштабованість. Вона має бути здатна ефективно обробляти зростаючі обсяги даних без втрати продуктивності. Надійність передбачає коректну обробку даних навіть у випадку відмов систем або некоректних вхідних даних.

Використання інструментів для автоматизації дозволяє значно знизити навантаження на технічних спеціалістів, забезпечуючи стабільне та швидке виконання ETL-процесів. Система повинна легко адаптуватися до змін у джерелах даних, форматах або вимогах бізнесу, дозволяючи швидко впроваджувати нові функції чи обробляти нові типи даних без суттєвих змін у загальній архітектурі

1.4. Аналіз існуючих технологій для автоматизації процесів повязаних з даними

У сучасних умовах обробка даних виходить за рамки простих завдань інтеграції та перетворення. Зростаючий обсяг інформації, різноманітність джерел і вимоги до швидкості обробки змушують компанії шукати ефективні інструменти для автоматизації процесів управління даними. Давайте розглянемо деякі ключові технології, які сприяють автоматизації та підвищенню ефективності управління даними.

Apache Airflow — це потужний інструмент для створення, управління та моніторингу робочих процесів, який став стандартом де-факто у сфері автоматизації роботи з даними. Основною метою Airflow є автоматизація складних конвеєрів обробки даних за допомогою графів спрямованих ациклічних процесів (DAG).

Особливості Apache Airflow:

1) Завдяки використанню Python як основної мови програмування, розробники можуть легко визначати призначення, їхню залежність і порядок виконання, що забезпечує гнучкість у налаштуванні робочих процесів.

2) Інтеграція з популярними інструментами: Airflow підтримує численні підключення до баз даних, хмарних сервісів (AWS, Google Cloud, Azure) і сховищ даних, що дозволяє використовувати його для Інтеграція з популярними інструментами.

3) Графічний інтерфейс дозволяє відслідковувати статус виконання завдань, повторно запускати невдалі операції та аналізувати продуктивність процесів (рис. 1.4).

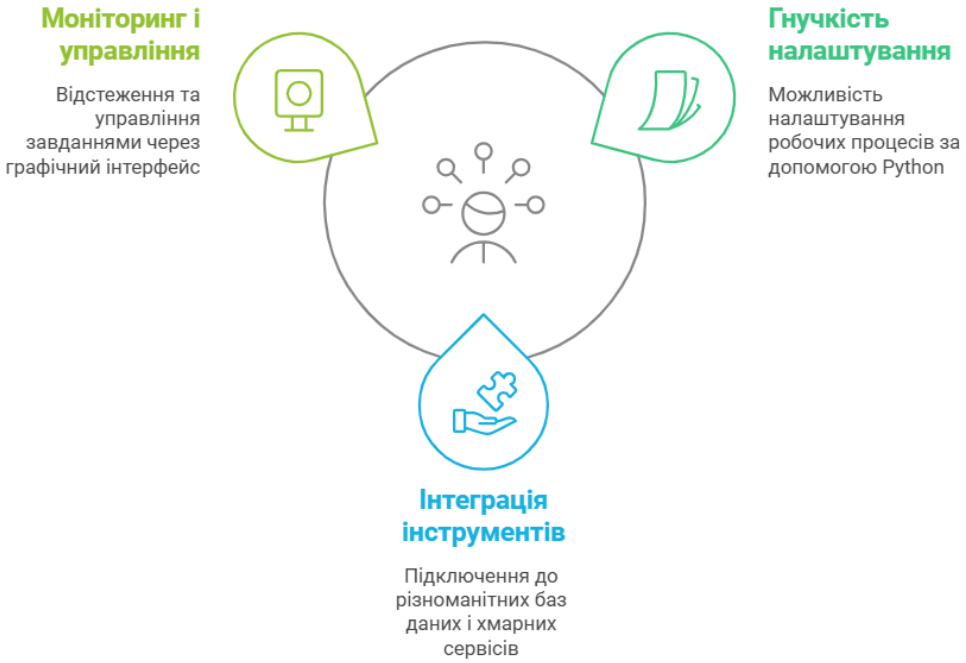

Рис. 1.4. Переваги Docker для автоматизації розгортання програмного забезпечення

Airflow ідеально підходить для сценаріїв, де є багато взаємозалежних завдань, наприклад, агрегування даних з декількох джерел, обробка інформації та завантаження в аналітичні сховища. Водночас Airflow не призначений для виконання завдань із високою швидкістю в реальному часі, через що його часто поєднують з іншими технологіями.

Docker став революційним інструментом для автоматизації розгортання програмного забезпечення [6]. У контексті роботи з даними Docker забезпечує контейнеризацію середовища, що дозволяє уникнути конфліктів між залежностями, бібліотеками чи версіями програм.

Переваги Docker для автоматизації систем ETL:

1) Контейнери забезпечують однакове середовище для роботи незалежно від платформи чи операційної системи (рис. 1.5.). Це особливо корисно при запуску скриптів для обробки даних, які залежать від специфічних бібліотек.

2) Завдяки легковаговим контейнерам можна легко запускати великі обчислювальні процеси паралельно, розподіляючи їх маштабаючи.

3) Docker дозволяє легко автоматизувати процеси тестування, побудови та розгортання обробки даних, що знижує кількість ручної роботи та помилок.

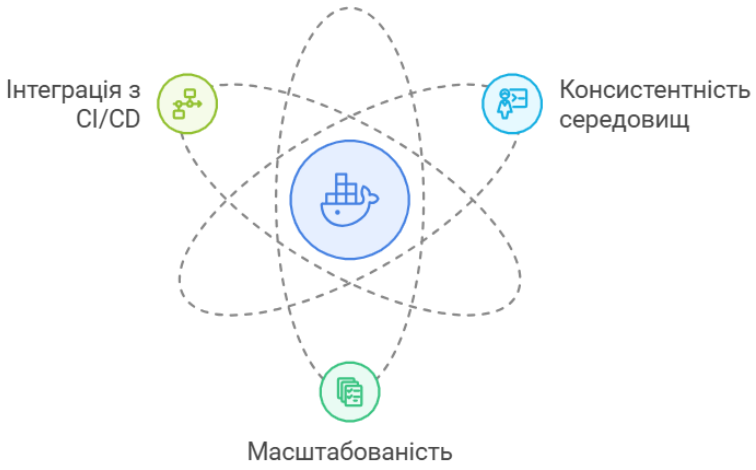

Рис. 1.5. Переваги Docker для автоматизації розгортання програмного забезпечення

Наприклад, Docker часто використовують для контейнеризації ETL-конвеєрів або мікросервісів, які виконують специфічні завдання, такі як очищення чи агрегування.

Нинішні хмарні платформи, такі як AWS, Google Cloud та Azure, надають вбудовані інструменти для автоматизації процесів обробки даних [7]. Використання цих платформ дозволяє поєднувати переваги контейнеризації (Docker) та оркестрації процесів (Airflow) з можливостями масштабування та високої доступності.

До прикладів автоматизації в хмарі можна віднест такі технології:

1) AWS Lambda, дозволяє запускати задачі без необхідності керування інфраструктурою. Це зручно для виконання невеликих, але часто повторюваних операцій.

2) Інструменти, як Google Dataflow, забезпечують поточну обробку даних у режимі реального часу.

3) Використання хмарних баз даних та об'єктних сховищ, таких як Amazon S3 або Google BigQuery, спрощує доступ до даних для автоматизованих процесів.

РОЗДІЛ 2
МЕТОДИ ТА ІНСТРУМЕНТИ РОЗРОБКИ ГНУЧКИХ І МАСШТАБОВАНИХ ETL-СИСТЕМ

2.1. Вибір архітектурної моделі для побудови системи обробки даних

У цьому розділі проведено дослідження на різні архітектурні моделі для побудови ETL-системи, щоб визначити найефективніший підхід для забезпечення гнучкості, масштабованості та продуктивності обробки великих обсягів даних. У процесі роботи було порівняно традиційну серверно-клієнтську архітектуру, мікросервісну модель і архітектуру потокової обробки, тестуючи їх на відповідність ключовим вимогам системи.

Дослідження включали можливості монолітної архітектури, яка передбачає централізовану обробку даних. У результаті тестування виявив її переваги, зокрема простоту реалізації та централізоване управління [8]. Однак під час порівняння з іншими методами з'ясувалося, що ця модель має обмежену масштабованість і недостатню стійкість до змін (рис. 2.1.). Як результат, можна стверджувати, що цей підхід підходить для ETL-систем, які працюють з малими або середніми обсягами даних і не потребують високої продуктивності.

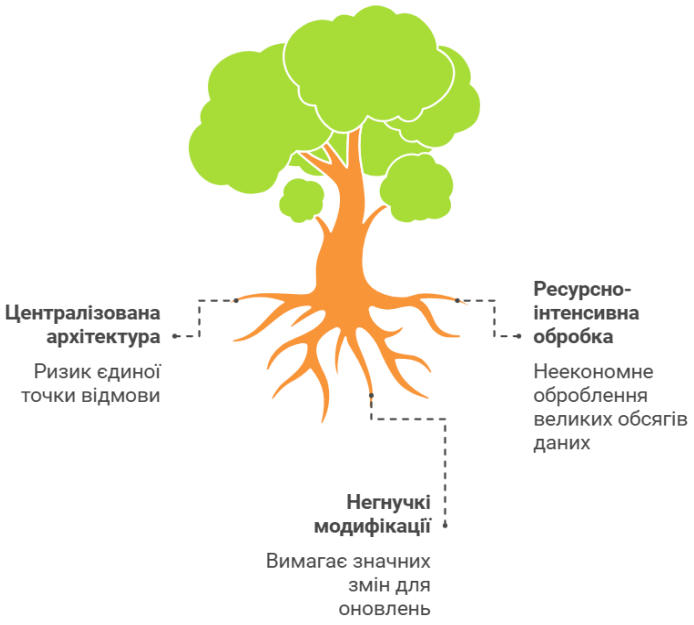

Рис. 2.1. Серверно-клієнтська архітектура

Основні характеристики та обмеження серверно-клієнтської архітектури:

1) Вся інформація збирається в одному центрі, що зменшує складність налаштування, управління та моніторингу процесів

2) Монолітні системи прості у виконанні, всі компоненти мають єдину функцію. Немає зобов'язань створювати складну інфраструктуру чи розповсюджувати послуги.

3) Основним недоліком цієї архітектури є складність роботи з великими обсягами даних. Один сервер може бути перевантажений і стати єдиним найважливішим компонентом збою, ця система не підходить для великих даних.

4) Внесення змін, таких як додавання нових джерел даних або оновлення бізнес-логіки, вимагає змін у всій програмі. Це може призвести до простоїв, що має велике значення для систем, які мають постійну обробку даних.

Отже, така архітектура підходить для ETL-систем, які обробляють малі або середні обсяги даних і не мають високих вимог до швидкості обробки чи адаптивності.

У процесі дослідження виявилось, що вона дозволяє розділити систему на незалежні компоненти, кожен з них виконує конкретну задачу [9]. Завдяки цьому забезпечується гнучкість і можливість масштабування окремих сервісів. Важливо, що після дослідження великої кількості інформації на прикладах обробки великих даних можна дійти висновку, що висока відмовостійкість системи підтверджує її ефективність для реальних бізнес-задач. На підставі цього можу стверджувати, що мікросервісна архітектура є оптимальним вибором для масштабних ETL-проєктів.

Ця архітектура є оптимальною для обробки великих обсягів інформації у реальному часі та систем, що вимагають високої надійності й гнучкості (рис. 2.2).

Рис. 2.2. Переваги мікросервісної архітектури

Мікросервісна архітектура полегшує розподіл праці між різними сервісами, кожен етап процесу ETL. Наприклад, одна служба призначена для отримання даних із зовнішніх джерел, інша служба відповідає за перетворення даних у щось корисне, а інша служба відповідає за збереження результатів у базі даних [10]. Цей метод полегшує розробку окремих частин системи окремо, без потенціалу негативного впливу на інші компоненти.

Завдяки незалежному складу мікросервісів система стала гнучкою і легко адаптується до змін. Це полегшує додавання нових послуг або зміну існуючих без переривання всієї системи. Це особливо важливо для компаній, які мають швидкі темпи зростання або потреби бізнесу яких змінюються.

Ще одна перевага мікросервісів полягає в тому, що вони можуть бути індивідуально налаштовані залежно від навантаження. Наприклад, якщо

перетворення даних є більш ресурсомістким, ви можете лише збільшити цей сервіс без негативного впливу на інші частини системи.

Крім того, мікросервісна архітектура має високий ступінь відмовостійкості. У разі втрати однієї служби інші служби продовжують працювати, що підвищує загальну стабільність системи (рис. 2.3). Мікросервіси легко ремонтувати та підтримують автоматичне встановлення, що значно полегшує їх обслуговування та керування.

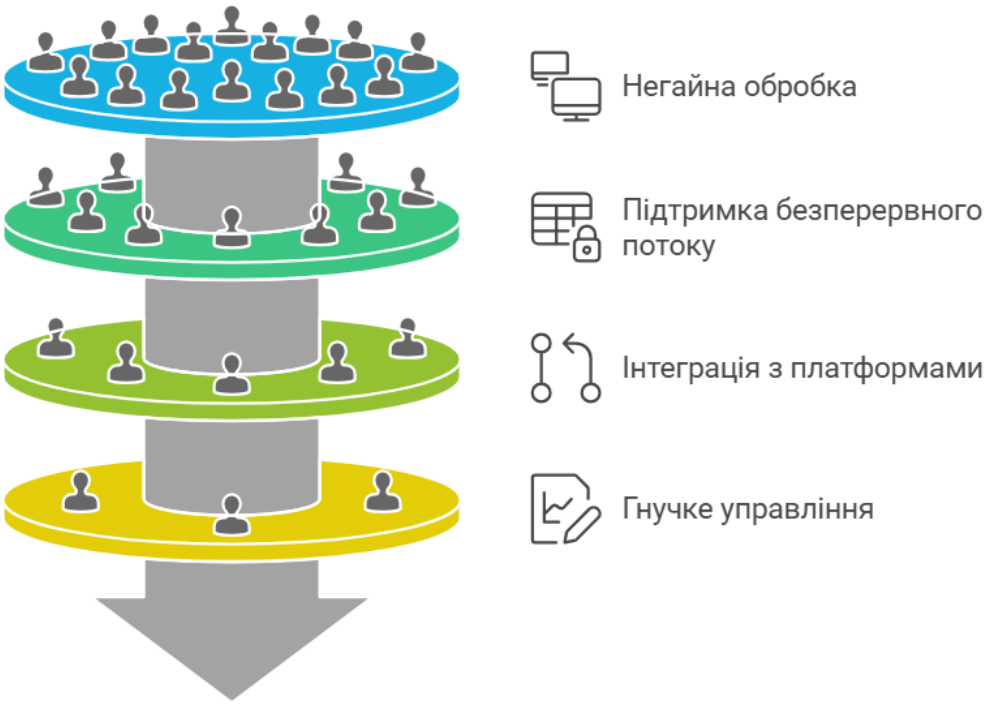

Рис. 2.3. Оптимізована обробка даних

У ході роботи було досліджено архітектуру потокової обробки, орієнтуючись на потреби системи у реальному часі. Я тестував різні платформи для потокової обробки, такі як Apache Kafka та Spark Streaming, і виявив їхню ефективність для обробки даних у режимі реального часу [11]. Аналіз показав, що цей метод забезпечує безперервну роботу системи та мінімальні затримки під час обробки даних. На основі проведених тестів я дійшов висновку, що архітектура потокової обробки ідеально підходить для завдань, де критично важлива швидкість та безперервна аналітика.

Обробка потоку даних гарантує, що дані обробляються в міру їх надходження, так що зміни можуть бути негайно відображені в системі. Це особливо важливо для завдань, які вимагають швидкого реагування, таких як моніторинг даних з пристроїв Інтернету речей, аналіз веб-трафіку або відстеження поведінки користувачів у додатках.

Завдяки підтримці безперервного потоку даних, системі не потрібно зберігати і обробляти великі обсяги інформації одночасно. Це забезпечує стабільну роботу процесу ETL навіть при високих навантаженнях і знижує ризик перевантаження системи.

Архітектура потокової обробки легко інтегрується з сучасними платформами, такими як Apache Kafka, Apache Flink і Spark Streaming, забезпечуючи високу продуктивність і надійність при обробці потоків даних. Крім того, потокова архітектура є гнучкою та масштабованою. Це дозволяє швидко додавати нові джерела даних та адаптувати їх до мінливих бізнес-вимог. Наприклад, система може отримувати дані з журналів, відгуків користувачів та інших джерел, обробляти їх в режимі реального часу і миттєво передавати для аналізу.

Таким чином, у другому розділі описано порівняльний аналіз різних архітектурних моделей, їх переваги та обмеження [12]. На підставі отриманих результатів рекомендую використовувати мікросервісну архітектуру для побудови ETL-системи з високими вимогами до продуктивності та масштабування, а потокову обробку — для завдань у реальному часі

2.2. Методи автоматизації етапів Extract, Transform, Load

Автоматизація ETL-процесів (Extract, Transform, Load) є невід’ємною складовою сучасних систем обробки даних. Ці процеси дозволяють забезпечити ефективний збір, трансформацію та збереження інформації з різноманітних джерел для подальшого аналізу та використання [13]. Кожен етап ETL має свої особливості та інструменти, що визначають його ефективність та адаптивність до бізнес-вимог.

2.2.1 Методи вилучення даних із зовнішніх джерел

Описуючи методи автоматизації етапу Extract було зосереджено дослідження на різних підходів до видобування даних із зовнішніх джерел. Цей процес є першим і дуже важливим етапом ETL, оскільки від якості й способу отримання даних залежить подальша їх обробка та інтеграція.

Отримання даних через відкриті API це один із найбільш надійних і автоматизованих способів. Багато сучасних платформ і сервісів надають API (Application Programming Interface), через яке можна отримувати структуровані дані у форматах JSON або XML [14].

Дослідження включало використання REST для запитів до сервісів, таких як Twitter, Google Maps або Kaggle. Робота з API-ключами для автентифікації та налаштування доступу. Оцінка переваг API, таких як структурованість даних, автоматичне оновлення та підтримка великого обсягу запитів.

Я виявив, що використання відкритих API дозволяє ефективно видобувати дані, уникаючи проблем із парсингом. Це особливо зручно для роботи з динамічними даними, які постійно оновлюються, наприклад, курси валют чи прогнози погоди.

Коли API недоступний, можна використовувати веб-скрапінг, щоб отримати дані з вебсайтів. Дослідження включало використання бібліотек на кшталт BeautifulSoup (Python) та Puppeteer (JavaScript) для отримання інформації зі сторінок [15]. Аналіз обмежень, таких як дотримання правил вебсайтів (robots.txt) і можливі юридичні наслідки. Цей метод може бути ефективним, але потребує додаткових зусиль для підтримки, особливо якщо структура сайту змінюється.

Ще один підхід, який було проаналізовано, це завантаження повної HTML-сторінки та локальна обробка даних (табл. 2.1). Методика роботи є збереження сторінки як файлу HTML або використання бібліотек для збереження структури сайту. Парсинг HTML локально за допомогою інструментів, таких як lxml або Cheerio [16]. Цей підхід може бути корисним для разових завдань, коли потрібен доступ до специфічних даних, але він менш зручний для автоматизації в масштабі.

Таблиця 2.1

Порівняння підходів

Метод	Простота реалізації	Надійність	Гнучкість	Приклади використання
Відкриті API	Висока	Висока	Середня	OpenWeatherMap, GitHub
Web scraping	Середня	Низька	Висока	BeautifulSoup, Selenium
Завантаження HTML	Середня	Середня	Середня	Збір архівів або сторінок

Все таке акцент був зосереджени на відкриті API. Тому що цей спосіб виявилося найбільш ефективним і універсальним способом.

Основною перевагою такого підходу є швидкість і надійність, оскільки структуровані дані можна отримати з мінімальною затримкою. Запит можна інтегрувати в конвеєр ETL за допомогою бібліотек, таких як запити Python або axios JavaScript, тому процес можна легко автоматизувати. Крім того, більшість API надають документацію, автентифікацію та захист даних, що робить їх зручними та безпечними у використанні.

Для роботи з відкритими API можуть знадобитися такі технології:

1) Postman для тестування запитів і вивчення документації API.

2) Python із бібліотеками requests та json для написання запитів і обробки відповіді.

3) AWS Lambda або AWS Glue для автоматизації процесу видобування даних у хмарі.

Таким чином, зосереджуючись на відкритих API, можна забезпечити ефективне видобування даних, придатних для інтеграції в ETL-процеси, і мінімізував труднощі, пов’язані з парсингом чи обробкою сирих даних.

Підсумовуючи, найкращий вибір для автоматизації етапу Extract — використання відкритих API, якщо вони доступні. API забезпечують структуровані дані, їх легко інтегрувати в ETL-процеси та підтримувати автоматизацію через інструменти на кшталт Apache Airflow або Python-скриптів. У разі недоступності API,

варто розглянути web scraping як альтернативу, проте необхідно враховувати ризики блокування та складність підтримки.

2.2.2 Інструменти та технології для обробки і трансформації даних

Наступним етап є Transform, це ключ для підготовки даних до аналізу або завантаження в базу даних [17]. На цьому етапі дані змінюються, очищуються та структуруються відповідно до вимог бізнес-логіки чи моделей.

У процесі трансформації даних було проаналізовано та описано різні підходи та інструменти, які дозволяють ефективно обробляти та змінювати дані залежно від поставлених завдань. Основною мовою для роботи обрав Python завдяки його універсальності, широкому спектру бібліотек і можливості інтеграції з іншими системами. Основна бібліотека для роботи з табличними даними – Pandas – забезпечує зручний інструментарій для обробки даних: від фільтрації та сортування до створення агрегацій і об'єднання наборів даних. Pandas дозволяє працювати з різними форматами, такими як CSV, Excel і JSON, що робить її ідеальною для задач середнього рівня складності.

Для більш ресурсомістких задач можна бібліотеку Dask, яка дозволяє працювати з великими обсягами даних, розподіляючи обчислення між кількома ядрами процесора. Цей підхід виявився ефективним, коли обсяг даних перевищував оперативну пам'ять комп'ютера [18]. Ще одним інструментом для масштабованої обробки даних можна використовувати PySpark – інструмент для розподілених обчислень, який забезпечує високу швидкість і продуктивність у кластерному середовищі. Проте PySpark вимагає більшої попередньої підготовки, що робить його менш зручним для швидкої інтеграції в невеликі проекти.

Для автоматизації ETL-процесів доречно застосовувати Airflow, який дозволяє створювати комплексні пайплайни з мінімальними витратами на підтримку. Також слід подумати про використання Jupyter Notebook, що дає змогу інтерактивно тестувати трансформації даних, створювати візуалізації та аналізувати результати.

На маю думку Python має основні інструментом завдяки своїй гнучкості, активній спільноті та підтримці великої кількості бібліотек. Для роботи з табличними даними Pandas забезпечила простоту реалізації й універсальність [19]. Таким чином, ефективна трансформація даних стала можливою завдяки поєднанню сучасних інструментів і гнучкого підходу до вирішення завдань.

2.2.3Стратегії завантаження та збереження даних у цільових сховищах

Етап завантаження даних (Load) є завершальним у процесі ETL. Він передбачає збереження оброблених даних у цільовому сховищі, яке буде використовуватися для аналізу або візуалізації. Для цього існує кілька підходів і типів баз даних, кожен з яких має свої плюси та мінуси.

Реляційні бд, такі як PostgreSQL і MySQL, є стандартом для роботи зі структурованими даними.

PostgreSQL підтримує складні SQL-запити, роботу з JSONB, геопросторовими даними та добре масштабується для середніх обсягів даних.

MySQL популярний завдяки швидкості виконання запитів і простоті налаштування, але менш функціональний.

Реляційні бд гарантує стабільність, дотримання ACID-принципів та зручність роботи зі структурованими таблицями [20]. Однак вони менш ефективні для великих обсягів швидкозмінних або неструктурованих даних.

Нереляційні бд (NoSQL), такі як MongoDB і Cassandra, краще підходять для роботи з напівструктурованими або неструктурованими даними.

MongoDB зберігає дані у форматі JSON і є гнучким інструментом для швидкозмінних даних.

Cassandra оптимізована для великих обсягів розподілених даних.

NoSQL-сховища забезпечують масштабованість і швидкий доступ до даних, проте не мають єдиного стандарту та підтримки складних транзакцій.

Для реалізаії проєкту було обрано PostgreSQL завдяки наступним перевагам:

1) Підтримка сучасних функцій, таких як робота з JSONB, що дозволяє зберігати напівструктуровані дані.

2) Інтеграція з Python за допомогою бібліотеки psycopg2.

3) Вільне програмне забезпечення з підтримкою активної спільноти користувачів.

Зрештою, вибір цільового сховища залежить від характеристик даних, вимог до їх обробки та подальшого використання (табл. 2.2). PostgreSQL забезпечив баланс між функціональністю, гнучкістю та простотою інтеграції, що робить його оптимальним вибором для завантаження даних у моєму проєкті.

Таблиця 2.2

Порівняння баз даних

Тип бази даних	Приклади	Переваги	Недоліки
Реляційні	PostgreSQL, MySQL	Строга схема, SQL, надійність	Повільна для великих даних
Нереляційні	MongoDB, DynamoDB	Гнучка схема, підтримка JSON	Немає складних SQL, складна інтеграція
Хмарні сховища	AWS Redshift, Snowflake	Масштабованість, аналітика	Висока вартість, потрібна оптимізація

Для більшості ETL-процесів оптимальним вибором є використання PostgreSQL, оскільки це реляційна база даних із розширеними можливостями, що дозволяє працювати як зі структурованими, так і напівструктурованими даними. Для обробки великих обсягів даних із високою частотою змін або неструктурованими даними слід розглядати NoSQL (MongoDB) або Data Warehouses (BigQuery).

2.3. Інструменти та технології для автоматизації ETL

Еволюція технологій ETL (Extract, Transform, Load) проходила декілька ключових фаз, починаючи від ручних процесів і переходячи до сучасних автоматизованих платформ з підтримкою штучного інтелекту та хмарних технологій.

Ранні етапи розвитку ETL-технологій припадають на 1970-1990-ті роки, коли

основним підходом було використання традиційних процедур SQL для ручного виконання процесів екстракції, трансформації та завантаження даних [21]. Також у цей період для автоматизації передачі даних між системами активно застосовували скрипти на базі Bash та Perl.

У 1990-х роках і на початку 2000-х відбувся розвиток спеціалізованих ETL-інструментів (табл. 2.3). Поява комерційних платформ, таких як Informatica PowerCenter, IBM DataStage та Microsoft SQL Server Integration Services (SSIS), значно спростила автоматизацію обробки даних і зробила ETL-процеси доступнішими для бізнесу.

Таблиця 2.3

Огляд сучасних інструментів для автоматизації ETL

Інструмент	Тип	Основні особливості	Платформи	Масштабованість
Informatica PowerCenter	Комерційний	Складний інтерфейс, багато інтеграцій	Windows, Unix	Висока
Talend Data Integration	Відкритий вихідний код / Комерційний	Гнучкий інтерфейс, ручне кодування	Windows, Linux	Середня до високої
Apache NiFi	Відкритий вихідний код	Автоматизація потоків, легкий інтерфейс	Windows, Linux, MacOS	Висока
AWS Glue	Хмарний	Інтеграція з AWS, серверлесс	AWS	Дуже висока
Google Cloud Dataflow	Хмарний	Масштабування, підтримка Apache Beam	Google Cloud	Дуже висока
Azure Data Factory	Хмарний	Інтеграція з Power, BI, Azure сервіси	Azure	Дуже висока
Apache Airflow	Відкритий вихідний код	Оркестрація, підтримка Python	Windows, Linux, MacOS	Висока

У 2010-х роках відзначається зростання інструментів для роботи з великими даними[22]. Такі технології, як Apache Hadoop і Apache Spark, відкрили нові можливості для обробки масивних обсягів інформації, змінивши традиційні підходи до ETL-процесів.

Сучасний етап розвитку ETL-технологій, що розпочався у 2020-х роках, характеризується широким використанням хмарних рішень. Платформи на основі хмарних технологій, такі як AWS Glue, Google Cloud Dataflow і Azure Data Factory,

забезпечують масштабованість, інтеграцію зі штучним інтелектом і пропонують low-code/no-code інтерфейси, що зменшує потребу у написанні коду.

Опис сервісів та їх технологічних особливостей:

1) AWS Glue – хмарне ETL-рішення від Amazon Web Services, яке забезпечує серверлесс архітектуру та інтегрується з іншими сервісами AWS, такими як S3, Redshift, та Lambda. Забезпечує виконання ETL-процесів без потреби керувати інфраструктурою.

2) Google Cloud Dataflow – це платформа для обробки потокових та пакетних даних, побудована на основі Apache Beam. Забезпечує високу продуктивність і можливість масштабування для обробки вел иких обсягів даних.

3) Azure Data Factory – хмарне платформа від Microsoft, що дозволяє створювати, планувати та оркеструвати процеси ETL.

4) Apache Airflow – це інструмент для оркестрації робочих процесів з відкритим вихідним кодом, який дозволяє автоматизувати складні функції ETL за допомогою створення DAG (Directed Acyclic Graphs). Підтримує використання Python та інтеграцію з іншими популярними бібліотеками для роботи з даними.

Таким чином, вибір ETL-інструмента залежить від конкретних потреб організації, включаючи вартість, необхідність масштабування та можливості інтеграції з існуючими системами.

2.4. Вибір методів для реалізації масштабованої обробки даних

У цьому розділі розглянуто різні методи, які застосовуються для забезпечення масштабованої обробки даних у ETL-системах. Порівняння проводилося за кількома критеріями, зокрема за ефективністю, масштабованістю та продуктивністю [23]. Основний акцент зроблено на таких підходах, як паралельна обробка, розподілене зберігання та обробка даних, а також використання індексів і кешування. Додатково тестувалася обробка даних у реальному часі. Кожен із цих методів має свої переваги,

що дозволяють значно підвищити продуктивність системи та ефективно управляти великими обсягами даних.

Паралельна обробка — це один із методів, що я детально досліджував і тестував для скорочення часу обробки великих обсягів даних (рис. 2.4.). В основі цього методу лежить поділ великих обсягів даних на менші частини, які можуть оброблятися одночасно кількома обчислювальними одиницями. Це дозволяє забезпечити швидке опрацювання великих масивів інформації, зменшуючи загальний час обробки.

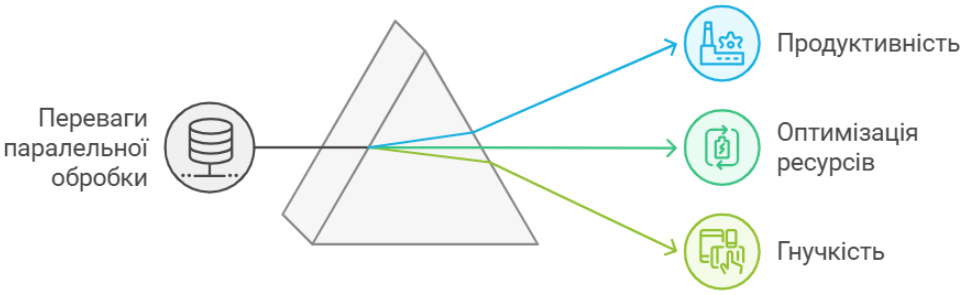

Рис. 2.4. Розкриття переваг пералельної обробки

Наприклад, було проведено порівняння використання Apache Spark як одного з провідних інструментів для паралельної обробки. Spark ефективно розподіляє завдання на підзадачі, які обробляються паралельно на різних вузлах кластера. Під час тестування Spark демонстрував ефективне розбиття лог-файлів на блоки для подальшої обробки, що дозволяло значно скоротити час, необхідний для обробки терабайт даних.

Паралельна обробка має кілька важливих переваг. По-перше, значно підвищується продуктивність, оскільки операції розбиваються на менші частини і виконуються одночасно на різних вузлах. Це дозволяє значно скоротити час обробки даних. По-друге, ресурси використовуються більш ефективно, оскільки вони оптимально розподіляються між вузлами кластера. Система залишається гнучкою і

легко масштабується, так що більші обсяги даних можна обробляти, просто додаючи нові вузли.

У дослідженні було зосереджено увагу на розподіленій обробці даних, яка дозволяє розділяти обробку великих обсягів даних між кількома фізичними машинами або вузлами в кластері [24]. Це забезпечує підвищену масштабованість та відмовостійкість системи.

Наприклад, було проаналізовано використання Apache Hadoop для розподіленої обробки даних. Hadoop, завдяки своїй файловій системі HDFS та механізму MapReduce, дозволяє ефективно обробляти великі обсяги даних, розподіляючи їх між вузлами кластера. У ході тестування Hadoop продемонстрував високу ефективність під час обробки даних із соціальних мереж (рис. 2.5). Система автоматично перенаправляє завдання на інші вузли у разі відмови, що забезпечує надійність та стійкість до збоїв.

Переваги розподіленої обробки даних:

- обробка великих обсягів даних здійснюється завдяки можливості розподілу обробки між багатьма вузлами;
- дані зберігаються та обробляються на кількох вузлах, що мінімізує ризик втрати даних і збоїв;
- робота кластера є оптимізованою, оскільки ресурси розподіляються відповідно до завантаження.

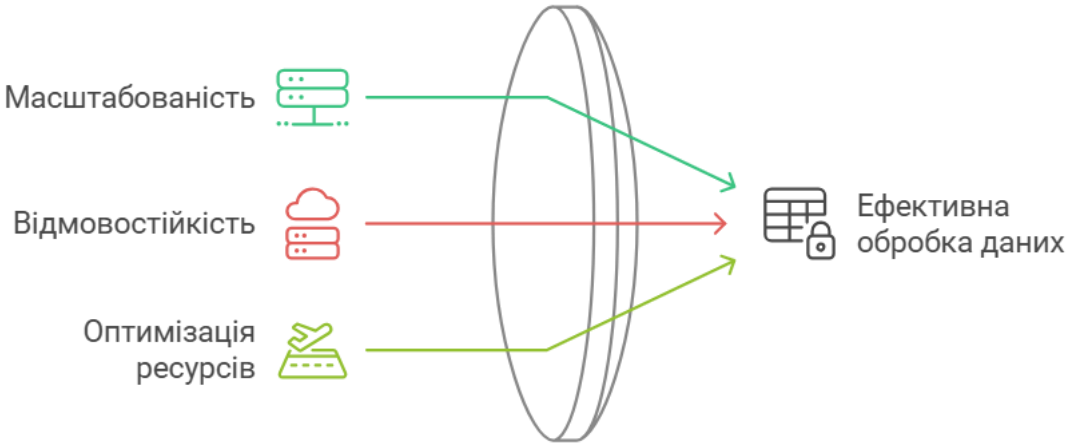

Рис. 2.5. Об'єднані переваги обробки даних

Окремо було оглянуто методи оптимізації, такі як індекси та кешування, для прискорення доступу до даних і зменшення навантаження на систему (рис 2.6). Ці методи особливо корисні для зменшення часу обробки при роботі з великими об'ємами часто використовуваних даних.

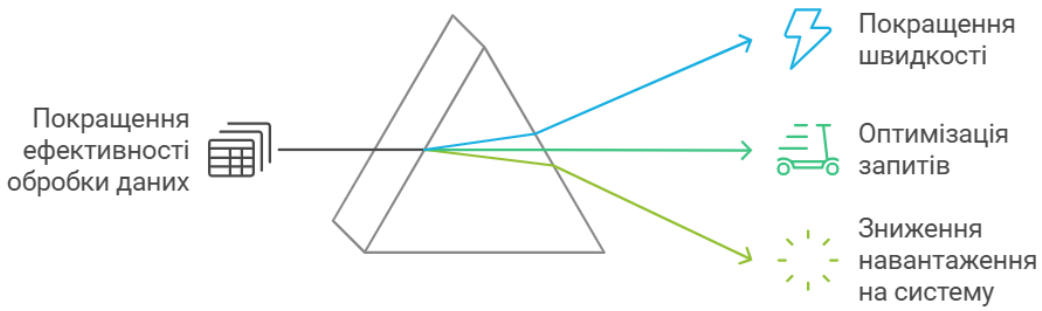

Рис. 2.6. Покращення обробки даних за допомгою індексів та кешування

У аналізі різних досліджень використання індексів дозволило значно зменшити час вибірки даних з великих таблиць. Наприклад, при роботі з даними про покупки клієнтів, індекси на колонках з ID клієнта дозволяють швидше знаходити відповідні записи [25]. Також було проведено перевірено використання кешування через Redis, що дозволяє зберігати результати часто виконуваних запитів або обчислень, тим самим зменшуючи навантаження на систему.

Переваги використання індексів та кешування:

– кешування дозволяє швидко отримувати часто використовувані дані без необхідності повторної обробки;

– використання індекси значно знижують час вибірки даних з великих таблиць;

– завдяки кешушуванню система може зменшити частоту запитів до бази даних, що знижує навантаження на інфраструктуру.

2.5. Інструменти та технології для автоматизації ETL

Реалізація ETL-системи для обробки великих обсягів даних вимагає використання надійних, масштабованих і гнучких інструментів. Мета такого підходу — забезпечити ефективний збір, трансформацію і завантаження даних із різних джерел у централізоване сховище. Для цієї мети були обрані такі основні технології: PostgreSQL**,** Apache Airflow, а також бібліотеки Python для роботи з API, обробки даних і їх аналізу. Кожна з цих технологій має свої сильні сторони та надає потрібні можливості для створення ефективної ETL-системи.

PostgreSQL — потужна об’єктно-реляційна база даних з відкритим кодом, що гарантує стабільність, можливість масштабування та ефективну продуктивність. Вона підтримує різноманітні типи даних, включно з JSON, і має вбудовані функції аналітики, що дає змогу ефективно зберігати та обробляти значні обсяги структурованих і напівструктурованих даних. PostgreSQL використовується як основне сховище для трансформованих даних і результатів аналітики.

Apache Airflow — це інструмент для оркестрації ETL-процесів, що дозволяє автоматизувати і контролювати робочі процеси. Завдяки можливості створення графів завдань на Python, система є гнучкою та адаптивною. Airflow забезпечує зручний моніторинг, журналювання й інтеграцію з кластерними середовищами, що дозволяє легко працювати з великими обсягами даних. Основні ролі — це планування завантажень даних, координація ETL-процесів і їхній контроль.

PostgreSQL було обрано як основне сховище для конвертованих даних завдяки її надійності та здатності обробляти великі обсяги даних. Ця система управління базами даних пропонує високу стабільність, масштабованість і ефективність обробки даних, включаючи можливість зберігання структурованих і напівструктурованих даних. Завдяки вбудованим функціям, таким як підтримка JSON і функції аналізу, PostgreSQL може виконувати складні запити і аналізи, а також добре підходить для зберігання результатів. Це дозволяє оптимізувати обробку даних і зберігати результати в зручному форматі для подальшого аналізу.

Apache Airflow - провідний інструмент для автоматизації та оркестрування ETL-процесів, що дозволяє керувати і контролювати складні робочі процеси. Можливість побудови діаграм завдань в Python дозволяє Airflow створити гнучку і легко масштабовану систему для автоматизації обробки великих обсягів даних. Завдяки інтеграції з іншими сервісами та функціям моніторингу і логування, інструмент не тільки автоматизує процеси, а й забезпечує надійний і комфортний контроль над виконанням завдань. Це особливо важливо для підтримки продуктивності та ефективності при роботі з великими обсягами даних.

Python з бібліотеками, такими як Requests, Pandas та Psycopg2, слугує універсальним інструментом для збору, трансформації та завантаження даних. Requests використовується для отримання даних із API, Pandas — для їхньої обробки у табличному форматі, а Psycopg2 забезпечує підключення до PostgreSQL та виконання запитів. Ці інструменти створюють ефективний робочий процес для обробки даних, їхньої візуалізації та підготовки до аналітики.

Використання PostgreSQL**,** Apache Airflow і бібліотек Python (зокрема, Psycopg2) забезпечує повний цикл обробки даних: від збору до зберігання й аналітики. Завдяки гнучкості цих інструментів, система легко масштабується та адаптується до потреб великого бізнесу.

У даному розділі було розглянуто ключові аспекти автоматизації ETL-процесів, сучасні підходи до їх реалізації та вибір інструментів. Особлива увага приділяється методам вилучення даних із зовнішніх джерел, способам їх трансформації відповідно до заданих вимог і можливостям збереження у цільових сховищах. Це створює основу для побудови гнучких і масштабованих рішень, які відповідають потребам сучасних інформаційних систем.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ РОЗРОБЛЕНОГО РІШЕННЯ

3.1. Розробка алгоритмів та програмного забезпечення для ETL-процесів

На основі результатів досліджень та аналізу, проведених у другому розділі, я розроблино ETL-процес, який забезпечує ефективне збирання, трансформацію та завантаження даних про COVID-19 з відкритої Api Kaggle до локальної бази даних. Це забезпечує зручний доступ до необхідної інформації, а також забезпечує масштабованість і гнучкість системи для обробки великих обсягів даних. У цьому розділі детально описано основні компоненти та методи, що були використані при реалізації даного ETL-процесу.

На діаграмі (рис. 3.1) представлено ключові елементи системи для формування запиту до даних Kaggle та обробки даних за допомогою ETL-процесу.

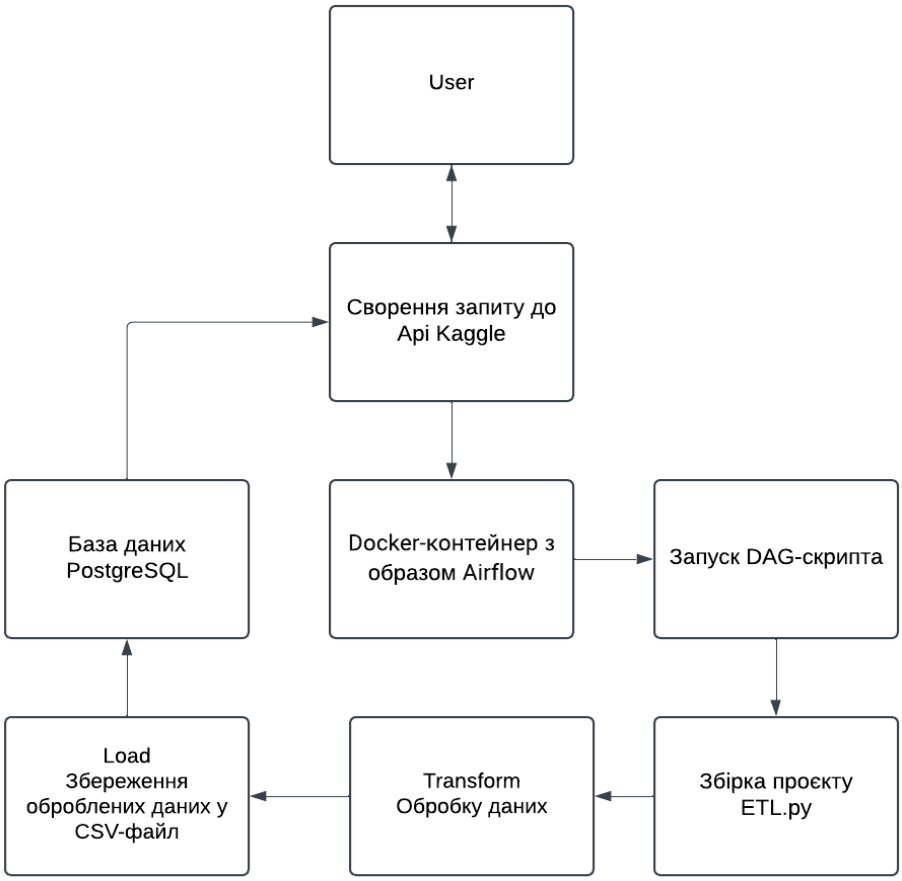

Рис. 3.1. Діаграма процесу створення запиту через FastAPI

Діаграма демонструє етапи обробки даних, починаючи з отримання запиту користувача до збереження оброблених даних у форматі CSV та їхнього збереження в базі даних PostgreSQL.

FastAPI виступає основним інструментом для обробки запитів користувачів. Це сучасний фреймворк для створення API, що забезпечує високу швидкість роботи та інтеграцію з іншими компонентами. Користувач надсилає запит через FastAPI, ініціюючи виконання ETL-процесу.

Docker-контейнер для забезпечення ізольованого середовища розробки та виконання всі компоненти ETL-процесу розгортаються в контейнері Docker. Використання контейнеризації для цієї роботи гарантує стабільність роботи системи незалежно від середовища, а також спрощує розгортання процесу на різних платформах.

Apache Airflow використовується для оркестрації усіх вищезгаданих ETL-процесу, що дозволяє автоматизувати виконання задач та відстежувати їхній статус. Така інтеграція забезпечує модульність системи, спрощує її масштабування та моніторинг.

Скрипт ETL.py реалізує три основні етапи ETL-процесу:

– Extract (вилучення даних) – передбачає виконунання HTTP-запиту до Kaggle API для отримання сирих та неструкторованих даних про COVID-19;

– Transform (трансформація даних) – цей етап проводить обробку та очищення даних, видалення непотрібної інформації та формування набору даних, готового до завантаження;

– Load (завантаження даних) – описує результати трансформації зберігаються у вигляді CSV-файлу, який використовується як проміжне сховище перед завантаженням даних у PostgreSQL.

Отже, діаграма відображає весь процес обробки запиту через FastAPI, трансформацію даних та їхнє збереження в базі даних, забезпечуючи ефективну автоматизацію обробки інформації

3.2. Створення модуля трансформації даних: перевірка, очищення та підготовка до аналізу

Клас KaggleFetcher було розроблено для збору даних із відкритого API Kaggle. Він ініціалізує HTTP-запит методом GET до відкритої бази даних COVID-19, яка була створена Білим домом у співпраці з провідними дослідницькими групами (рис. 3.2). Для забезпечення коректного виконання запитів використовується бібліотека Requests, яка дозволяє легко обробляти HTTP-запити, включно з обробкою помилок, встановленням тайм-аутів і підтримкою аутентифікації.

```
if __name__ == "__main__":
    base_url = "https://www.kaggle.com/api/v1/datasets"
    # Ваш токен для API Kaggle
    api_token = "1234567890abcdef1234567890abcdef"

    # Ініціалізація класу KaggleFetcher
    kaggle_fetcher = KaggleFetcher(base_url, api_token)

    # Виклик методу fetch_data для отримання даних
    dataset_path = "whitehouse/covid-19-dataset"
    data = kaggle_fetcher.fetch_data(dataset_path)
```

Рис. 3.2. Схема підключення до відповідно API

У процесі розробки алгоритму я реалізував метод fetch_data, який виконує запит для отримання вмісту датасету з необхідними даними (рис. 3.3). Метод враховує можливі помилки під час виконання запитів і повертає None, якщо виникає проблема. Такий підхід дозволяє ефективно обробляти непередбачені ситуації, зберігаючи стабільність роботи системи.

```python
import requests

class KaggleFetcher:
    def __init__(self, base_url, api_token):
        self.base_url = base_url
        self.headers = {
            'Authorization': f'Bearer {api_token}'
        }

    def fetch_data(self, dataset_path, timeout=10):
        url = f"{self.base_url}/{dataset_path}"
        try:
            response = requests.get(url, headers=self.headers, timeout=timeout)
            response.raise_for_status()  # Перевірка на помилки HTTP
            return response.json()
```

Рис. 3.3. Схема підключення за допомогою реквеста

Функція transform_covid_data виконує аналіз та обробку отриманого вмісту (рис. 3.4). У цій функції реалізовано перевірку наявності даних та їх подальший парсинг через Pandas та BeautifulSoup. Ключова інформація про наукові статті (наприклад, назва, автори, дата публікації, ключові слова) витягується з JSON-об'єкта, розташованого в скрипті з ідентифікатором __NEXT_DATA__. Отримані дані зберігаються у форматі словників у списку data_article, який перетворюється в DataFrame.

```python
import pandas as pd

def transform_covid_csv_data(directory_path):
    data_frames = []

    for file_name in os.listdir(directory_path):
        if file_name.endswith('.csv'):
            file_path = os.path.join(directory_path, file_name)
            try:
                df = pd.read_csv(file_path)

                if 'title' in df.columns and 'authors' in df.columns:
                    df['Publication Date'] = pd.to_datetime(df['published_date'], errors='coerce')
                    df['Authors'] = df['authors'].apply(lambda x: x.split(';') if pd.notna(x) else [])
                    df['Keywords'] = df.get('keywords', '').apply(lambda x: x.split(';') if pd.notna(x) else [])

                    df = df[['title', 'Authors', 'Publication Date', 'Keywords']].rename(columns={
                        'title': 'Title'
                    })

                    data_frames.append(df)
```

Рис. 3.4. Трансформація даних

У JSON-даних вибираються лише потрібні поля: назва статті, дата публікації, автори, ключові слова, посилання на оригінальне джерело. Витягнуті дані зберігаються у списку словників `data_covid`, після чого перетворюються в `DataFrame` для подальшого завантаження.

Завдяки тестуванню та аналізу впроваджено функцію insert_data_into_db, яка відповідає за завантаження даних у базу даних. Цей підхід забезпечує швидкий та надійний запис великих обсягів даних у сховище, що дозволяє гарантувати масштабованість системи.

Головна функція main ініціалізує необхідні об'єкти та викликає весь ETL-процес (рис. 3.5). Це дозволяє легко контролювати виконання збирання та обробки даних і забезпечує необхідну гнучкість для подальших змін у системі.

```
from fetch_data import IMDBFetcher
from transform_data import transform_imdb_data

def main():
    fetcher = KaggleFetcher()
    html_content = fetcher.fetch_data()
    transform_imdb_data(html_content)

if __name__ == "__main__":
    main()
```

Рис. 3.5. Ініціалізація об'єкта та виклик процесу ETL

На основі тестувань і практичного застосування цього алгоритму можна переконатися, що розроблений ETL-процес є ефективним інструментом для обробки даних з зовнішніх джерел, таких як API Kaggle, що дозволяє досягти високої продуктивності та надійності системи

3.3. Підключення до PostgreSQL для зберігання оброблених даних

Алгоритм програми для виконання ETL-процесу, зображений на рис. 3.6, реалізує кілька основних етапів обробки даних, починаючи від ініціалізації контейнера до завантаження даних у базу даних PostgreSQL.

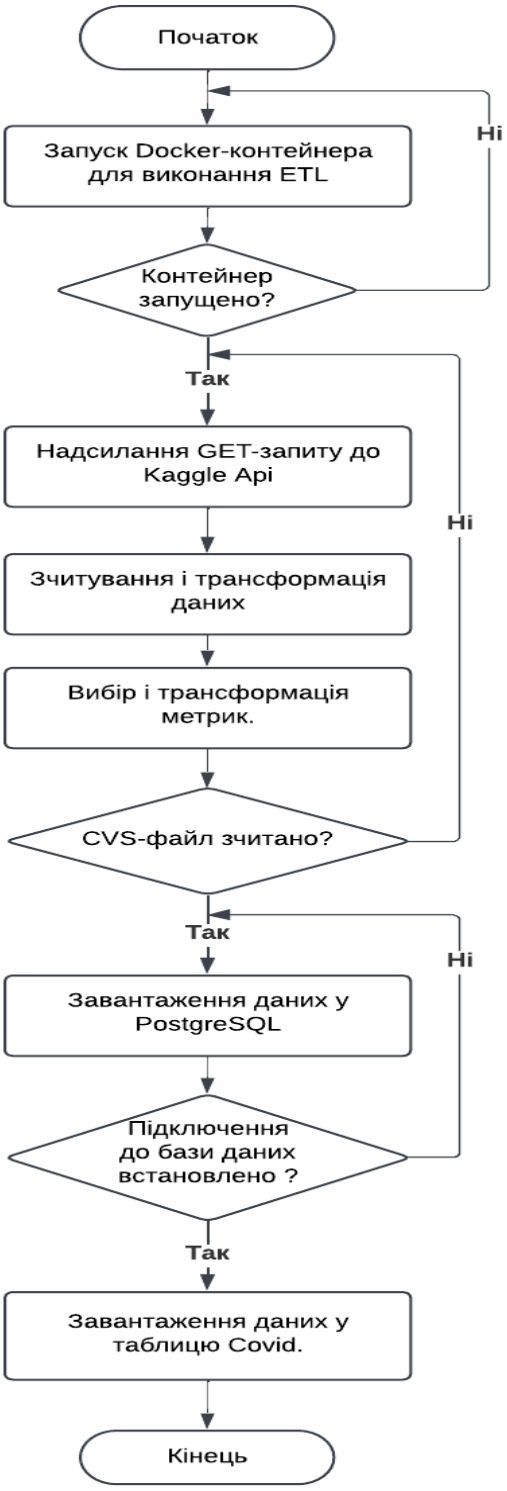

Рис. 3.6. Блок-схема алгоритму ETL-системи

На початковому етапі запускається Docker-контейнер, який забезпечує стабільне середовище для виконання ETL-процесу. У контейнері додатково інтегровано Apache Airflow для оркестрації задач ETL-процесу, що дозволяє автоматизувати роботу та забезпечувати моніторинг і керування процесами.

Далі здійснюється отримання даних з платформи Kaggle через GET-запит до відкритого API. Отримані дані зберігаються у файлі в початковому сирому форматі для подальшої обробки.

Після цього відбувається етап трансформації даних. Сирі дані очищуються, парсяться та обробляються за допомогою спеціалізованих бібліотек Python. Вибираються ключові метрики (наприклад, назва набору даних, дата оновлення, кількість записів тощо), які формуються у структурованому форматі CSV.

На завершальному етапі трансформовані дані завантажуються у реляційну базу даних PostgreSQL. Процес включає читання CSV-файлу, встановлення з'єднання з базою даних та запис інформації в таблицю datasets, що була попередньо створена для зберігання цієї інформації (рис. 3.7.).

```
Data columns (total 20 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   cord_uid          764394 non-null  object
 1   sha               325959 non-null  object
 2   source_x          764394 non-null  object
 3   title             764394 non-null  object
 4   doi               533220 non-null  object
 5   pmcid             311636 non-null  object
 6   pubmed_id         403269 non-null  object
 7   license           764394 non-null  object
 8   abstract          764394 non-null  object
 9   publish_time      414530 non-null  datetime64[ns]
 10  authors           759718 non-null  object
 11  journal           688086 non-null  object
 12  mag_id            0 non-null       float64
 13  who_covidence_id  303510 non-null  object
 14  arxiv_id          14242 non-null   object
 15  pdf_json_files    325959 non-null  object
 16  pmc_json_files    270778 non-null  object
 17  url               559092 non-null  object
 18  s2_id             708578 non-null  float64
 19  publication_year  414530 non-null  float64
dtypes: datetime64[ns](1), float64(3), object(16)
```

Рис. 3.7. Дані з платформи Kaggle у таблиці PostgreSQL

До основних особливостей розробленої платформи можна віднести:

1) Інтерфейс для розробника, який дозволяє швидко виконувати операції з даними та відстежувати статус обробки в реальному часі.

2) Автоматизація та масштабованість релізовано завдяки використанню Docker і Apache Airflow, платформа підтримує автоматичний запуск процесів та їхнє масштабування залежно від обсягів даних.

3) Розроблена система ефективно працює з великими обсягами інформації, використовуючи паралельні процеси для підвищення продуктивності.

На основі отриманих досліджень, виконаних у другому розділі, у третьому розділі описано і продемонстровано платформу для обробки даних, яка забезпечує швидку, масштабовану та надійну обробку великих обсягів інформації. Наведені теоретичні напрацювання було реалізовано у вигляді комплексного рішення, що інтегрує сучасні методи автоматизації, безпеки та обробки даних.

Дана реалізація забезпечує повну автоматизацію ETL-процесу, починаючи від отримання даних з Kaggle до їхнього збереження у PostgreSQL (рис. 3.8.). Це забезпечує ефективну обробку та аналіз великих обсягів даних, підтримуючи стабільну роботу та зручність використання системи.

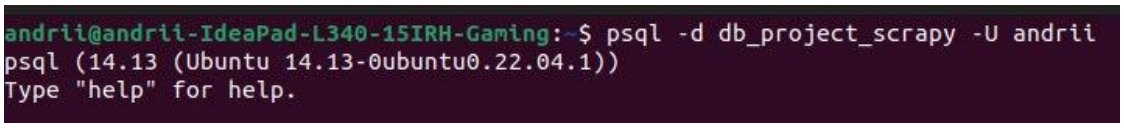

Рис. 3.8. Підключення до бази даних PostgreSQL

3.4. Використання Apache Airflow для оркестрації ETL-процесів

Для спрощення налаштування та управління середовищем я використовую Docker для запуску Airflow. Docker Compose дозволяє створити ізольоване середовище, що включає всі необхідні сервіси, такі як вебінтерфейс Airflow, планувальник (scheduler) та виконавець (worker). Docker надає можливість легко масштабувати роботу та інтегрувати додаткові сервіси

Airflow координує роботу кількох етапів:

1) Етап завантаження даних — модуль KaggleFetcher ініціює завантаження даних із джерела.

2) Етап трансформації — використовується функція transform_imdb_data, яка зчитує датасет, оброляє дані та готує їх до аналітики та формує структуровані дані.

3) Етап завантаження в PostgreSQL — функція insert_data_into_db записує дані у відповідну таблицю в базі даних.

Для кожного з цих етапів у Airflow створюється окреме завдання, об’єднане у Directed Acyclic Graph (DAG).

```python
from fetch_data import KaggleFetcher
from transform_data import transform_imdb_data
from db_operations import insert_data_into_db

default_args = {
    'owner': 'andrii',
    'depends_on_past': False,
    'start_date': datetime(2024, 12, 15),
    'retries': 1,
}

def fetch_data():
    fetcher = KaggleFetcher()
    return fetcher.fetch_data()

def transform_data(**context):
    html_content = context['ti'].xcom_pull(task_ids='fetch_data'
    transform_imdb_data(html_content)

with DAG(
    'etl_process',
    default_args=default_args,
    schedule_interval='@daily',
```

Рис. 3.9. Приклад DAG для нашого ETL-процесу.

Кожен із цих кроків представлений у вигляді завдань (tasks) у DAG-файлі Airflow. Взаємодія між сервісами налаштована через Airflow Operators.

На основі цього налаштування я досяг ефективної обробки даних Kaggle з подальшим завантаженням у PostgreSQL. Вебінтерфейс Airflow показує успішне виконання DAG, що включає всі кроки: завантаження, трансформацію та збереження даних у базу

У цьому розділі описується як використовується інтеграція Apache Airflow у Docker-контейнері та використання його для координації всіх етапів обробки даних: завантаження, трансформації та збереження в базі даних PostgreSQL

3.5. Аналіз та тестуання ефективності та масштабованості системи

Для дослідження було розроблено прототип ETL-системи, яка автоматизує процес збору, обробки та збереження великих обсягів даних з різних джерел. Основні технології, що використовувалися в системі, включають Apache Airflow для оркестрації робочих процесів, Docker для контейнеризації всіх компонентів, та PostgreSQL для зберігання результатів. Для доступу до оброблених даних було розроблено REST API на основі Flask.

Система пройшла серію тестувань з використанням реальних наборів даних, що дозволило оцінити її продуктивність, гнучкість і масштабованість. Під час тестування основна увага приділялася продуктивності на великих наборах даних і стабільності роботи в контейнеризованому середовищі.

Для перевірки працездатності ETL-системи було проведено тестування на реальних наборах даних (рис. 3.10.). Система отримувала вхідні дані з веб-сторінок, зокрема інформацію про фільми, яку потрібно було перетворити у відповідний формат і завантажити в базу даних.

```
dfe = rnr.groupby('location')[['continent', 'total_cases', 'total_cases_per_million', 'total_deat
hs', 'total_deaths_per_million', 'people_vaccinated', 'people_fully_vaccinated', 'total_booster
s', 'people_vaccinated_per_hundred', 'people_fully_vaccinated_per_hundred', 'total_boosters_per_h
undred', 'population', 'population_density', 'median_age', 'aged_65_older', 'aged_70_older', 'gdp
_per_capita', 'life_expectancy', 'human_development_index']].max()

# Reset index for easier manipulation
sd = dfe.reset_index()

# Remove rows with unwanted indices
sm = sd.drop(index=[1, 12, 70, 71, 96, 127, 128, 162, 169, 211, 241, 251])

# Reset index and sort values for further analysis
sn = sm.reset_index().sort_values(['continent', 'location']).reset_index()

# Extract and plot Top 20 Countries by Total Cases
top20_ca = sn.sort_values('total_cases', ascending=False).head(20)
top20_cases = top20_ca[['location', 'total_cases']].reset_index().drop('index', axis=1)
```

Рис. 3.10. Фрагмент коду для обробки даних з Kaggle Api.

Далі отримані дані оброблялися і завантажувалися до PostgreSQL (рис. 3.11.). Для перевірки правильності обробки використовувався API-запит, який повертав топ-250 публікафій авторів про COVID-19 із бази даних.

```
curl -X GET "http://localhost:5000/covid19/top250"
```

Рис. 3.11. API-запит для отримання результатів обробки даних через Flask API.

Технології для процесу проектування архітектури системи. Вона включає такі компоненти:

1) ETL-система – використовується для обробки великих наборів даних у розподіленому середовищі, що дозволяє значно скоротити час обробки;

2) Docker – забезпечує контейнеризацію всіх компонентів, що полегшує процес налаштування та розгортання системи;

3) PostgreSQL – реляційна база даних, в якій зберігаються результати обробки даних, забезпечуючи надійне і ефективне зберігання великих обсягів інформації.

Усі компоненти було розгорнуто за допомогою Docker для забезпечення гнучкого керування та ізоляції сервісів (рис. 3.12), що дозволяє легко адаптувати систему до різних обсягів даних та вимог продуктивності.

```
# Використовуємо базовий образ Python
FROM python:3.9-slim

# Встановлюємо робочу директорію в контейнері
WORKDIR /app

# Копіюємо requirements.txt в контейнер
COPY requirements.txt .

# Встановлюємо всі залежності з requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Копіюємо весь вміст локальної директорії в контейнер
COPY . /app

# Виставляємо змінні середовища (якщо необхідно)
ENV DB_HOST=postgres_host
ENV DB_NAME=database_name
ENV DB_USER=database_user
ENV DB_PASSWORD=database_password

# Команда для запуску ETL-скрипта
CMD ["python", "etl_script.py"]
```

Рис. 3.12. Фрагмент docker-compose файлу для розгортання основних компонентів системи.

Результатом тестування став FastAPI-запит, який повертає підготовлені метрики про ковід із бази даних, що свідчить про успішність реалізації ETL-процесу

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1. Охорона праці

Темою моєї магістерської роботи є розробка методів та інструментів для побудови гнучких та масштабованих ETL-систем, які ефективно обробляють великі обсяги даних. Очікується, що процес виконання цієї роботи буде пов'язаний з інтенсивним використанням комп'ютерного обладнання та спеціалізованого програмного забезпечення. В таких умовах особлива увага повинна приділятися дотриманню вимог охорони праці та техніки безпеки при використанні комп'ютерної техніки та засобів автоматизації; з метою забезпечення безпечних та комфортних умов праці розробників та спеціалістів, які підтримують функціонування системи ETL, необхідно дотримуватися чинних норм охорони праці та техніки безпеки.

Робочі місця, де здійснюється проектування, тестування та експлуатація ETL-систем, повинні відповідати вимогам НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями».

Згідно з НПАОП 0.00-7.15-18, приміщення для роботи фахівців мають бути обладнані автоматичними системами пожежної сигналізації, які відповідають:

- нормативним вимогам до встановлення пожежогасіння та сигналізації;
- державним будівельним нормам «Інженерне обладнання будинків і споруд».

Крім того, такі приміщення слід оснастити вогнегасниками, кількість і тип яких визначаються чинними нормами. У серверних кімнатах та центрах обробки даних доцільно застосовувати теплові пожежні сповіщувачі й спеціалізовані системи пожежогасіння згідно з вимогами ДБН В.1.1-7-2016 та НАПБ А.01.001-2014.

Електромережі, що забезпечують живлення обладнання для ETL-систем, повинні бути організовані у вигляді трипровідних мереж із обов’язковим заземленням усіх пристроїв. Використання одного провідника для нульового робочого і нульового захисного проводів категорично заборонено. Розетки й штепсельні з’єднання повинні

відповідати сучасним стандартам безпеки, забезпечуючи захист від перенапруг і пов’язаних із ними ризиків.

У приміщеннях із понад п’ятьма комп’ютерами обов’язковим є встановлення аварійного вимикача для повного знеструмлення обладнання, за винятком систем освітлення.

Не менш важливо забезпечити ергономічність робочих місць, що передбачає:

- відповідність меблів і обладнання ергономічним стандартам;
- дотримання оптимальної відстані між монітором і очима користувача;
- правильне розташування периферійних пристроїв у зоні легкої досяжності.

Освітлення робочих приміщень повинно відповідати вимогам ДБН В.2.5-28:2018 «Природне і штучне освітлення», забезпечуючи оптимальний рівень освітленості для тривалої роботи з екранними пристроями.

Психологічний клімат у колективі також є важливим аспектом. Забезпечення підтримки, відкритої комунікації та профілактики стресів сприяє підвищенню продуктивності працівників і зниженню ризиків професійного вигорання.

Дотримання сучасних вимог з охорони праці, електробезпеки, ергономіки та забезпечення належного психологічного клімату є важливим фактором для створення сприятливих умов роботи. Це гарантує не тільки безпеку та здоров'я працівників, але й підвищує ефективність виконання проекту, зокрема розробки та експлуатації ETL-систем для обробки великих обсягів даних.

4.2. Безпека в надзвичайних ситуаціях

4.2.1. Середовище проживання людини: навколишнє, виробниче, побутове. Проблема безпека людини і завдання керівного складу і її забезпечені

Середовище проживання людини поділяється на три основні сфери: навколишнє, виробниче та побутове. Кожна з них має значний вплив на здоров'я, добробут і безпеку людини. Забезпечення безпеки у цих сферах є комплексним

завданням, що потребує уваги як на рівні державної політики, так і від керівного складу підприємств та організацій.

Навколишнє середовище є основним джерелом ресурсів, необхідних для існування людини. Водночас воно може бути джерелом загроз через забруднення атмосфери, ґрунтів і води, радіаційний вплив, підвищений рівень шуму тощо. Негативні антропогенні зміни у природному середовищі спричиняють серйозні проблеми, зокрема захворювання органів дихання, онкологічні та алергічні захворювання.

Завдання керівників на цьому рівні полягає у впровадженні екологічно чистих технологій, моніторингу екологічної ситуації та дотриманні природоохоронного законодавства. Крім того, велике значення має екологічна освіта та підвищення обізнаності населенн[30].

Виробниче середовище охоплює умови праці на підприємствах та в організаціях. Його безпека визначається факторами, такими як стан виробничого обладнання, дотримання норм охорони праці, рівень освітлення, мікроклімат у приміщеннях, відсутність токсичних і шкідливих речовин. Недотримання цих умов може спричинити професійні захворювання, травми та нещасні випадки.

Керівний склад організації несе відповідальність за створення безпечних умов праці, проведення інструктажів з техніки безпеки, забезпечення працівників засобами індивідуального захисту та організацію регулярного технічного обслуговування обладнання.

Побутове середовище включає житлові приміщення, умови в будинках і місцях проживання людини. Безпечне побутове середовище визначається такими чинниками, як якість питної води, належна вентиляція, наявність систем пожежної сигналізації, правильне електричне обладнання тощо.

Вирішення завдань безпеки у побутовій сфері передбачає підвищення стандартів житлових умов, контроль за якістю будівельних матеріалів, регулярне проведення профілактичних заходів для запобігання аваріям, зокрема витокам газу чи займанням електромереж [31].

Головна проблема безпеки людини полягає у зменшенні негативного впливу зовнішніх і внутрішніх чинників на її здоров’я та життя. В умовах сучасного світу, де кількість техногенних ризиків постійно зростає, забезпечення безпеки стає стратегічним завданням як для держави, так і для керівників підприємств.

Завдання керівного складу

1. Розробка та впровадження політик безпеки: Керівники мають забезпечувати впровадження систем управління ризиками у всіх сферах діяльності.
2. Організація навчання персоналу. Систематичне підвищення кваліфікації працівників у сфері безпеки є необхідним для запобігання небезпечним ситуаціям.
3. Забезпечення належного фінансування. Інвестиції у безпеку праці, екологічні проєкти та поліпшення побутових умов – ключовий аспект відповідальності керівників.
4. Контроль і аудит. Постійний моніторинг дотримання правил і нормативів безпеки у всіх сферах діяльності.

Таким чином, ефективна взаємодія державних органів, керівництва підприємств та громадян дозволить створити сприятливі умови для збереження життя, здоров’я та добробуту людей у сучасному суспільстві.

ВИСНОВКИ

У межах цієї кваліфікаційної роботи було досягнуто таких наукових і практичних результатів:

1. Проведено огляд існуючих технологій та сервісів у сфері видобутку даних, що дало змогу визначити ключові вимоги до систем обробки даних і обґрунтувати вибір сучасних інструментів для автоматизації ETL-процесів.

2. Розроблено архітектуру ETL-системи, яка базується на використанні Docker, FastAPI та PostgreSQL, що забезпечує автоматичний збір даних із зовнішніх API, їх трансформацію у необхідний формат та збереження в базі даних, гарантуючи масштабованість і гнучкість системи.

3. Налаштовано та створено DAG-файл в Apache Airflow для оркестрації всіх ETL-процесів, що дало змогу автоматизувати процеси завантаження, трансформації та збереження даних, оптимізувавши їх виконання.

4. Реалізовано спеціалізовану та автоматизовану систему очищення даних з відкритої API Kaggle, що дозволило забезпечити попередню фільтрацію та очищення даних для подальшого аналізу.

5. Налаштовано базу даних PostgreSQL для локального зберігання оброблених даних, що дало змогу гарантувати їх надійне збереження та доступність для подальшого використання.

6. Створено Docker-контейнер для забезпечення ізольованого та переносного середовища виконання ETL-процесів, що підвищило їх надійність і спростило розгортання системи на різних середовищах.

7. Розроблено процеси трансформації даних, що дозволяють здійснювати ефективну обробку великих обсягів інформації з використанням сучасних ETL-інструментів, гарантуючи відповідність даних встановленим вимогам.

8. Підтверджено доцільність застосування сучасних технологій контейнеризації та оркестрації для автоматизації ETL-процесів, що дало змогу створити гнучку та масштабовану систему для роботи з великими обсягами даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Salqvist P. A comparative study of the Data Warehouse and Data Lakehouse architecture : Degree Project in the Field of Technology Information Technology and the Main Field of Study Computer Science and Engineering. Stockholm, 2023. 147 с.

2. Realdy C. ETL Processes: Evolution and Trends. Cambridge University Press, 2020. С. 256-273.

3. Karau H. Learning Spark: Lightning-Fast Big Data Analysis. O'Reilly Media, 2015. С. 63-96.

4. Integrating Airflow & Databricks for ETL | Databricks Blog, 2023. URL: https://databricks.com/blog/2023/02/integrating-airflow-databricks-for-etl (дата звернення 23.11.2024).

5. Apache Airflow Documentation. URL: https://airflow.apache.org/docs/ (дата звернення 26.11.2024).

6. Spark Documentation. URL: https://spark.apache.org/docs/latest/ (дата звернення 28.11.2024).

7. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення 29.11.2024).

8. Docker Documentation. URL: https://docs.docker.com/ (дата звернення 30.10.2024).

9. Heinrich T. Using Apache Airflow for ETL Pipelines. Data Science Journal, 2022. С. 103-118.

10. Frank H., Smith K. PostgreSQL for Data Analytics. O'Reilly Media, 2021.

11. Wang X., Zhou J. Implementing ETL Pipelines with Docker. Journal of Cloud Computing, 2021. С. 45-5

12. Thabet N., Soomro T. Big Data Challenges. Journal of Computer Engineering & Information Technology. 2015. Т. 4, № 3. С. 1–2. 2.

13. Simitsis A., Skiadopoulos S., Vassiliadis P. The History, Present, and Future of ETL Technology. Test-of-Time Award - Invited Talk. 2023. С. 3–12.

14. Gorhe S. ETL in Near-real-time Environment: A Review of Challenges and

Possible Solutions : Research. Auckland, 2020.

15. Zode M. The Evolution of ETL -From Hand-coded ETL to Tool-based ETL. Cognizant Technology Solutions. С. 2–5.

16. What is container orchestration?. Red Hat. URL: https://www.redhat.com/en/topics/containers/what-is-container-orchestration. (дата звернення: 15.12.2024). 82

17. What is container orchestration?. VMWare. URL: https://www.vmware.com/topics/glossary/content/container-orchestration.html (дата звернення: 15.12.2024).

18. Vauban V. Orchestrator Comparison: Docker Swarm vs Kubernetes vs Apache Mesos. Linkedin. URL: https://www.linkedin.com/pulse/orchestrator-comparison-docker-swarm-vskubernetes-apache-vauban/ (дата звернення: 15.12.2024).

19. Nickoloff J. Evaluating Container Platforms at Scale. Medium. URL: https://medium.com/on-docker/evaluating-container-platforms-atscale-5e7b44d93f2c#.pbxx2w5ai (дата звернення: 15.12.2024).

20. Kronis. Docker Swarm over Kubernetes. Kronis Blog. URL: https://blog.kronis.dev/articles/docker-swarm-over-kubernetes (дата звернення: 16.12.2024).

21. Deploy to Swarm Swarm.docs. URL: https://docs.docker.com/get-started/swarm-deploy/ (дата звернення: 16.12.2024).

22. Raft consensus in swarm mode. Swarm.docs. URL: https://docs.docker.com/engine/swarm/raft/ (дата звернення: 16.12.2024).

23. Swarm mode key concepts. Swarm.docs. URL: https://docs.docker.com/engine/swarm/key-concepts/ (дата звернення: 16.12.2024).

24. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Програма та методичні рекомендації з проходження практики за тематикою кваліфікаційної роботи для студентів спеціальності 123 «Комп’ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль: ТНТУ. 2024. 45 с.

25. Луцик Н.С., Луцків А.М., Осухівська Г.М., Тиш Є.В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів

спеціальності 123 «Комп’ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. 2024. 44 с.

26. Варавін А.В., Лещишин Ю.З., Чайковський А.В. Методичні вказівки до виконання курсового проєкту з дисципліни «Дослідження і проєктування комп’ютерних систем та мереж» для здобувачів другого (магістерського) рівня вищої освіти спеціальності 123 «Комп’ютерна інженерія» усіх форм навчання. Тернопіль: ТНТУ, 2024. 32 с.

27. Луговий А.С., Тиш Є.В. Методи та засоби роботи ETL-процесів в умовах високих навантажень. Матеріали XII міжнародної науково-практичної конференції молодих учених та студентів «Актуальні задачі сучасних технологій» (11-12 грудня 2023 року). Тернопіль: ТНТУ. 2024. С. 420.

28. Луговий А.С., Тиш Є.В. Методи оптимізації продуктивності etl-систем у багаторівневих аналітичних платформах. Матеріали XI науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (18-19 грудня 2024 року). Тернопіль: ТНТУ. 2024. С. 162.

29. Жидецький В.Ц. Охорона праці користувачів комп´ютерів. Львів: Афіша, 2011. 176 с

30. Стручок, Володимир Сергійович. "Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання." Тернопіль: ТНТУ. 2022. С. 67.

31. Стручок, Володимир Сергійович. "Техноекологія та цивільна безпека. Частина «Цивільна безпека». Навчальний посібник." Тернопіль: ТНТУ. 2022. С. 42.

32. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп’ютерні мережі. Книга 1 [навчальний посібник]. Львів : «Магнолія 2006», 2013. 256 с.

33. Микитишин А. Г., Митник М. М., Стухляк П. Д., Пасічник В. В. Комп’ютерні мережі. Книга 2. [навчальний посібник]. Львів : "Магнолія 2006", 2014. 312 с.

34. Лупенко С. А., Пасічник В. В., Тиш Є. В. Комп’ютерна логіка. Львів: Видавництво «Магнолія - 2006». 2015. 354 с.

35. Sachin D. N. Distinguishing HDFS from Cloud Data Lakes: ADLS Gen2 and Amazon S3. Linkedin. URL: https://www.linkedin.com/pulse/distinguishing-hdfs-from-cloud-data-lakes-adlsgen2-amazon-d-n--7e2fc/ (дата звернення: 16.12.2024).

36. BryteFlow. How to choose between Parquet, ORC and AVRO for S3, Redshift and Snowflake?. BryteFlow. URL: https://bryteflow.com/how-to-choose-between-parquet-orc-and-avro/ (дата звернення: 16.12.2024).

Додаток А.

Тези конференцій

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Університет імені П'єра і Марії Кюрі (Франція)
Маріборський університет (Словенія)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Наукове товариство ім. Т.Шевченка

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник
тез доповідей

XIII Міжнародної науково-практичної конференції молодих учених та студентів
11-12 грудня 2024 року

УКРАЇНА
ТЕРНОПІЛЬ – 2024

Матеріали XIII Міжнародної науково-практичної конференції молодих учених та студентів
«АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ» – Тернопіль, 11-12 грудня 2024 року

УДК 681.855

А.С. Луговий, Є.В. Тиш, к.т.н.
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

МЕТОДИ ТА ЗАСОБИ РОБОТИ ETL-ПРОЦЕСІВ В УМОВАХ ВИСОКИХ НАВАНТАЖЕНЬ

A.S. Lugovyi, I.V. Tysh, Ph.D.

METHODS AND TOOLS FOR ETL PROCESSES UNDER HIGH LOAD CONDITIONS

У сучасних умовах, коли обсяги оброблюваних даних постійно зростають, забезпечення безперебійної роботи ETL-процесів (Extract, Transform, Load) стало важливою вимогою для аналітичних платформ. Ключовим завданням є досягнення максимальної ефективності в обробці даних, навіть при високих навантаженнях, що виникають через великий обсяг і складність інформації. Зі зростанням розмірів даних, що підлягають обробці, збільшується потреба у впровадженні інноваційних технологій та нових методів для забезпечення бездоганної продуктивності.

Існуючи виклики, як-от перевантаження систем, непередбачуване навантаження на сервери та значні затримки під час виконання складних трансформацій, потребують ефективних рішень. Трансформації даних, особливо складних запитів або обробки неструктурованої інформації, можуть створювати серйозні проблеми, знижуючи швидкість та ефективність процесів. Ці труднощі ставлять перед розробниками завдання з оптимізації процесів і управління ресурсами.

Для досягнення стабільної роботи ETL-процесів в умовах великих обсягів даних і високих навантажень необхідно використовувати передові методи та стратегії, такі як інтеграція хмарних платформ та автоматизованих систем моніторингу. Хмарні сервіси дозволяють масштабувати обчислювальні ресурси за потреби, що дозволяє підтримувати високий рівень продуктивності. Водночас автоматизовані системи моніторингу допомагають здійснювати постійний контроль за навантаженнями і своєчасно коригувати роботу системи.

Застосування інтелектуальних алгоритмів для балансування навантаження та управління ресурсами також сприяє покращенню продуктивності. Такі технології допомагають знизити ризики перевантаження, а також забезпечити рівномірний розподіл обчислювальних потужностей на всіх етапах обробки даних. Це дозволяє підвищити ефективність ETL-процесів навіть в екстремальних умовах.

У результаті, поєднання класичних і новітніх підходів в обробці даних забезпечує стабільну роботу аналітичних платформ, що дозволяє організаціям швидко адаптуватися до змінюваних умов та зберігати високу продуктивність навіть за високих навантажень.

Література

1. Suraj K. C., 2021. Optimizing ETL Pipelines under High-Pressure Data Conditions. Proceedings of the Data Engineering Conference. ACM, 2021. 142-149 с
2. Pavlyuk T. O., 2022. Robust ETL Solutions for Streaming Data in Extreme Workloads. Computer Science and Information Technologies, 2012. 102-113 с.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18-19 грудня 2024 року

ТЕРНОПІЛЬ
2024

Додаток Б.

Скрипт запуску ETL-системи

```
import logging
import json
import pandas as pd
from fetch_data import IMDBFetcher
from transform_data import transform_imdb_data
from db_operations import insert_data_into_db

# Configure logging
logging.basicConfig(level=logging.INFO,     format='%(asctime)s     -
    %(levelname)s - %(message)s')

def main():
    logging.info("Starting IMDB data extraction process.")
    fetcher = IMDBFetcher()
    html_content = fetcher.fetch_data()

    if html_content:
        logging.info("Fetched     data     successfully.     Starting
    transformation.")
        transformed_data = transform_imdb_data(html_content)

        if not transformed_data.empty:
            logging.info("Data  transformed  successfully.  Inserting
    into database.")
            insert_data_into_db(transformed_data)
            logging.info("Data inserted successfully.")
        else:
            logging.warning("Transformed data is empty. No data to
    insert.")
    else:
        logging.error("Failed to fetch data. Exiting process.")
```

```
if __name__ == "__main__":
    main()
import requests
import logging

class IMDBFetcher:
    def                                                    __init__(self,
    url='https://www.imdb.com/chart/top/?ref_=nv_mv_250'):
        self.url = url
        self.headers = {
            'User-Agent': (
                'Mozilla/5.0 (Windows NT 10.0; Win64; x64) '
                'AppleWebKit/537.36 (KHTML, like Gecko) '
                'Chrome/91.0.4472.124 Safari/537.36'
            )
        }

    def fetch_data(self):
        try:
            response = requests.get(self.url, headers=self.headers)
            response.raise_for_status()
            logging.info("Data fetched from IMDB successfully.")
            return response.text
        except requests.RequestException as e:
            logging.error(f"Error fetching data: {e}")
            return None
import json
import logging
import pandas as pd
from bs4 import BeautifulSoup

def transform_imdb_data(html_content):
    if html_content is None:
        logging.error("No HTML content to transform.")
        return pd.DataFrame()
```

```
soup = BeautifulSoup(html_content, 'html.parser')
movie_soup = soup.find('script', id='__NEXT_DATA__')

if not movie_soup:
    logging.error("No movie script data found in HTML content.")
    return pd.DataFrame()

try:
    movie_json = json.loads(movie_soup.get_text())
    movies                                                    =
movie_json['props']['pageProps']['pageData']['chartTitles']['edg
es']

    data_movie = []
    for movie in movies:
        node = movie['node']
        title = node['titleText']['text']
        year = node['releaseYear']['year']
        duration = node.get('runtime', {}).get('seconds', 0)
        rating = node.get('certificate', {}).get('rating', 'N/A')
        imdb_rating          =          node.get('ratingsSummary',
{}).get('aggregateRating', 0)
        vote_count           =          node.get('ratingsSummary',
{}).get('voteCount', 0)

        data_movie.append({
            "Title": title,
            "Year": year,
            "Duration": duration,
            "Rating": rating,
            "IMDb Rating": imdb_rating,
            "Vote Count": vote_count
        })

    df = pd.DataFrame(data_movie)
    logging.info("Data transformed successfully.")
```

```
        return df

    except (KeyError, TypeError, json.JSONDecodeError) as e:
        logging.error(f"Error transforming data: {e}")
        return pd.DataFrame()
import pandas as pd
import logging
from db_connection import get_connection

def insert_data_into_db(df: pd.DataFrame):
    if df.empty:
        logging.warning("DataFrame is empty. No data to insert.")
        return

    data_to_insert = df.to_records(index=False).tolist()

    insert_query = (
        "INSERT INTO movies (Title, Year, Duration, Rating,
    imdb_rating, vote_count) "
        "VALUES (%s, %s, %s, %s, %s, %s)"
    )

    conn = get_connection()

    try:
        cur = conn.cursor()
        cur.executemany(insert_query, data_to_insert)
        conn.commit()
        logging.info("Data inserted into database successfully.")
    except Exception as e:
        conn.rollback()
        logging.error(f"Error inserting data into database: {e}")
    finally:
        cur.close()
        conn.close()
import os
```

```
import psycopg2
from dotenv import load_dotenv

# Load environment variables from .env file
load_dotenv()

def get_connection():
    return psycopg2.connect(
        dbname=os.getenv("DB_NAME"),
        user=os.getenv("DB_USER"),
        password=os.getenv("DB_PASSWORD"),
        host=os.getenv("DB_HOST"),
        port=os.getenv("DB_PORT")
  )

#DAG файл
from airflow import DAG
from airflow.providers.postgres.hooks.postgres import PostgresHook
from airflow.providers.docker.operators.docker import DockerOperator
from airflow.operators.python import PythonOperator
from airflow.utils.dates import days_ago
from airflow.operators.dummy_operator import DummyOperator
import requests
import pandas as pd

def download_data_from_api():
    url = 'https://api.kaggle.com/datasets/download/<dataset_id>'
    response = requests.get(url)
    with open('/tmp/data.zip', 'wb') as f:
        f.write(response.content)
    return '/tmp/data.zip'

def clean_transform_data(file_path):
    # Приклад: розпакування та обробка CSV файлів
    import zipfile
    with zipfile.ZipFile(file_path, 'r') as zip_ref:
```

```
        zip_ref.extractall('/tmp/extracted_data')

    df = pd.read_csv('/tmp/extracted_data/data.csv')

    df_cleaned = df.dropna()  # Приклад очищення (видалення
    пропусків)

    hook = PostgresHook(postgres_conn_id='postgres_default')
    conn = hook.get_conn()
    df_cleaned.to_sql('processed_data', conn, if_exists='replace',
    index=False)

    return 'Data transformation complete'

# Опис DAG
default_args = {
    'owner': 'airflow',
    'retries': 3,
    'retry_delay': timedelta(minutes=5),
    'start_date': days_ago(1),
}

with DAG('ETL_Process_DAG',
         default_args=default_args,
         schedule_interval='@daily',  # Задаємо періодичність
    запуску DAG
         catchup=False) as dag:

    start_task = DummyOperator(task_id='start')

    download_task = PythonOperator(
        task_id='download_data',
        python_callable=download_data_from_api
    )

    transform_task = PythonOperator(
```

```
        task_id='transform_data',
        python_callable=clean_transform_data,
        op_args=['/tmp/data.zip']
    )

    docker_task = DockerOperator(
        task_id='docker_etl_task',
        image='my_etl_container',
        api_version='auto',
        auto_remove=True,
        command='python /etl_script.py',
        docker_url='unix://var/run/docker.sock',
        network_mode='bridge'
    )

    end_task = DummyOperator(task_id='end')

    start_task >> download_task >> transform_task >> docker_task >>
    end_task

#Dockerfile
FROM python:3.9-slim
WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . /app

ENV DB_HOST=postgres_host
ENV DB_NAME=database_name
ENV DB_USER=database_user
ENV DB_PASSWORD=database_password

CMD ["python", "etl_script.py"]
```