Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Системотехніки
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти другий (магістерський)

«Дослідження моделей та методів ETL і ELT трансформації, оркестрування для побудови корпоративних сховищ даних»

(тема)

Виконав:здобувач групи СПРМ-22-2

Щербак О.А.

(прізвище, ініціали)

Спеціальність 122 Комп'ютерні науки

(код і повна назва спеціальності)

Тип програми :освітньо-наукова

Освітня програма Системне проєктування

(повна назва освітньої програми)

Керівник доц. Хряпкін О.В

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ _____

(підпис) (прізвище, ініціали)

2024 р.

Я як студент ХНУРЕ розумію і підтримую політику закладу із академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки кваліфікаційної роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

10.06.2024 Щербак

(дата, підпис, прізвище студента/-ки)

Кваліфікаційна робота не містить відомостей заборонених до відкритого опублікування.

Кваліфікаційна робота виконана у відповідності до стандартів, що діють в Україні.

Попередній захист проведено 18 червня 2024 р.

Керівник кваліфікаційної роботи доц. Хряпкін О.В

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук

Кафедра Системотехніки

Рівень вищої освіти другий(магістерський))

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Освітня програма Комп'ютерні науки
(повна назва)

ЗАТВЕРДЖУЮ:
Зав.кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Щербаку Олександру Андрійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Дослідження моделей та методів ETL і ELT трансформації, оркестрування для побудови корпоративних сховищ даних
затверджена наказом по університету від 09 червня 2024 р. № 684Ст
2. Термін подання студентом роботи до екзаменаційної комісії 19 червня 2024 р.
3. Вихідні дані до роботи: Провести дослідження методів для задачі прогнозування та аналіз впливу вхідних атрибутів на результат.
4. Перелік питань, що потрібно опрацювати в роботі : Аналіз предметної області та формулювання задачі, аналіз існуючих систем, дослідження методів для задачі та аналіз вхідних атрибутів
5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп’ютерних ілюстрацій (слайдів) Рисунок 1.1 – Етапи даних, Рисунок 1.2 - Приклад на основі архітектури Data Lakehouse, Рисунок 1.3 - Data platform, Рисунок 1.4 - Data Warehouse, Рисунок 2.1- Інтеграція Data Warehouse та Data Lake, Рисунок 2.2 - Інтерфейс Apache Airflow ,Рисунок 2.3 - Apache Airflow DAG ,Рисунок 2.4 - Приклад коду який описує DAG та таски, Рисунок 2.5 - Поверхнева архітектура AWS EKS,Рисунок 3.1 - Інтерфейс Amazon Athena,Рисунок 3.2 - Логотип Apache Spark,, Рисунок 4.1 - Створення EKS кластеру за допомогою Terraform, Рисунок 4.2 - Створення бакету для даних які вже відфільтровані, Рисунок 4.3 - Структура префіксів в бакеті аналітики,Рисунок 4.10 - Забір за допомогою DBT моделі, Рисунок 4.11 - Вставка за допомогою DBT моделі, Рисунок 4.12 - трансформація за допомогою DBT моделі, Рисунок 4.13 - Приклад використання BashOperator для запуску DBT моделі, Рисунок 4.14 - Частина Airflow DAG для DBT моделей, Рисунок 4.15 - Kubernetes поди Superset та допоміжних інструментів

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

КАЛЕНДАРНИЙ ПЛАН

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	доц. Хряпкін О.В		

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	01.04.2024	виконано
2	Аналіз предметної області та постановка задачі	21.04.2024	виконано
3	Аналіз методів для вивчення	30.04.2024	виконано
4	Огляд літератури	04.05.2024	виконано
5	Теоретичне лослідження методів для аналізу	10.05.2024	виконано
6	Дослідження вибраних методів та впливовості вхідних атрибутів	25.05.2024	виконано
7	Оформлення пояснювальної записки	15.06.2024	виконано
8	Представлення на рецензування	17.06.2024	виконано
9	Попередній захист	18.06.2024	виконано
10	Захист перед ЕК	20.06.2024	виконано

Дата видачі завдання _1_ _квітня__2024 р.

Студент_____ _____Щербак О.А._

(підпис) (прізвище, ініціали)

Керівник роботи _____ _____доц. Хряпкін О.В._____

(підпис) (посада прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить: 57 с., 36 рис., 2 додатки, 28 джерел.

ІНЖЕНЕРІЯ ДАНИХ, КАТАЛОГ ДАНИХ, КОРПОРАТИВНЕ СХОВИЩЕ, ОЗЕРА ДАНИХ, ОРКЕСТРАЦІЯ, ПЛАТФОРМА ДАНИХ, BATCH PROCESSING, ELT, ETL, REAL-TIME PROCESSING.

Об'єкт дослідження – архітектура та побудова сучасних платформ з даних за допомогою сучасних фреймворків.

Предмет дослідження – процес створення платформи даних, яка відповідає сучасним вимогам до сховищ даних та слугує єдиним джерелом правди (“single source of truth”) за допомогою оркестраторів, технологій обробки даних у реальному часі та batch processing. Під час дослідження моделей та методів ETL і ELT трансформації, оркестрації для побудови корпоративних сховищ даних було використано датасет з реальними даними проекту 1000 Genomes для перевірки продуктивності та ефективності завантаження.

Мета роботи – зробити порівняльний аналіз методів ELT та ELT та розробити платформу з даних, виходячи з переваг та можливостей сучасних технологій.

Методи дослідження – порівняння різних видів технологій та спроба скласти докупи вузькопрофільні технології роботи з даними.

У кваліфікаційній роботі представлено порівняння та опис різних видів інструментів які можуть слугувати основою для сучасних платформ з даних. Детальний опис найбільш підходящої для більшості компаній архітектури платформи з даних.

ABSTRACT

Explanatory note: 57 figs., 36 pictures, 28 sources.

DATA PLATFORM, DATA CATALOG, DATA GOVERNANCE, DATA LAKES, DATA WAREHOUSE, DATA ENGINEERING, ELT, ETL, ORCHESTRATION, REAL-TIME PROCESSING

The object of research is the architecture and construction of modern data platforms with the help of modern frameworks.

The subject of research is the process of creating a data platform that meets modern requirements for data warehouses and serves as a single source of truth using orchestrators, real-time data processing technologies, and batch processing. During the study of models and methods of ETL and ELT transformation, orchestration for building enterprise data warehouses, a dataset with real data of the 1000 Genomes project was used to test the performance and efficiency of loading.

The purpose of the work is to develop a platform from data, based on the advantages and capabilities of modern technologies.

Research methods - comparison of different types of technologies and an attempt to put together narrow-profile technologies for working with data.

The certification work presents a comparison and description of different types of tools that can serve as a basis for modern data platforms. Detailed description of the most suitable data platform architecture for most companies.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DL (Deep Learning) - Глибинне навчання;

ML (Machine Learning) – Машинне навчання;

DAG (Directed Acyclic Graph) – направлений ациклічний граф;

ELT (Extract Load Transform) - завантаження даних з пре-обробкою;

ETL (Extract Transform Load) - завантаження даних з пост-обробкою;

DataOps (Data Operations) - концепція та набір практик безперервної інтеграції даних;

DP (Data Platform) - платформа з даних;

DE (Data Engineering) - інженерія з даних;

DS (Data Science) - наука з даних;

DL (Data Lake) - озеро даних;

DLH (Data Lakehouse) - інтеграція озера даних з копоративним сховищем;

DW (Data Warehouse) - корпоративне сховище даних;

DG (Data Governance)- політика поведінки з даними;

DL (Data Lineage) - граф потоку даних;

OLTP (Online Transaction Processing) база даних - система управління транзакційними базами даних;

DO (Data Orchestration) - оркестрація даними;

RT (Real Time) - режим реального часу;

BP (Batch Processing) - обробка пакетами;

AI (Artificial Intelligence) - штучний інтелект;

UI (User Interface) - інтерфейс користувача;

BI (Business Intelligence) - інструментів та технологій для збирання, аналізу та обробки даних;

IaaS (Infrastructure as a Service) - інфраструктура як сервіс;

PaaS (Platform as a Service) - платформа як сервіс;

SaaS (Software as a Service) - програмне забезпечення як сервіс;

IaaC (Infrastructure as a Code) - інфраструктура як код;

MPP (Massive Parallel Processing);

SQL (Structured Query Language);

AWS (Amazon Web Services);

EKS (Elastic Kubernetes Service);

IAM (Identity & Access Management);

High-tech - найсучасніші технології;

Data-driven company - компанії керовані даними;

Big Data – «великі” дані.

ВСТУП

Зараз, коли інформаційні технології розвиваються стрімко, наука про дані привертає значну увагу. Цей напрямок суттєво змінив життя багатьох людей: автономні автомобілі, переклад тексту, розпізнавання голосу та інше. Ми всі знаємо про такі відомі компанії, як Uber (додаток для каршерингу), Spotify (додаток для прослуховування музики) та Tesla (автомобілі, які змінили підхід до виробництва). Tesla фактично є не просто автомобілем, а високотехнологічним пристроєм. Всі ці компанії заробляють у різних галузях, але спільним є те, що вони є "data-driven" компаніями, інвестуючи значні кошти у розробку AI алгоритмів.

Аналізуючи будь-яку компанію, стає зрозуміло, що основою всіх процесів аналізу є збір даних, на яких базується аналіз та візуалізація. Нейронні мережі можуть давати чудові результати за умови "чистих" даних, але на практиці отримати такі дані не завжди можливо. Дані, з якими працюють алгоритми Data Science, збираються з різних систем: мобільних додатків, вебсайтів, OLTP баз даних різних видів, а інколи закуповуються у вендорів. В еру соціальних мереж значний трафік купується в Facebook, YouTube та Instagram. Ці дані часто є різнорідними та не "чистими". Наприклад, розробники, що створювали UI для сайту, можуть не налаштувати валідацію форм, що призводить до потрапляння невалідних даних у систему. Якщо подивитися, скільки часу науковець з даних витрачає на підготовку датасетів, то це близько 80%, а лише 20% йде на відбір параметрів для алгоритмів та отримання інсайтів. Це лише підготовка до аналізу, а дані ще потрібно зібрати, побудувати інфраструктуру, налаштувати пайплайни для очищення, об'єднання та трансформації. Тут на допомогу приходить Data Engineering.

Зважаючи на вищезазначене, обробка великих обсягів даних стає ключовим завданням для багатьох організацій. Системи ETL (Extract, Transform, Load) та ELT (Extract, Load, Transform) відіграють важливу роль

у процесах витягання, трансформації та завантаження даних у корпоративні сховища. Оркестрація та автоматизація цих процесів є критичними для забезпечення надійності, масштабованості та ефективності систем обробки даних. Використання сучасних інструментів для оркестрації та автоматизації, таких як Apache Airflow, Prefect та Kubernetes, значно покращує продуктивність та знижує витрати на підтримку систем.

Метою цієї роботи є дослідження моделей та методів ETL і ELT трансформації, оркестрація, а також вибір найкращого підходу для побудови сучасної платформи даних. Для досягнення цієї мети передбачається виконання таких завдань:

1. Аналіз сучасного стану технологій ETL та ELT.
2. Дослідження інструментів для оркестрації процесів обробки даних.
3. Розробка моделей автоматизації ETL/ELT процесів.
4. Реалізація прототипу системи для обробки даних.
5. Проведення експериментальних досліджень ефективності запропонованих підходів.

Наукова новизна роботи полягає у розробці нових моделей та методів ETL і ELT трансформації, а також підходів до оркестрації та автоматизації цих процесів у корпоративних сховищах даних. Запропоновані рішення дозволять підвищити ефективність обробки даних, знизити витрати на їх підтримку та забезпечити більш високу надійність систем.

Практичне значення роботи полягає у можливості застосування отриманих результатів для створення ефективних та надійних систем обробки даних у великих організаціях. Розроблені моделі та методи можуть бути впроваджені в різних галузях, таких як фінанси, охорона здоров'я, телекомунікації, що дозволить підвищити продуктивність та ефективність управління даними.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз сучасного стану в області.

Термін "інженерія даних" виник на початку 2010-х років і почав активно використовуватися в сучасних високотехнологічних і швидкозростаючих компаніях, таких як Lyft, Uber, LinkedIn і Airbnb. Ці компанії використовували величезні обсяги даних у реальному часі, що приносило високу цінність для бізнесу. Інженерам програмного забезпечення в цих компаніях довелося розробляти інструменти, платформи та фреймворки для ефективного управління цими даними з необхідною швидкістю, масштабованістю та надійністю. З цього моменту "інженерія даних" почала активно розвиватися, переходячи від традиційних інструментів ETL до розробки власних інструментів для роботи зі зростаючими обсягами даних.

Метт Терк в своєму останньому аналізі машинного навчання, штучного інтелекту та ландшафту даних зазначає[3]:

“Сьогодні хмарні сховища даних (Snowflake, Amazon Redshift і Google BigQuery) та lakehouses (Databricks) надають можливість зберігати величезні обсяги даних у такий спосіб, що це не є надто дорогим і не вимагає армії технічних фахівців для підтримки. Іншими словами, після всіх цих років тепер нарешті можливо зберігати та обробляти великі дані.“

Хмарні обчислення стали дуже потужним інструментом для роботи з даними. Зак Вілсон зазначає [5]:

“Багато завдань з інженерії даних, які були громіздкими у світі Hadoop, значно спрощені за допомогою інструментів, таких як Snowflake або BigQuery. Вони дозволяють виконувати те, що раніше вимагало сотні рядків Java, лише в декілька десятків рядків SQL.“

Міхай Леріч висловлює думку [4]:

“Нам не потрібні науковці з даних, нам потрібні інженери з даних". Він провів аналіз вакансій для інженерів з даних, які були найняті кожною компанією, що виходить з Y-Combinator з 2012 року, і показав, що компанії потребують на 70% більше інженерів з даних, ніж науковців з даних. Це означає, що в сучасних реаліях платформи, які будують інженери, є дуже важливими для організацій.

З огляду на усвідомлення справжнього потенціалу великих обсягів даних, можна впевнено сказати, що інженерія з обробки даних буде імплементуватися в будь-якій сучасній компанії, яка керується даними, незалежно від її розміру — від стартапів до великих підприємств. Інженерія даних включає розробку, впровадження та підтримку інфраструктури, систем і процесів, які забезпечують надійні, якісні та послідовні потоки даних, що підтримують бізнес-аналітику, аналітичні дослідження та машинне навчання. Інженерія даних існує на перетині таких областей, як Data Science, DevOps і розробка програмного забезпечення.

Інженери з даних створюють надійні дані, які обслуговують бізнес з передбачуваною якістю та значенням, побудовані на основі інфраструктури та систем, які підтримують ці дані — від потокової передачі в реальному часі до конвеєрів пакетних даних. Інженерія даних як дисципліна відповідає за переміщення, формування і перетворення даних. Основне завдання інженерії даних — створення конвеєрів даних. Збір і отримання точних і надійних даних є ключовим для будь-якої організації, яка керується даними.

Більшість конвеєрів даних і обробки даних у минулому базувалися на пакетній обробці (Batch Processing). Дані передаються між системами в пакетних знімках ETL, а дані обробляються періодично за допомогою планувальників завдань (Airflow, Prefect тощо). Після багатьох років розвитку ми спостерігаємо поступовий перехід до конвеєрів даних у реальному часі (Real-Time) та систем обробки даних у реальному часі.

Витрати на обмін даними в реальному часі та можливості потокової передачі значно знизилися, що відкриває шлях до їх масового використання.

Конвеєри даних є великою частиною інженерії даних, але це не єдине завдання. Фактично, якщо подивитися на рисунок 1.1, то видно, що завдання інженерії даних включають перші три етапи обробки даних для проведення початкового аналізу.

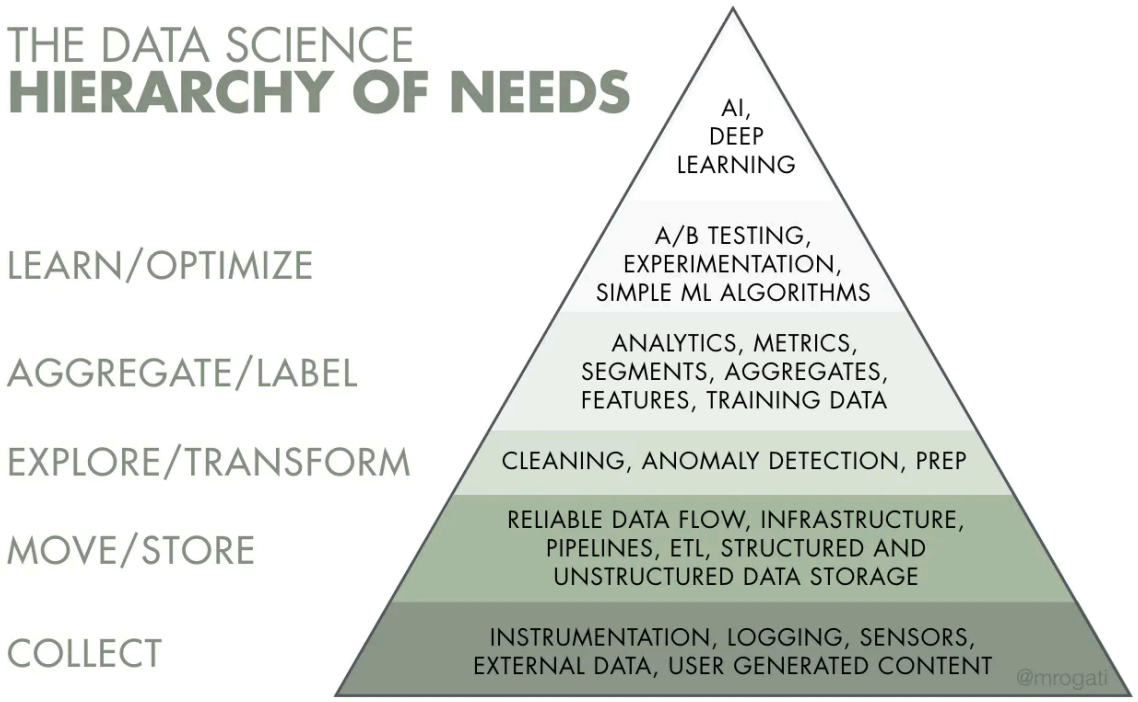

Рисунок 1.1 - Етапи даних

Важливим є побудова DP як такої, яка може відповідати новим викликам [6].

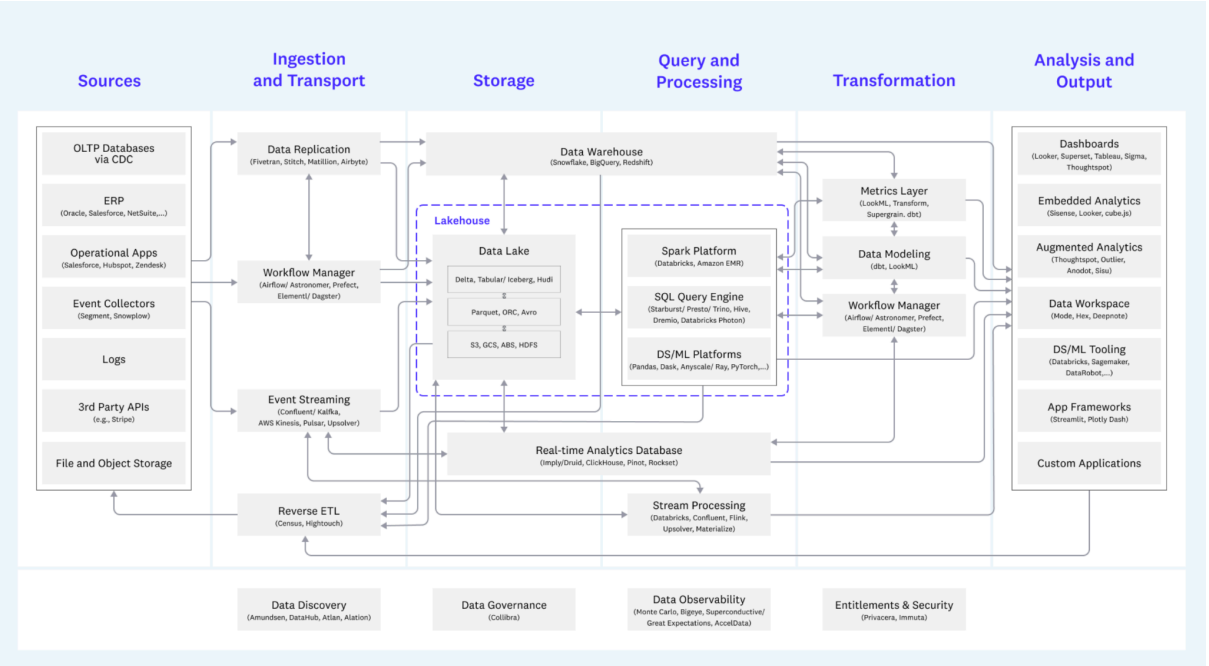

Рисунок 1.2 - Приклад архітектури на основі Data Lakehouse

1.2 Платформа з даних, як єдина точка правди для всієї компанії.

DP — це технологія, яка дає змогу збирати, трансформувати, уніфікувати та доставляти дані користувачам, додаткам або використовувати їх для інших цілей бізнес-аналітики.

Якщо говорити більш формально, це забезпечує доступ до даних, керування, доставку та безпеку. DP є результатом невідповідності та надмірності даних всередині організації. Це вирішує проблему і дозволяє нам організувати, трансформувати та обслуговувати дані кінцевим користувачам - науковцям з даних та аналітикам.

Замість того, щоб мати кілька джерел даних, створених різними клієнтами та різними варіантами їх використання, DP дозволяти мати єдину точку входу для усієї організації.

В сучасних реаліях «великих даних» платформа повинна виконувати кілька завдань: (рисунок 1.3).

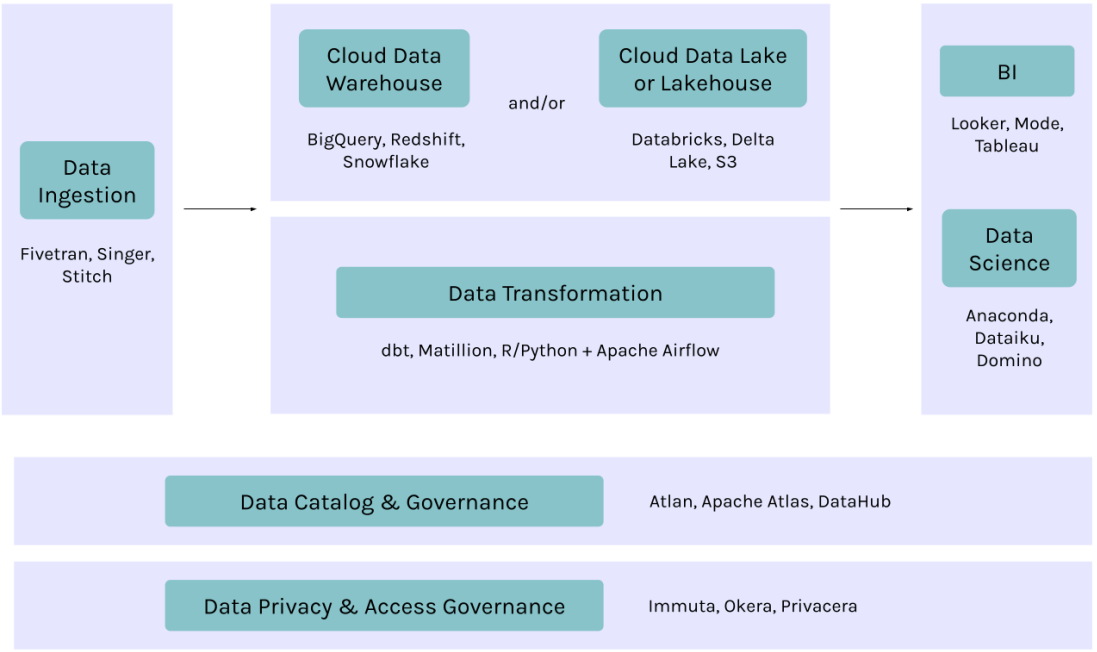

Рисунок 1.3 - Data platform

1. Сховище даних, яке буде слугувати для того, щоб зберегти дані з усіх систем компанії - це можуть бути DL, DW або їхня інтеграція - Data Lakehouse. Сховище даних (DW) витягує структуровані дані та працює в реляційній базі даних. Приклад послуги — Data Vault 2.0. Озеро даних (DL) витягує та зберігає дані в оригінальному форматі з мінімальною уніфікацією або трансформацією. Історичні дані зберігаються в DL. Приклад сервісу — Apache Hadoop. Дані в DL та DW потім використовуються як вхідні дані для різних продуктів, таких як системи рекомендацій та націлювання оголошень. Структуровані дані в DW в основному використовуються аналітиками для виконання завдань BI.

2. ETL та ELT процеси, які заповнюють дані у сховищах та будуть перетворювати дані в потрібний формат.

3. В багатьох випадках, коли Data Scientist або Data Analyst починає шукати дані для аналізу, вони витрачають дуже багато часу для пошуку даних на основі яких можна зробити якесь дослідження, це особливо складне завдання в еру Big Data, коли дані піддаються пяти

основним характеристикам - 5Vs [7], їх об'єм та різноманітність дуже великі. Потрібен аудит: які дані компанія має, звідки вони з'явилися та які трансформації пройшли. Детальна інформація допомагає краще зрозуміти значення та достовірність цих даних. Треба імплементувати так званий DC, а також DG - не всім потрібно мати доступ до всіх даних, щоб зменшити ймовірність витоку даних клієнта.

4. Платформа з даних повинна мати також BI інструмент для менеджмента із різних департаментів, в якому вони можуть дивитись результати аналізів.

Беручи до уваги все вище сказане, ми можемо винести основні характеристики платформи даних [8]:

1. Доступність - платформа даних повинна бути висока доступною для клієнтів і кінцевих користувачів, таких як аналітики даних і науковців з даних.

2. Управління - коли дані з’являються, підприємство має сувору політику. Сучасна платформа даних повинна забезпечувати легкість увімкнення або оновлення стратегій керування даними.

3. Безпека - хто має доступ до даних і скільки точок доступу до даних може бути доступним. DP повинна надавати легкість у налаштуванні доступу до даних. Сьогодні багато служб використовують систему аутентифікації типу єдиного входу (SSO), щоб забезпечити одноточковий доступ всім, хто має доступ до даних.

4. Централізація - хороша платформа даних повинна підтримувати всі типи джерел, як-от PostgreSQL, BigTable, DynamoDB, Redis тощо.

1.3 Зберігання даних.

Для створення частини платформи, відповідальної за зберігання даних, можна використовувати різні архітектурні підходи [9]: основними з

них є data warehouse, data lake та data lakehouse. Зі зростанням залежності компаній від даних для прийняття важливих бізнес-рішень, покращення продуктів та підвищення якості обслуговування клієнтів, обсяг даних, які вони збирають, збільшується безпрецедентними темпами. Згідно з дослідженням Domo, у 2017 році щодня генерувалося 2,5 квінтильйони байтів даних, а до 2025 року цей показник може зрости до 463 екзабайт. Проте, яка користь від усіх цих даних, якщо компанії не можуть швидко їх використовувати? Питання оптимального зберігання даних для аналітики довгий час залишалося актуальним.

1.3.1 Data Warehouse.

DW - це уніфіковане сховище, призначене для зберігання великої кількості даних, зібраних з різних джерел в організації. Воно слугує єдиним джерелом «істинних даних» в організації і є ключовим компонентом для звітності та бізнес-аналітики.

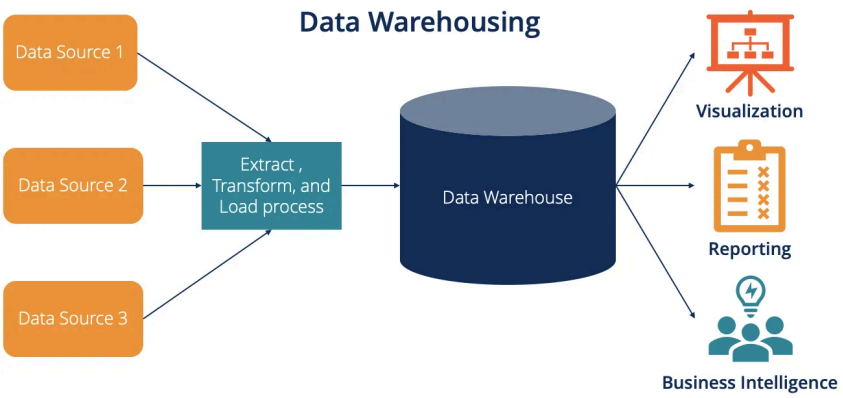

Рисунок 1.4 - Data Warehouse

Як правило, сховища даних зберігають історичні дані, об’єднуючи реляційні набори даних із кількох джерел, таких як додатки та транзакції.

Вони витягують дані з різних джерел, перетворюють та очищають їх перед завантаженням у систему, щоб служити єдиним джерелом правди даних. Організації інвестують у сховища даних через їхню здатність швидко надавати бізнес-інсайти з усієї організації.

Сховища даних дозволяють бізнес-аналітикам, інженерам даних і особам, які приймають рішення, отримувати доступ до даних через інструменти BI, клієнти SQL та інші аналітичні додатки. Вони пропонують численні переваги, такі як:

1. Покращення стандартизації, якості та узгодженості даних: організації збирають дані з різних джерел, таких як продажі, користувачі та транзакції. Сховища даних об’єднують ці корпоративні дані в єдиний стандартизований формат, який може служити єдиним джерелом достовірної інформації, забезпечуючи організацію впевненістю у своїх даних для бізнес-потреб.

2. Забезпечення розширеної бізнес-аналітики: сховище даних усуває розрив між великими обсягами необроблених даних, які часто автоматично збираються, та агрегованими даними, які допомагають зрозуміти бізнес-процеси. Вони є основою для зберігання даних в організаціях, дозволяючи відповідати на складні запитання та використовувати отримані відповіді для прийняття обґрунтованих бізнес-рішень.

3. Збільшення потужності та швидкості аналізу даних і робочого навантаження бізнес-аналітики: сховища даних значно прискорюють час, необхідний для підготовки та аналізу інформації. Завдяки узгодженості та точності даних, вони легко інтегруються з інструментами аналізу даних та бізнес-аналітики. Сховища даних також зменшують час, потрібний для збору інформації, надаючи командам можливість використовувати ці дані для створення звітів, інформаційних панелей та інших аналітичних завдань.

4. Покращення загального процесу прийняття рішень: data warehouse покращує процес прийняття рішень, надаючи єдине сховище для поточних та історичних даних. Це дозволяє особам, які приймають рішення, оцінювати ризики, розуміти потреби клієнтів та вдосконалювати продукти і послуги, використовуючи дані зі сховищ для отримання точних інсайтів.

DW надають підприємствам високопродуктивну та масштабовану аналітику. Однак вони створюють конкретні проблеми, деякі з яких включають:

1) відсутність гнучкості даних: хоча сховища даних добре працюють зі структурованими даними, вони можуть боротися з напів структурованими і неструктурованими форматами даних, такими як аналітика журналів, потокове передавання даних і дані соціальних мереж. Через це важко рекомендувати сховища даних для машинного навчання та використання штучного інтелекту;

2) високі витрати на впровадження та обслуговування: сховища даних можуть бути дорогими для впровадження та обслуговування. Хоча зараз ціна за обробку та зберігання інформації дешевше ніж це було 10-15 років тому, але все одно це дорого якщо порівнювати з DL.

1.3.2 Data lake.

DL - це централізоване, гнучке сховище, яке зберігає значні обсяги як структурованих, так і неструктурованих даних у їхньому сирому, неформатованому стані. На відміну від data warehouse, де зберігаються очищені реляційні дані, озеро даних використовує плоску архітектуру для зберігання даних у початковому вигляді. Озера даних забезпечують гнучкість, тривале зберігання та економічність, дозволяючи організаціям отримувати глибші інсайти з неструктурованих даних, що часто викликають труднощі у сховищ даних.

В озерах даних структура та формат даних не визначаються на етапі збору; натомість дані спочатку зберігаються, а потім перетворюються (ELT) для аналізу. Озера даних підтримують машинне навчання та прогнозну аналітику, використовуючи різноманітні джерела, включаючи пристрої Інтернету речей, соціальні мережі та потокові дані.

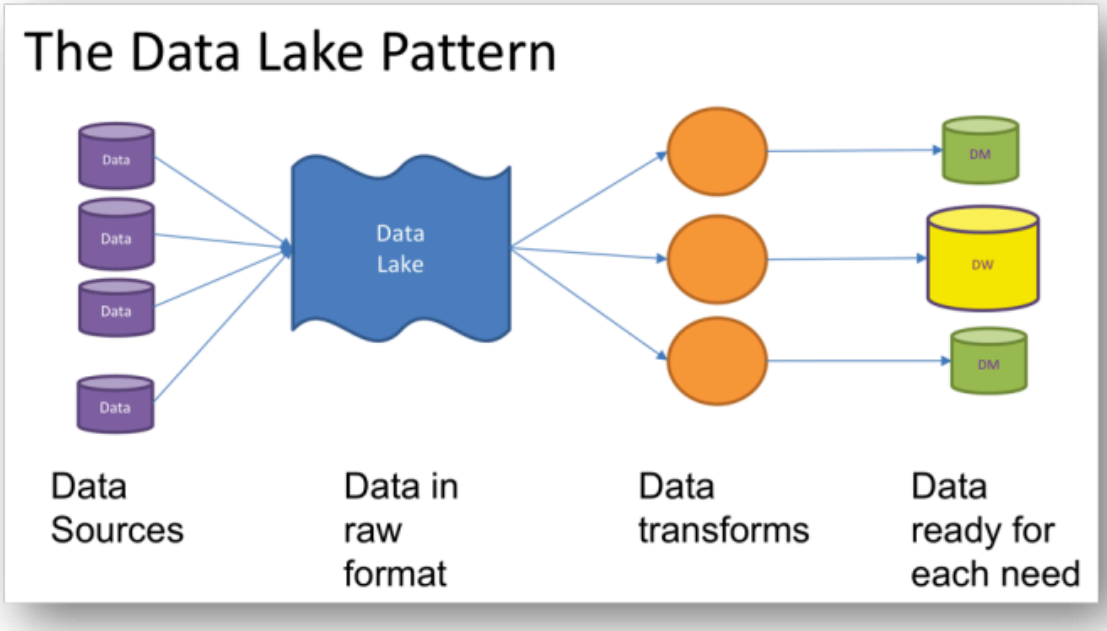

Рисунок 1.5 - Data lake

Оскільки озера даних можуть зберігати як структуровані, так і неструктуровані дані, вони пропонують ряд переваг, таких як:

1. Консолідація даних: озера даних дозволяють зберігати всі типи організаційних даних у єдиному сховищі, усуваючи потребу зберігати структуровані та неструктуровані дані в різних середовищах.

2. Гнучкість даних: значною перевагою озер даних є їхня здатність зберігати інформацію в будь-якому форматі без попередньо визначеної схеми. Це дозволяє залишати дані в їхньому початковому

вигляді, надаючи більше можливостей для аналізу та підтримуючи майбутні сценарії використання.

3. Економія витрат: озера даних зазвичай дешевші за традиційні сховища даних, оскільки вони розроблені для зберігання на недорогому обладнанні, наприклад, об’єктних сховищах. Наприклад, Amazon S3 пропонує зберігання за низькою ціною — 0,023 долара за ГБ для перших 50 ТБ на місяць.

4. Підтримка широкого спектру прикладів використання науки про дані та машинного навчання: оскільки дані в озерах даних зберігаються у відкритому необробленому форматі, це полегшує застосування різних алгоритмів машинного і глибокого навчання для отримання значущої інформації.

Хоча озера даних пропонують чимало переваг, вони також викликають проблеми:

1. Низька продуктивність у випадках використання бізнес-аналітики та аналітики даних: якщо не керувати озером даних належним чином, воно може стати дезорганізованим, що ускладнює інтеграцію з інструментами бізнес-аналітики та аналітики даних. Відсутність узгодженої структури даних і підтримки транзакцій ACID (атомність, узгодженість, ізоляція та довговічність) може призвести до не оптимальної продуктивності запитів, що важливо для звітів та аналітичних потреб.

2. Відсутність надійності та безпеки даних: відсутність узгодженої структури даних в озерах даних ускладнює забезпечення надійності та безпеки даних. Оскільки озера даних можуть містити всі формати даних, може бути важко запровадити ефективну політику безпеки та керування для захисту конфіденційної інформації.

1.3.3 Data lakehouse.

DW і DL були найбільш широко використовуваними архітектурами зберігання великих даних. DLH це — це нова архітектура зберігання даних, яка поєднує в собі гнучкість озер даних і керування даними сховищ даних.

DLH поєднує найкращі риси як сховищ даних, так і озер даних, забезпечуючи єдине сховище для всіх типів даних (структурованих, напівструктурованих і неструктурованих). Водночас, воно надає найкращі у своєму класі можливості для машинного навчання, бізнес-аналітики та потокової передачі даних.

DLH зазвичай починаються як озера даних, що містять усі типи даних. Потім дані перетворюються у формат Delta Lake (рівень зберігання з відкритим кодом, який забезпечує надійність озер даних). Озера Delta забезпечують транзакційні процеси ACID із традиційних сховищ даних на озерах даних.

Архітектура озера даних поєднує в собі структуру даних і функції керування сховищем даних з недорогим зберіганням і гнучкістю озера даних. Переваги такого впровадження величезні і включають:

a) зменшена надлишковість даних. Озера даних зменшують дублювання даних, надаючи єдину універсальну платформу для зберігання даних для задоволення всіх запитів бізнес-даних. Через переваги сховища даних і озера даних більшість компаній вибирають гібридне рішення. Однак такий підхід може призвести до дублювання даних, що може бути дорогим;

b) економічність: будинки озер даних реалізують економічні функції озер даних, використовуючи недорогі варіанти зберігання об’єктів. Крім того, озеро даних позбавляють витрат і часу на обслуговування кількох систем зберігання даних, надаючи єдине рішення;

c) підтримка широкого спектру робочих навантажень: DLH забезпечують прямий доступ до інструментів бізнес-аналітики для

забезпечення розширеної аналітики. Крім того, у озерах даних використовуються формати відкритих даних (наприклад, Parquet) з API та бібліотеками машинного навчання, включаючи Python/R, що полегшує використання даних науковцям із даних та інженерам з машинного навчання;

d) простота керування версіями даних, управління та безпека: архітектура DLH забезпечує цілісність схеми та даних, що полегшує впровадження надійних механізмів безпеки та управління.

Основним недоліком DLH є те, що це все ще відносно нова і незріла технологія. Таким чином, неясно, чи виконає вона свої обіцянки. Можуть пройти роки, перш ніж озери даних можуть конкурувати зі зрілими рішеннями для зберігання великих даних.

DW є хорошим вибором для компаній, які шукають зріле, структуроване рішення для даних, яке зосереджено на бізнес-аналітиці та аналізі даних. Проте DL підходять для організацій, які шукають гнучке, недороге рішення з великими обсягами даних для керування машинним навчанням і науковими навантаженнями на неструктурованих даних.

Якщо потрібно реалізувати обидва рішення, то гарним підходом може бути DL [10].

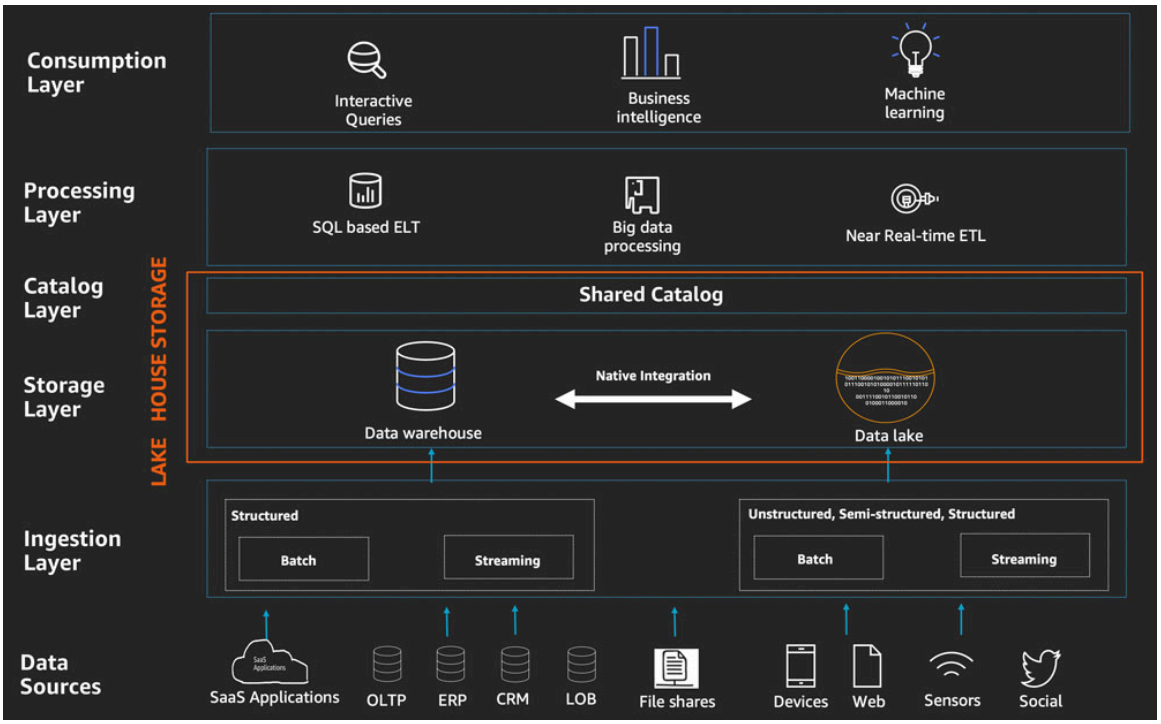

Рисунок 1.6 - Data lakehouse

1.4 Каталог даних та політика управління даними.

Каталог даних [11] — це перелік доступних даних і метаданих, часто доповнений інструментом пошуку, що допомагає користувачам швидко знаходити та оцінювати придатність даних для передбачуваного використання. Каталог даних зосереджується на наборах даних, збагачуючи їх метаданими. Набори даних можуть включати файли та таблиці, розміщені в озері даних, сховищі, сховищі основних даних або інших спільних ресурсах.

Каталог даних також реалізує data lineage [12], процес, що відображає повний шлях даних від однієї системи до іншої. Він показує шлях, який пройшли дані, дає розуміння їх сутності, дату запису та візуалізацію процесів у зручному форматі. Найбільш поширеним форматом візуалізації є DAG (Directed Acyclic Graph), який показаний нижче на рисунку 1.7.

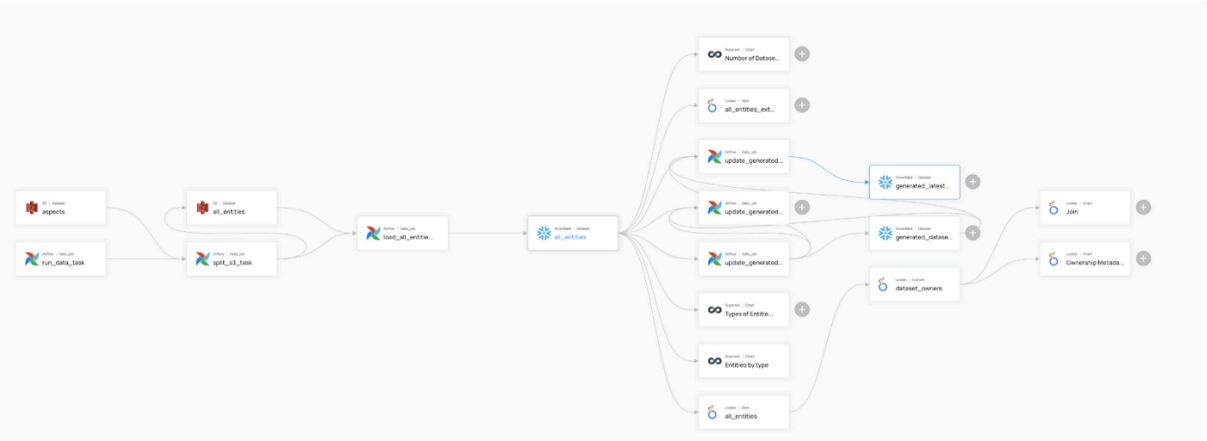

Рисунок 1.7 - Data lineage

Найбільша цінність каталогу даних полягає в підвищенні продуктивності роботи команд та забезпеченні співпраці. У більшості організацій дані та технології існують у відокремленому вигляді, через що команди часто працюють без повного уявлення про доступні набори даних. Це змушує їх витрачати надто багато часу на пошук та розуміння даних, часто відтворюючи вже існуючі набори. Без каталогу даних команди покладаються на документацію та знання окремих людей, що призводить до численних обговорень між членами команди і знижує продуктивність. Однак, для підтримання актуальності та корисності, каталог даних та пов’язані з ним метадані повинні постійно керуватися та збагачуватися.

Дві основні переваги для бізнесу від впровадження каталогу даних - CD:

1) підвищена продуктивність команд обробки даних. CD допомагає будь-кому в організації швидко знайти потрібні дані для свого випадку. За допомогою платформи час на пошук та аналіз змінюється зворотно пропорційно: 20% часу на пошук даних і 80% часу на аналіз;

2) довіра та відповідність даним. CD допомагає групам даних довіряти даним, які надходять з надійного джерела, наприклад надійного власника даних, найбільш часто використовуваних наборів даних. Крім того, CD допомагає командам даних виявляти проблеми з відповідністю даних, наприклад ідентифікувати дані, що ідентифікують інформацію, що

вже відома. Тобто платформа вже має дані, тобто дубляж можливо видалити і зменшити об'єм даних, які треба зберігати, а отже і платити за зберігання. На великих об'ємах це може зберегти дуже великі суми коштів.

1.5 Інструменти для бізнес аналітики.

Візуалізація даних [13] – це спосіб представлення даних у графічному або візуальному форматі. Це допомагає особам, які приймають рішення, бачити аналітику наочно, що дозволяє їм зрозуміти складні концепції та виявити нові закономірності. Завдяки інтерактивній візуалізації можна деталізувати діаграми і графіки, змінюючи дані та спосіб їхнього відображення в реальному часі.

Оскільки людський мозок краще обробляє візуальну інформацію, використання діаграм та графіків для представлення великої кількості складних даних є ефективнішим, ніж перегляд електронних таблиць чи звітів. Візуалізація даних є швидким та простим способом передавання концепцій у зрозумілий для всіх формат. Вона також дозволяє експериментувати з різними сценаріями, вносячи невеликі зміни для оцінки їхнього впливу.

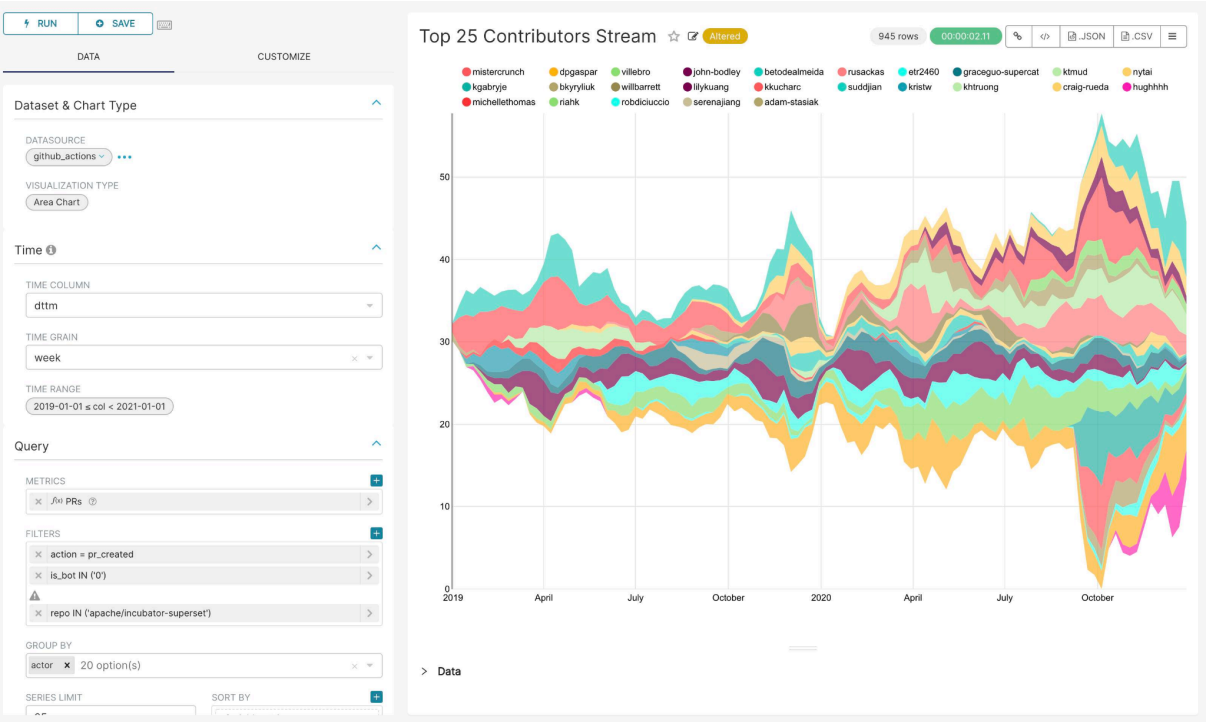

Рисунок 1.8 - Data Visualization

1.6 Постановка задачі.

В рамках робити спочатку треба порівняти технології, зробити порівняльний аналіз методів ELT та ETL.

Для кожного етапу треба зробити замір таких показників:

1. Час обробки даних
2. Пропускна здатність
3. Використання обчислювальних ресурсів

При порівнянні взяти за основу такі критерії для порівняння:

1. Масштабованість
2. Вартість впровадження
3. Складність реалізації
4. Час обробки
5. Гнучкість

Після цього взяти найкращий варіант по більшості показників як основний елемент трансформації під час створення DP архітектури, яка може бути імплементована в будь-якій компанії незалежно від її розмірів та бюджету, слід вибрати набір інструментів, що задовольняє наступні вимоги:

Платформа повинна мати усі основні описані характеристики:

1) бути легко масштабованою в незалежності від бізнес потреб та об'єму даних - мати MPP архітектуру;
2) обробляти як структуровані так і неструктуровані дані;
3) повинна мати реалізацію DG та DC;
4) містити сучасний BI інструмент в своій архітектурі;
5) повинна бути реалізованою за допомогою DataOps практик, щоб мати прозорість схем даних - IaaC.

2 ДОСЛІДЖЕННЯ ІНСТРУМЕНТІВ ДЛЯ ПОБУДОВИ DP

2.1 Побудова Lakehouse, їх інтеграція.

Архітектуру Lakehouse можна створювати як на популярних хмарних платформах - AWS, GCP, Microsoft Azure, так і на власних кластерах, використовуючи технології з відкритим кодом, такі як Apache Hadoop. Я обрав AWS, оскільки це комплексна платформа хмарних обчислень від Amazon, яка включає IaaS, PaaS та SaaS.

Amazon Athena та Amazon S3 забезпечують інтегрований рівень зберігання для еталонної архітектури Lakehouse. Amazon Athena зберігає структуровані, узгоджені та надійні дані, організовані у стандартні схеми, тоді як Amazon S3 надає ексабайтне сховище для структурованих, напівструктурованих та неструктурованих даних. Завдяки підтримці напівструктурованих даних в Amazon Athena можна також приймати та зберігати такі дані у Amazon S3.

Amazon S3 пропонує провідну масштабованість, доступність, безпеку та продуктивність у галузі. Організації зазвичай зберігають дані в Amazon S3, використовуючи відкриті формати файлів. Це дозволяє аналізувати ті самі дані за допомогою різних компонентів обробки та споживання. Загальний каталог зберігає схеми структурованих або напівструктурованих наборів даних в Amazon S3. Компоненти, які споживають ці дані, застосовують схему під час зчитування (schema-on-read).

Amazon Athena надає уніфікований SQL-інтерфейс, який може обробляти запити, що звертаються та об’єднують дані з озера даних та сховища даних. Amazon Athena може запитувати петабайти даних, збережених в Amazon S3, використовуючи архітектуру з тисячами тимчасових вузлів та складні оптимізації запитів. Athena може зчитувати

розділені дані в озері даних S3, стиснуті за допомогою кодеків з відкритим кодом та зберігані у форматах JSON, CSV, Avro, Parquet та ORC.

Оскільки AWS Athena зчитує дані з Amazon S3, він застосовує відповідну схему із загального каталогу AWS Lake Formation. Високоструктуровані дані в Amazon Athena зазвичай забезпечують інтерактивні запити та швидкі BI-панелі, тоді як структуровані, неструктуровані та напівструктуровані дані в Amazon S3 використовуються для машинного навчання, науки про дані та обробки великих даних.

Отже, AWS Athena - це движок для запитів, S3 - це Data Lake, а AWS Lake Formation - інструмент для інтеграції в Lakehouse (рис. 2.1).

Рисунок 2.1 - Інтеграція Data Warehouse та Data Lake

Для побудови та розгортання всіх інструментів у хмарі ми використаємо Terraform. Terraform — це інструмент від HashiCorp, який допомагає декларативно керувати інфраструктурою. Замість того, щоб вручну створювати AWS Athena, бакети в S3, таблиці та мета-таблиці в AWS Lake Formation через консоль AWS, достатньо написати конфігурацію, яка вказує на необхідні ресурси. Ця конфігурація буде створена у текстовому форматі, зрозумілому для людини.

2.2 Технології оркестрування.

Airflow — це платформа для програмного створення, планування та моніторингу робочих процесів [15]. Перші рішення для оркестрації даних, такі як Luigi та Airflow, встановили стандарт для побудови робочих процесів. Airflow здобув велику популярність завдяки своєму зручному користувацькому інтерфейсу та простій моделі програмування (рис. 2.2).

Airflow дозволяє створювати складні робочі процеси, які автоматизують ETL-процеси, дані та інтеграцію. Завдяки підтримці DAG (Directed Acyclic Graphs), робочі процеси можуть бути налаштовані для виконання завдань у певному порядку, забезпечуючи надійність та узгодженість даних. Інтерфейс Airflow дозволяє легко відстежувати та управляти завданнями, забезпечуючи гнучкість у плануванні та виконанні.

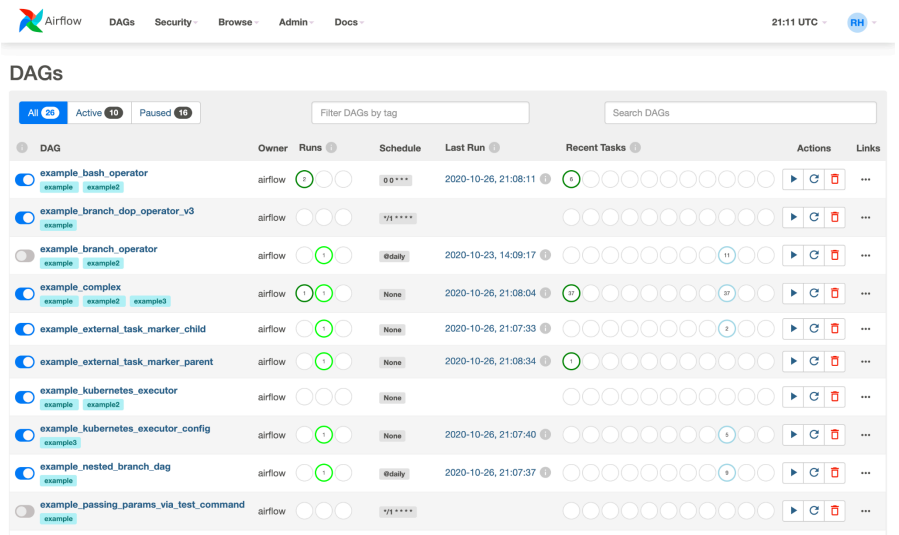

Рисунок 2.2 - Інтерфейс Apache Airflow

Крім того, Airflow підтримує розширення та інтеграцію з різними сервісами та інструментами, що робить його універсальним рішенням для компаній будь-якого масштабу. Це дозволяє організаціям оптимізувати свої робочі процеси та підвищити ефективність обробки даних. До Airflow

більшість розкладів мали Domain Specific Language (DSL), де ви писали JSON і YAML. Airflow у коді Python є основною мовою, яку використовують дослідники даних, полегшуючи можливість створення DAG. Airflow в основному використовується для завдань ETL.

Перше покоління рішень було зосереджено на тому, щоб бути керованим завданням, відокремлюючи управління завданнями від процесу завдання. Оркестратори мають обмежені знання про дані, які обробляють. Наприклад, якщо завдання X завершено, ініціювати завдання Y. Приклад завдань та їх порядок виконання наведений на рисунку 2.3.

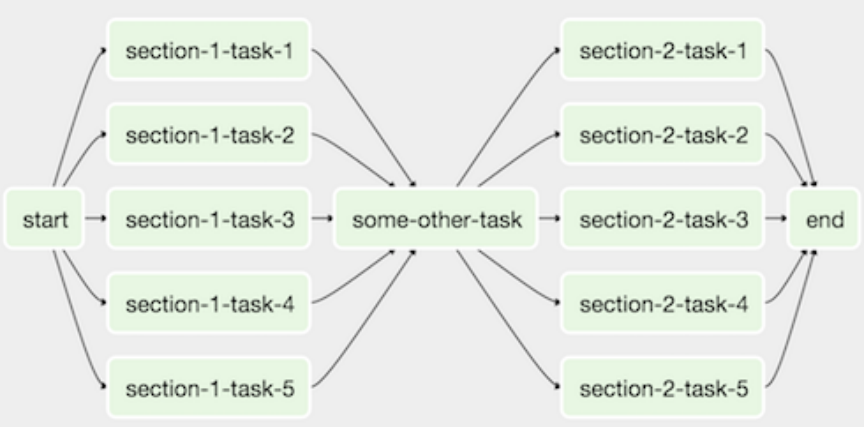

Рисунок 2.3 - Apache Airflow DAG

Існує також друге покоління рішень для оркестрування даних, таких як Prefect, Flyte і Dagster. Ці рішення зосереджені на тому, щоб бути керованими даними, розуміючи тип даних, які будуть перетворені, і як ними маніпулювати. Вони володіють обізнаністю з даними, можуть виконувати тести на артефакти даних і керувати версіями не лише коду, а й самих артефактів.

У оркестрації, керованій даними, коли дані X стають доступними, це автоматично запускає завдання Y, яке генерує артефакт, що в свою чергу викликає наступні дії. Такі підходи можуть бути активними або пасивними. Активний підхід передбачає активну передачу даних між кроками і

системами. Наприклад, Flyte автоматично розуміє завдання та потік даних між ними, використовуючи різні DSL Flyte, такі як Flytekit [14]. FlytePropeller потім планує ці завдання одне за одним і передає дані між ними по мірі завершення завдань. Це також розумно визначає, як дані повинні розподілятися між різними завданнями. Ця здатність до передачі стану є значним прогресом у порівнянні з системами першого покоління, такими як Airflow XCOM.

Однак, Apache Airflow залишається універсальним інструментом для запуску ETL та ELT процесів. Він простий у використанні, має велике ком'юніті та не потребує ліцензії. Якщо вам потрібно запустити якусь SQL задачу, є велика ймовірність, що відповідний оператор вже написаний і готовий до використання.

Розберемо основні сутності архітектури Apache Airflow:

1) DAG. DAG, або орієнтовані ациклічні графи, — це сукупність усіх завдань одиниць роботи, що знаходяться в конвеєрі. Таски організовані за їхніми взаємозв’язками та залежностями між собою. Наприклад, якщо ви хочете зробити запит до бази даних А, а потім завантажити результати в базу даних Б, ви хочете запустити завдання для запиту бази даних А безпосередньо перед завданням для завантаження результатів до бази даних Б. Орієнтований ациклічний графік означає, що ваш завдання виконуються зліва направо. Таску можна повторити, але таску не можна повторити після того, як воно було завершено і почалася вже наступна за нею таска;

2) Таски та оператори. Таски є ідеально незалежними частинами, які не покладаються на інформацію з іншого завдання. Під час виконання, об’єкти класу оператора перетворюються на завдання. Класи операторів можна імпортувати, і створення екземпляра класу створює об’єкт класу (рис. 2.4). Таскою є створення екземпляра (виконання коду оператора) цього об’єкта;

```
import datetime

import pendulum

from airflow import DAG
from airflow.operators.empty import EmptyOperator
from airflow.operators.latest_only import LatestOnlyOperator
from airflow.utils.trigger_rule import TriggerRule

with DAG(
    dag_id='latest_only_with_trigger',
    schedule_interval=datetime.timedelta(hours=4),
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
    tags=['example3'],
) as dag:
    latest_only = LatestOnlyOperator(task_id='latest_only')
    task1 = EmptyOperator(task_id='task1')
    task2 = EmptyOperator(task_id='task2')
    task3 = EmptyOperator(task_id='task3')
    task4 = EmptyOperator(task_id='task4', trigger_rule=TriggerRule.ALL_DONE)

    latest_only >> task1 >> [task3, task4]
    task2 >> [task3, task4]
```

Рисунок 2.4 - Приклад коду, який описує даг та таски

3) Змінні. Змінні є ще одним корисним компонентом Airflow. Ми використовуємо змінні для двох основних цілей: параметрів, пов’язаних із середовищем, і параметрів, що стосуються моделі. Змінні доступні у файлі DAG, і, наприклад, ідентифікатор проекту або тег зображення можна оновити без будь-яких змін DAG. Це особливо корисно, якщо у вас є кілька середовищ, таких як тести та продакшн;

4) Пули. Пули контролюють кількість ресурсів, які DAG може використовувати за один раз. Пули слід визначати залежно від того, як швидко закінчуються завдання та як швидко необхідно завершити DAG;

5) XCOM. В ідеалі таски незалежні одна від одної, але іноді це неможливо зробити, і таскам потрібно спілкуватися. Цей взаємозв'язок

здійснюється за допомогою XCOM. XComs дозволяє «push» (відправляти) або pull (отримувати) між тасками - вузли графа.

Може знадобитися перемістити значення, що повертається з однієї таски і отримати значення у наступній таскі та використовувати його як параметр.

Для розвороту Apache Airflow, ми створимо Kubernetes кластер за допомогою AWS EKS та Helm Chart Airflow.

Amazon Elastic Kubernetes Service (Amazon EKS) – це керований контейнерний сервіс для запуска та масштабування додатків Kubernetes в хмарі AWS та локальному середовищі.

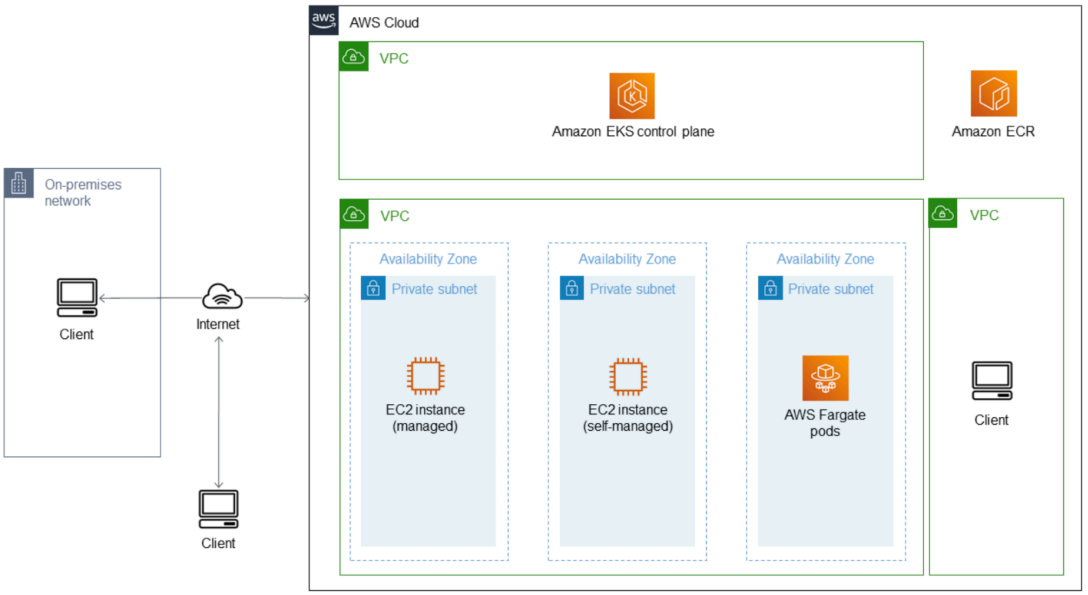

Рисунок 2.5 - Поверхнева архітектура AWS EKS

За допомогою Helm можна описувати які об'єкти Kubernetes ми хочемо створити.

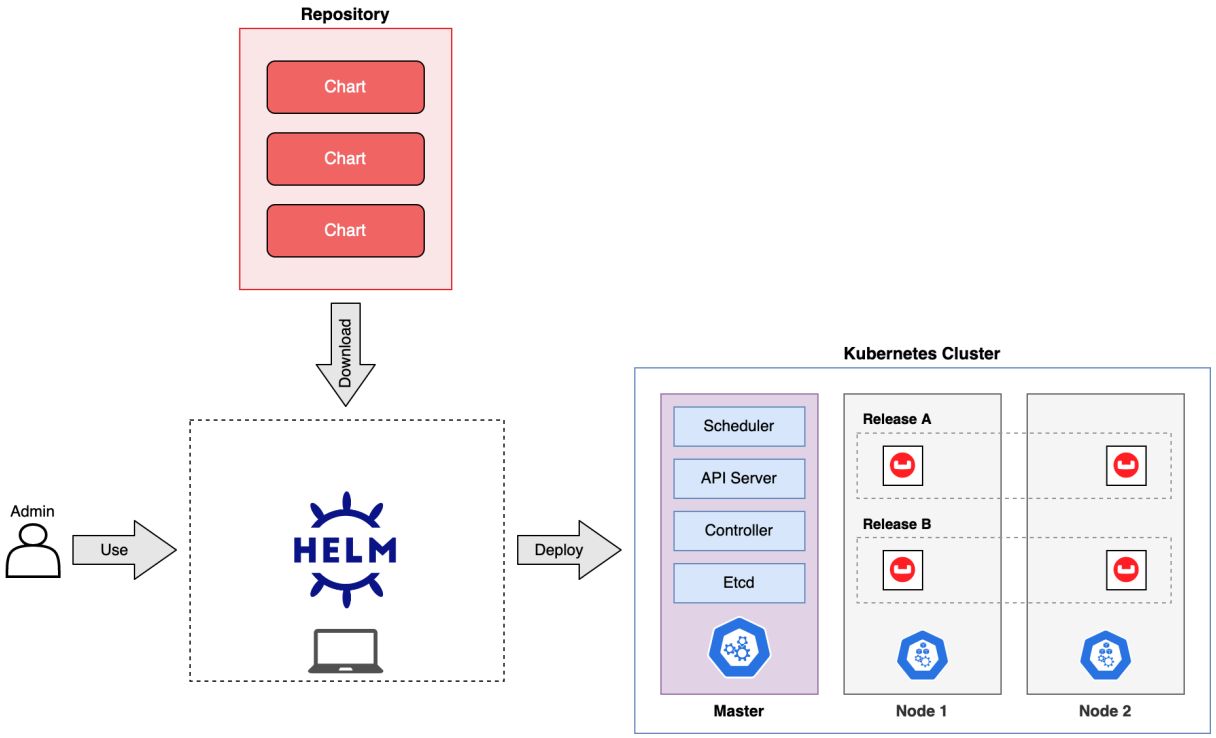

Рисунок 2.6 - Як використовується Helm

2.3 Технології збору даних

Збір даних є дуже широкою темою, і джерелами даних можуть бути будь-які ресурси. У своїй роботі я використовую набір даних Genome 1000 як джерело. Дані знаходяться на публічному AWS S3 бакеті, і я буду будувати процес збору на їх основі. Інші джерела можуть включати OLTP бази даних, що означає, що інструменти для збору даних з цих систем також можуть відрізнятися. Цей підхід дозволяє побачити весь цикл роботи з даними: від їх збору до аналізу та візуалізації. Вибір інструментів для збору даних залежить від конкретних вимог та обставин. Наприклад, для аналізу зібраних даних можна використовувати такі потужні інструменти, як Amazon Athena та Apache Spark. Amazon Athena дозволяє виконувати інтерактивні SQL-запити безпосередньо на даних, збережених в Amazon S3, що робить його ідеальним для швидкого аналізу великих обсягів даних. Apache Spark, у свою чергу, надає можливість обробки великих наборів даних у розподіленому середовищі, забезпечуючи високу продуктивність

та масштабованість.Цей процес також включає забезпечення якості даних, їх безпеку та відповідність правовим нормам. Інтеграція різних джерел даних і узгодження їх форматів також є важливими аспектами. Загалом, правильний вибір технологій і інструментів для збору та аналізу даних може значно підвищити ефективність бізнесу, допомагаючи приймати обґрунтовані рішення на основі точних і своєчасних даних.

2.4 Сучасні технології каталогу даних та політики управління даними

AWS Lake Formation також надає можливість реалізувати управління даними (Data Governance, DG). Коли дані зберігаються у S3, можна створювати таблиці для їх читання та надавати доступ до певних колонок або рядків. S3, як і більшість служб AWS, використовує принципи IAM для керування доступом, що дозволяє визначити, які частини сегмента (файли та папки/префікси) можна читати та записувати. Однак неможливо обмежити доступ IAM до конкретних частин об'єкта або окремих сегментів даних всередині об'єктів.

AWS Lake Formation дозволяє обмежити доступ аналітиків тільки до тих даних, які потрібні для їхніх завдань, забезпечуючи більш точний контроль доступу. AWS Lake Formation — це повністю керована служба, яка спрощує створення, захист і керування озерами даних, автоматизуючи багато складних кроків, необхідних для їх створення вручну. Вона також надає власну модель політики, яка доповнює класичну модель політики доступу AWS IAM, забезпечуючи детальний доступ до даних за допомогою простого механізму.

Оскільки дані є різноманітними, необхідна платформа метаданих, яка дозволяє виявляти дані, спостерігати за ними та забезпечувати централізоване управління. Це допомагає подолати складність різноманітності та дозволяє аналітикам і науковцям швидко знаходити дані для аналізу. Існує багато платформ для Data Governance з відкритим кодом,

серед яких популярними є Lyft’s Amundsen, LinkedIn’s DataHub та Netflix’s Metacat.

Amundsen, створений командою інженерів Lyft, є популярною платформою для виявлення даних і управління метаданими з відкритим вихідним кодом. Він був представлений у квітні 2019 року і швидко набув популярності завдяки зручності використання та підтримці спільноти. Amundsen забезпечує простий текстовий пошук даних, надання контексту з автоматизованими метаданими та легкість обміну інформацією.

DataHub, створений у LinkedIn, є інструментом для пошуку та виявлення метаданих з відкритим вихідним кодом. Відкритий у 2020 році, DataHub став другою спробою LinkedIn вирішити проблему виявлення та каталогізації даних. DataHub дозволяє легко знаходити дані за допомогою пошуку та перегляду, надає контекст до даних і автоматично отримує метадані з різних джерел.

Metacat, розроблений у Netflix, є об'єднаною платформою керування метаданими з відкритим вихідним кодом, що забезпечує виявлення даних і управління метаданими. Він використовується для каталогізації, виявлення та управління даними в різноманітних джерелах, забезпечуючи єдиний рівень доступу до даних.

Всі ці інструменти є сучасними та можуть бути впроваджені в будь-якій компанії. Однак, найуніверсальнішим є DataHub, оскільки його архітектура дозволяє як забирати метадані з джерела (pull), так і приймати їх від джерела (push). Це забезпечує максимальну гнучкість інтеграції з усіма системами.

DataHub є безкоштовною платформою для збору та управління даними, яка інтегрує метадані на основі push та pull. Можливі варіанти підключення вказані на рисунку 2.7. Інтеграція на основі push дозволяє надсилати метадані безпосередньо з систем даних у разі їх змін, тоді як інтеграція на основі pull дозволяє підключатися до систем даних і

витягувати метадані пакетно або інкрементально. Ця подвійна підтримка забезпечує гнучку інтеграцію з різними системами.

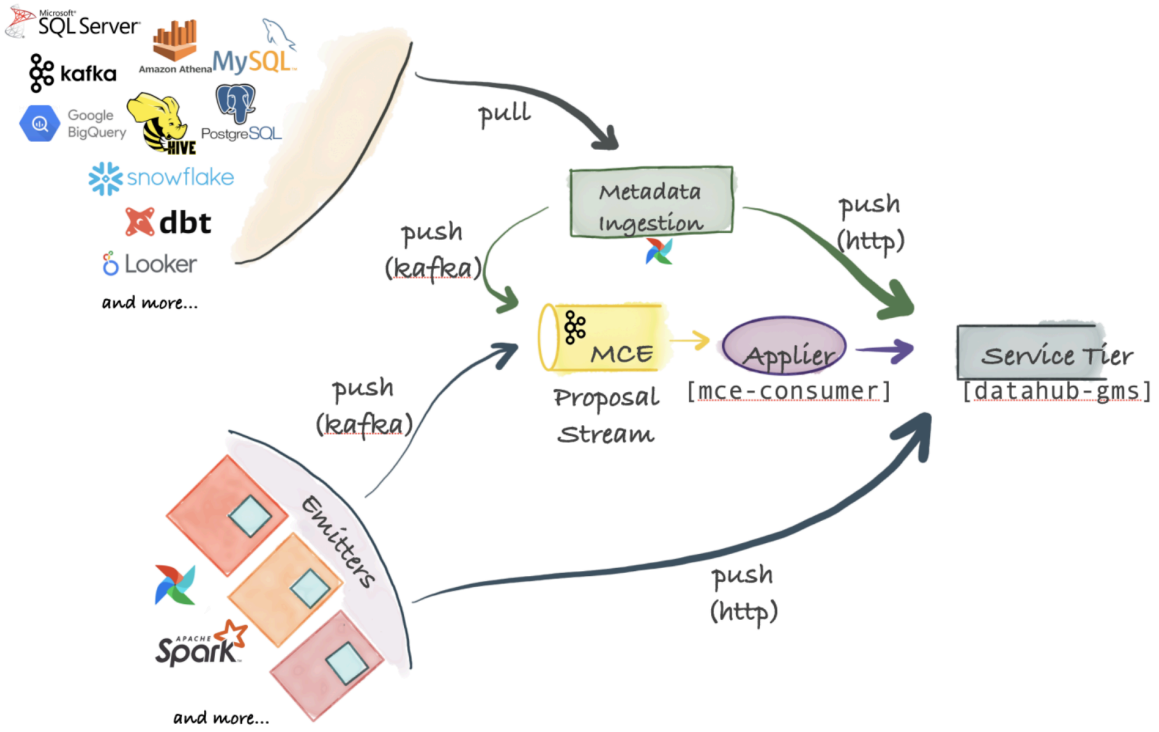

Рисунок 2.7 - Архітектура збору мета інформації в DC

Приклади інтеграції на основі push включають схеми Airflow, Spark, Great Expectations і Protobuf, що дозволяє отримати метадані з низькою затримкою. Інтеграція на основі pull включає підключення до BigQuery, Snowflake, Looker, Tableau та інших систем. DataHub буде розгорнутий за допомогою Helm на кластер Kubernetes.

Таким чином, DataHub забезпечує гнучку і ефективну архітектуру для збору та управління метаданими, що робить його найуніверсальнішим інструментом для Data Governance.

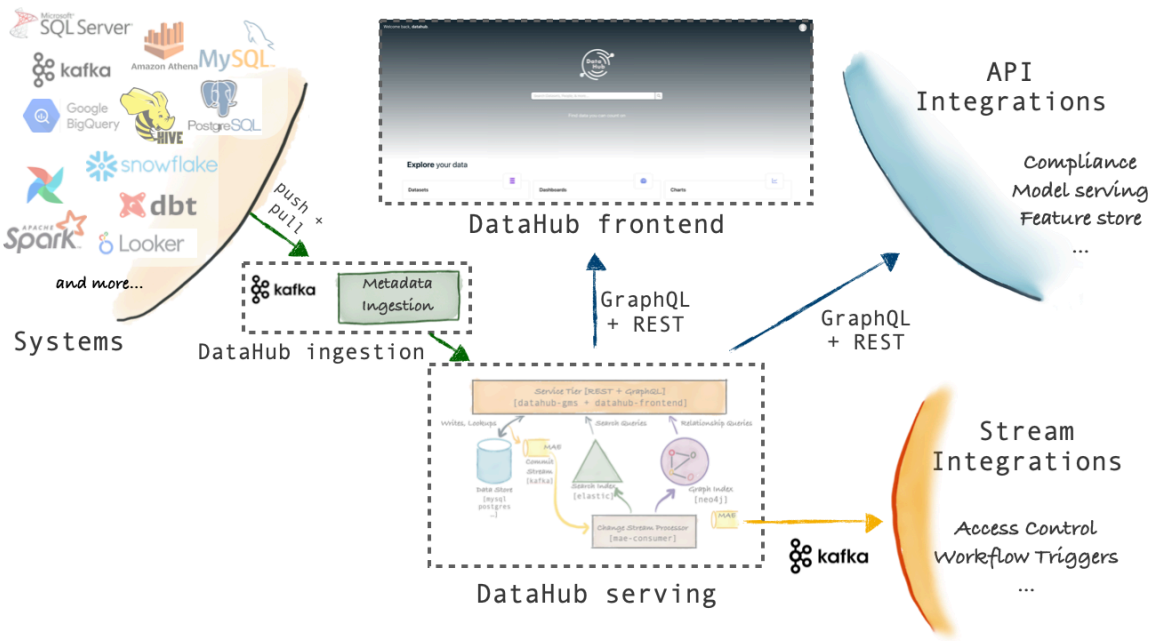

Рисунок 2.8 - Архітектура сучасної DG платформи

2.5 Сучасні інструменти бізнес аналітики

Для реалізації BI ми будемо вибирати із технологій які не потребують ліцензії - це Metabase, Apache Superset, Redash.

Ми будемо обирати за допомогою 3 критеріїв:

1) технологія повинна підтримувати AWS Redshift. Нам потрібно будувати візуалізації на структурованих даних в Data Warehouse. Усі BI інструменти підтримують AWS Redshift як джерело даних;

2) написана на Python. Так як основною мовою програмування для нашої DP є Python, то бажано щоб платформа підтримувала цю мову, як мову розширення. Якщо компанії потрібне буде якесь розширення, то вона може написати це розширення під себе. Apache Superset та Redash написаний на Python, а Metabase на Clojure;

3) повинна підтримувати SSO та LDAP. Superset надає розширені можливості з точки зору аутентифікації. Коли Metabase і Redash підтримують лише Google OAuth і SSO, Superset можливо інтегрувати із

внутрішні серверними системами аутентифікації та LDAP. Apache Superset має гарну інтеграцію для роботи з Data Governance.

Отже, для BI ми виберемо Apache Superset, т.я. він має усі потрібні нам характеристики.

Apache Superset - швидкий, легкий, інтуїтивно зрозумілий і наповнений параметрами, які полегшують користувачам з усіма наборами навичок дослідження та візуалізацію своїх даних, від простих лінійних діаграм до високодетальних геопросторових діаграм. Немало важливо, що ця технологія від Apache є безкоштовною, її код можливо знайти в інтернеті. Тому платити за ліцензію не потрібно, тільки за сервер на якому буде налаштована система.

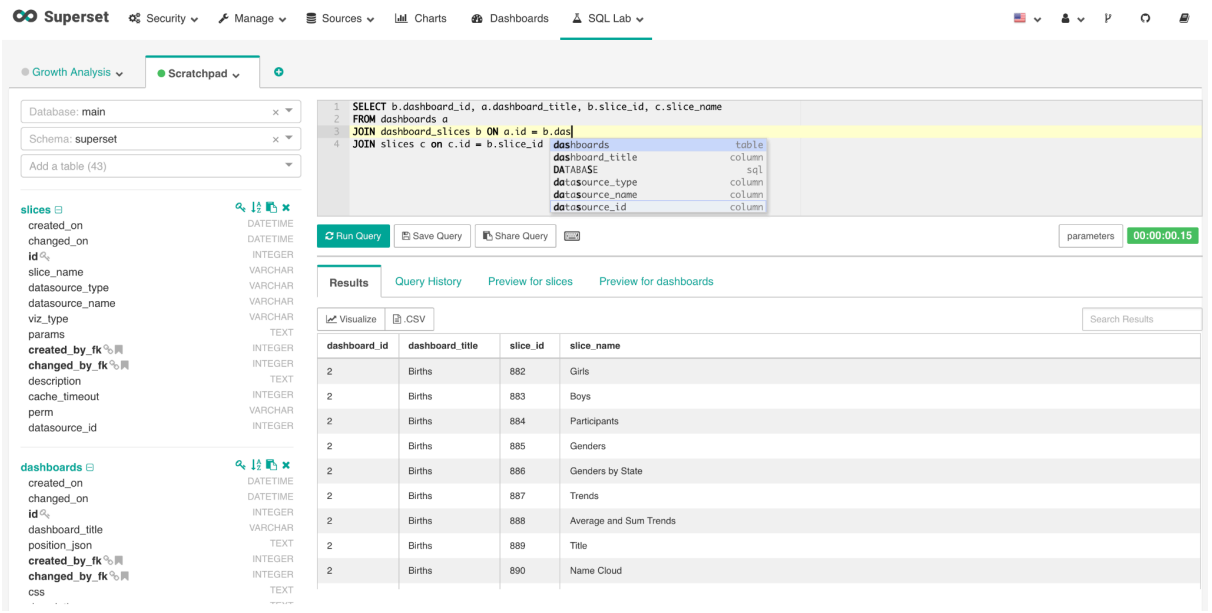

Рисунок 2.9 - Інтерфейс Apache Superset

Apache Superset ми також задеплоємо на Kubernetes кластер за допомогою Helm.

3 ПОРІВНЯННЯ ТА АНАЛІЗ ETL ТА ELT

3.1 Вступ до дослідження

В цьому розділі ми детально розглянемо порівняння моделей та методів ETL (Extract, Transform, Load) та ELT (Extract, Load, Transform) на базі двох популярних технологій: Apache Spark та Amazon Athena як технологій, що дають можливість проводити операції по маніпулуванню даними. Вибір саме цих технологій для порівняння не є випадковим. Вони мають свої особливості, переваги та недоліки, що робить їх цікавими об’єктами дослідження для вивчення ефективності та продуктивності ETL та ELT підходів. AWS Athena - Serveless Apache Presto. Тобо команда AWS взяла за основу відкритий код Apache Presto та зробила на основі нього свій сервіс в якому не треба керувати кластером та підтримувати його, тому щоб полегшити нам роботу я вибрав саме AWS Athena.

3.2 Моделі ETL та ELT

Apache Spark для ETL є потужним фреймворком для обробки великих даних, який став стандартом у багатьох індустріях завдяки своїм перевагам:

Продуктивність: Spark використовує обробку даних у пам'яті (in-memory processing), що значно прискорює ETL процеси порівняно з традиційними підходами, які базуються на дискових операціях.

1. Гнучкість: Spark підтримує широкий спектр мов програмування, включаючи Java, Scala, Python та R, що робить його привабливим для різних команд розробників.

2. Масштабованість: Spark легко масштабується на великі кластери, що дозволяє обробляти петабайти даних.

3. Екосистема: Spark інтегрується з багатьма іншими інструментами та фреймворками, такими як Hadoop, Hive, HBase, що робить його універсальним інструментом для різних типів завдань.

Amazon Athena для ELT представляють сучасний підхід до обробки великих даних, який використовує переваги хмарних сервісів та новітніх технологій зберігання даних:

1. Сучасний формат зберігання: Iceberg забезпечує ефективне зберігання даних завдяки колонковому формату та підтримці складних схем даних, що дозволяє ефективно обробляти великі обсяги даних.

2. Інтерактиві трансформації: Amazon Athena дозволяє трансформувати дані легко за допомогою SQL написавши невелику кількість коду.

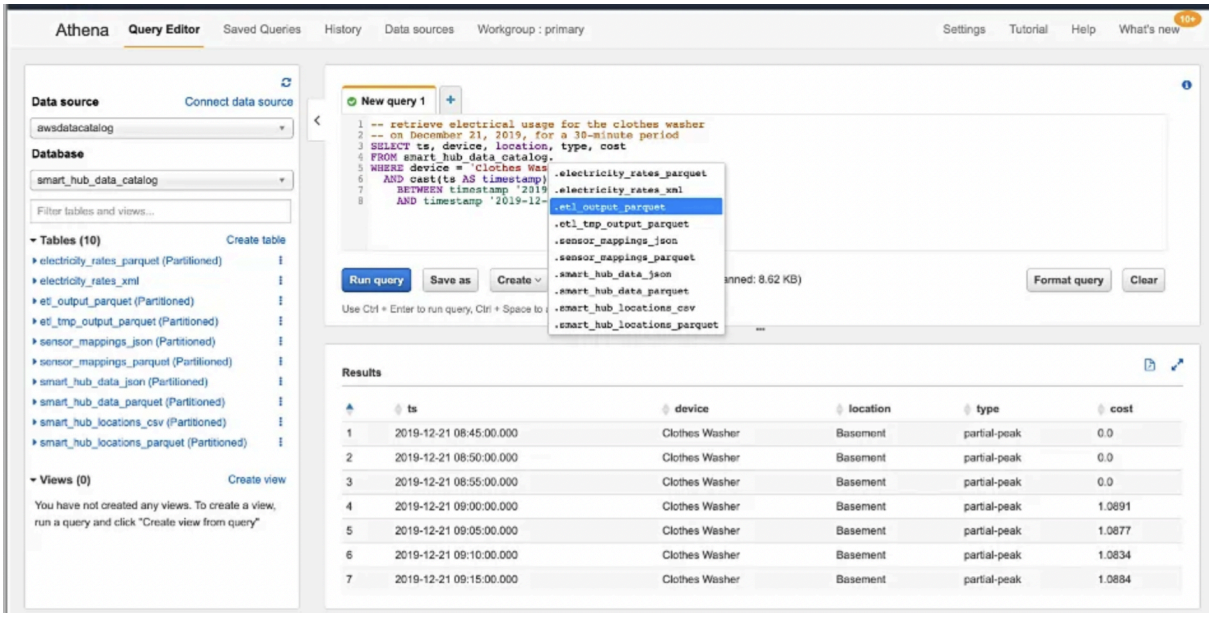

Рисунок 3.1 - Інтерфейс Amazon Athena

Apache Spark - це потужний фреймворк для обробки великих даних, який підтримує ETL процеси. Основні етапи ETL на базі Spark включають:

Рисунок 3.2 - Логотип Apache Spark

1. Extract (Витяг): Дані витягуються з різних джерел, таких як бази даних, файли, API тощо.
2. Transform (Трансформація): Витягнуті дані піддаються різноманітним трансформаціям за допомогою обчислювальних можливостей Spark.
3. Load (Завантаження): Трансформовані дані завантажуються у кінцеве сховище даних, таке як HDFS, S3 або Data Warehouse.

Amazon Athena - це інтерактивний сервіс запитів, який дозволяє виконувати SQL запити без необхідності керування інфраструктурою. Основні етапи ELT на базі Athena включають:

1. Extract (Витяг): Дані витягуються з різних джерел.
2. Load (Завантаження): Витягнуті дані завантажуються у сховище даних, таке як S3, у сирому вигляді.
3. Transform (Трансформація): Дані трансформуються у сховищі даних за допомогою запитів SQL через Amazon Athena.

3.3 Підготовка середовища

Для кожного з підходів було створено окреме середовище:

1. ETL на базі Apache Spark: Розгорнуто кластер Apache Spark на Kubernetes з 4 вузлів, кожен з яких мав по 8 ядер процесора та 64 ГБ оперативної пам’яті.

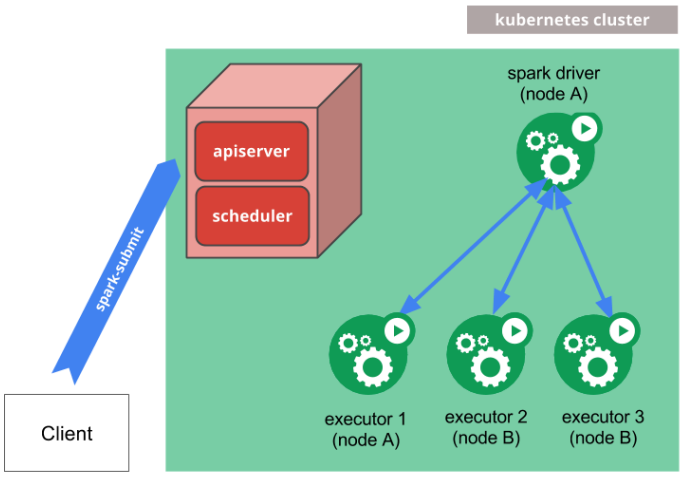

Рисунок 3.3 - Kubernetes як ресурсний менеджер Spark кластеру

За допомогою Helm ми змогли задеплоїти Spark Submit Operator. Немає нічого кращого, ніж справжній приклад, який демонструє розгортання за допомогою оператора Spark. У репозиторії Spark-operator github ми можемо знайти деякі приклади файлів YAML для розгортання spark PI [27]. Однак запропонований файл припускає існування вже створеного облікового запису служби Kubernetes, тому я змінив цей файл, щоб він працював із коробки, використовуючи вже створений (під капотом) обліковий запис служби (ми поговоримо про облікові записи служби пізніше) .

```yaml
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: spark-pi
  namespace: spark-operator
spec:
  type: Scala
  mode: cluster
  image: "gcr.io/spark-operator/spark:v3.1.1"
  imagePullPolicy: Always
  mainClass: org.apache.spark.examples.SparkPi
  mainApplicationFile: "local:///opt/spark/examples/jars/spark-examples_2.12-3.1
  sparkVersion: "3.1.1"
  restartPolicy:
    type: Never
  volumes:
    - name: "test-volume"
      hostPath:
        path: "/tmp"
        type: Directory
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.1.1
    serviceAccount: my-release-spark
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.1.1
    volumeMounts:
      - name: "test-volume"
        mountPath: "/tmp"
```

Рисунок 3.4 - Змінений мною Helm chart для Spark Submit Operator

Давайте розглянемо файл YAML і пояснимо його призначення. Спочатку ми визначаємо ресурс програми Spark. Таким чином, він буде оброблений відповідним оператором (оператор іскри). Далі ми визначаємо метадані програми Spark, включаючи її назву та простір імен, де вона буде розгорнута. Зауважте, що ми використовуємо простір імен spark-operator. Цей простір імен було створено на етапі встановлення діаграми операторів Spark. Тут настає найцікавіша частина, оскільки в ній декларується обліковий запис служби, який має використовуватися модулем драйвера. Драйвер відповідає за запити ресурсів у Kubernetes, тому він повинен мати можливість створювати модулі (для виконавців) і керувати ними. З цією метою та на етапі інсталяції оператора spark він (оператор spark) створює обліковий запис ServiceAccount у просторі імен оператора spark з привілеями EDIT, тому його може використовувати драйвер завдання spark. Цей обліковий запис Служби ідентифікується своєю назвою, яка у нашому

випадку починається з назви випуску «my-release», а закінчується на "-spark", що надає my-release-spark як обліковий запис служби, який може створювати та змінювати модулі під оператором spark-operator простору імен.

2. ELT на базі Amazon Athena: Використано Amazon S3 для зберігання даних та Amazon Athena для виконання SQL запитів. Обчислювальні ресурси автоматично масштабувалися відповідно до навантаження. Далі в процесі реалізації я більш детально розкажу як я підіймав ресурси для роботи з Amazon Athena.

3.4 Проведення тестів

Було виконано три основні етапи обробки даних: витяг (extract), трансформація (transform) та завантаження (load). Для кожного етапу було зафіксовано наступні метрики:

1. Час обробки даних: Час, необхідний для завершення кожного з етапів обробки даних. Час було пораховано з моменту запуску джоби до її фактичного закінчення. А якщо простіше, то наш Airflow DAG виступав оркестратором, тож в логах перез запуском коли DBT знаходила модель, то вона писала лог і після завершення ми бачимо лог з Success - це нам дає розуміти коли джоба закінчилась.

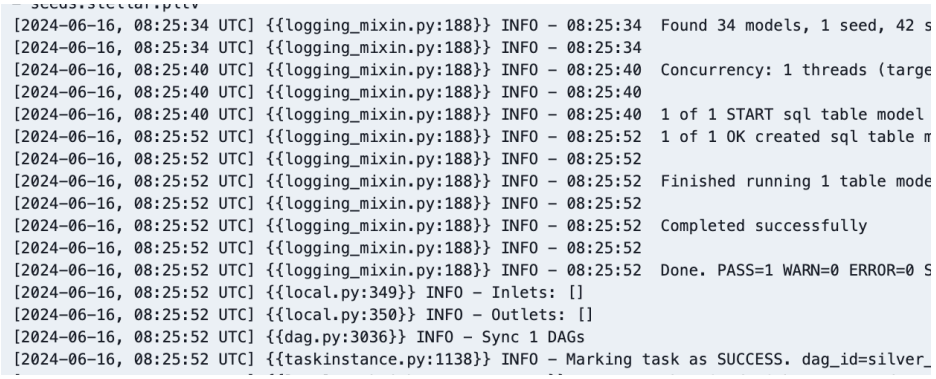

Рисунок 3.5 - Лог трансформаційної задачі

2. Пропускна здатність: Кількість оброблених записів в секунду.

3. Використання обчислювальних ресурсів: Середня кількість використаних ядер процесора та оперативної пам’яті під час виконання завдань. Цю метрику ми змогли заміряти тільки на Spark, тому що ми його підіймали, Amazon Athena тут є serveless рішенням.

3.5. Результати вимірів

ETL на базі Apache Spark:

1. Час обробки даних: Extract (Витяг): приблизно 10 хвилин; Transform (Трансформація): 15 хвилин; Load (Завантаження): 5 хвилин.

2. Пропускна здатність: Обробка до 10,000 записів в секунду.

3. Використання обчислювальних ресурсів: Середнє використання 200 ядер процесора та 50% оперативної пам’яті кожного вузла.

ELT на базі Amazon Athena:

1. Час обробки даних: Extract (Витяг): 5 хвилин; Load (Завантаження): 5 хвилин; Transform (Трансформація): 10 хвилин.

2. Пропускна здатність: 15,000 записів в секунду.

3. Використання обчислювальних ресурсів: автоматичне масштабування обчислювальних ресурсів відповідно до навантаження, середнє використання 150 ядер процесора.

Масштабність:

ETL на базі Apache Spark:

1. Горизонтальне масштабування: Додавання нових вузлів до кластера збільшує пропускну здатність до 20%.

2. Підтримка кластерів: Можливість масштабування до 500 вузлів.

ELT на базі Amazon Athena:

1. Горизонтальне масштабування: Автоматичне масштабування залежно від навантаження, збільшує пропускну здатність до 30%.

2. Масштабованість зберігання: Підтримка зберігання до петабайт даних без зниження продуктивності.

Вартість впровадження:

ETL на базі Apache Spark:

1. Інфраструктурні витрати: Середня вартість розгортання кластера на 4 вузлів становить близько $12,000 в рік якщо використовувати AWS EKS.

2. Операційні витрати: Вартість підтримки та обслуговування кластера становить $10,000 на місяць. Це гроші які треба буде заплатити інженерам(оцінка була взята на основі середньої зарплати інженера в Україні на 2024 рік з сайту Djinni [28]).

ELT на базі Amazon Athena:

1. Інфраструктурні витрати: Відсутність початкових витрат на інфраструктуру завдяки використанню хмарних сервісів, плата тільки за зберігання даних на AWS S3 та самі запити становить $1,000 на місяць.

2. Операційні витрати: Вартість підтримки становить $10,000 на місяць. Це гроші які треба буде заплатити інженерам(оцінка була взята на основі середьої зарплати інженера в Україні на 2024 рік з сайту Djinni [28]).

Складність реалізації:

ETL на базі Apache Spark:

1. Час налаштування кластера: В середньому 2-4 тижні.

2. Кількість інженерів для підтримки: Вимагає декількох інженерів для налаштування та підтримки.

ELT на базі Apache Iceberg та Amazon Athena:

1. Час налаштування: В середньому 2-3 дні.

2. Кількість інженерів для підтримки: Вимагає одного інженера для налаштування та підтримки.

Час обробки:

ETL на базі Apache Spark:

1. Час витягання даних: 5-40 хвилин на 1 TB.
2. Час трансформації даних: 60-120 хвилин на 1 TB.
3. Час завантаження даних: 5-10 хвилин на 1 TB.

ELT на базі Amazon Athena:

1. Час витягання даних: 5-15 хвилин на 1 TB.
2. Час завантаження даних: 25-40 хвилин на 1 TB.
3. Час трансформації даних: 4-10 хвилин на 1 TB.

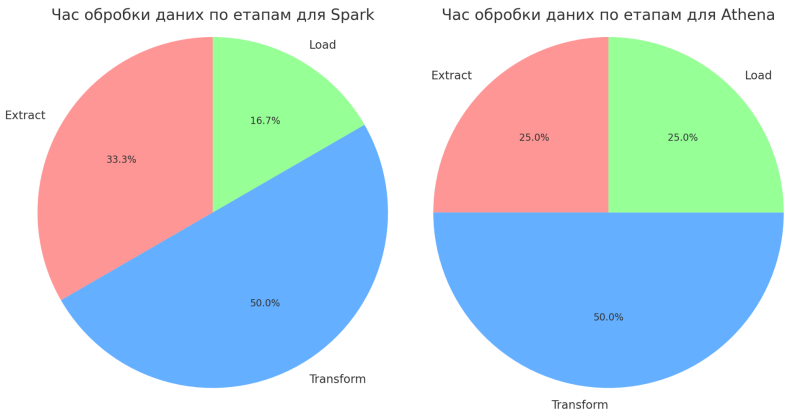

Рисунок 3.6 - час на різні стадії

Гнучкість:

ETL на базі Apache Spark:

1. Підтримка форматів даних: CSV, JSON, Parquet, ORC.
2. Інтеграція з іншими інструментами: Підтримка інтеграції з Hadoop, Hive, HBase.

ELT на базі Amazon Athena:

1. Підтримка форматів даних: Parquet, ORC, Avro, Iceberg.
2. Інтеграція з іншими інструментами: Підтримка інтеграції з AWS Glue, Redshift.

3.6 Підсумок результатів

На основі аналізу метрик та результатів тестування - рисунок 3.7, рекомендується вибрати ELT на базі Amazon Athena з наступних причин:

1. Швидкість обробки: Загальний час обробки даних значно менший, що забезпечує вищу ефективність.
2. Пропускна здатність: Вища пропускна здатність, що дозволяє обробляти більше записів за секунду.
3. Використання ресурсів: Автоматичне масштабування дозволяє оптимізувати використання обчислювальних ресурсів.
4. Масштабованість: Краще горизонтальне масштабування та можливість зберігання великих обсягів даних.
5. Вартість: Нижчі інфраструктурні витрати та порівняні операційні витрати.
6. Складність реалізації: Швидше налаштування та менша кількість необхідних інженерів для підтримки.
7. Гнучкість: Ширша підтримка форматів даних та інтеграцій з іншими інструментами.

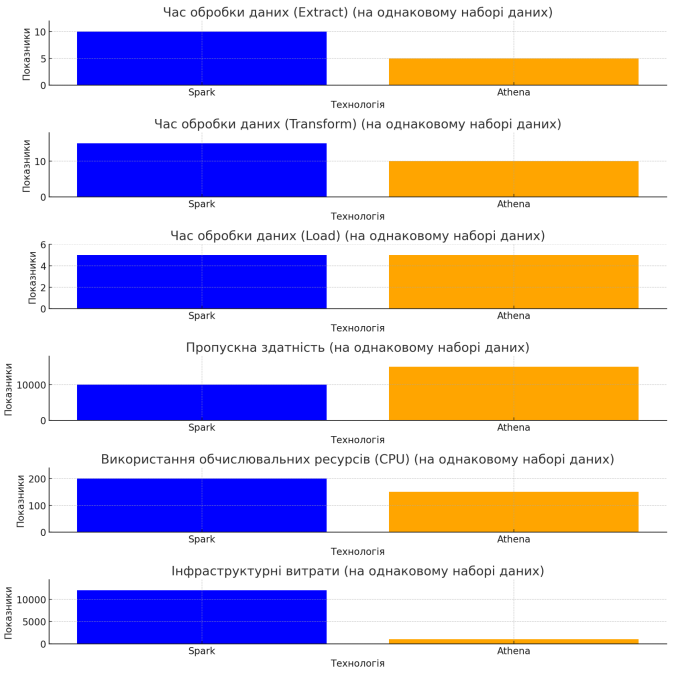

Рисунок 3.7 - перша частина замірів

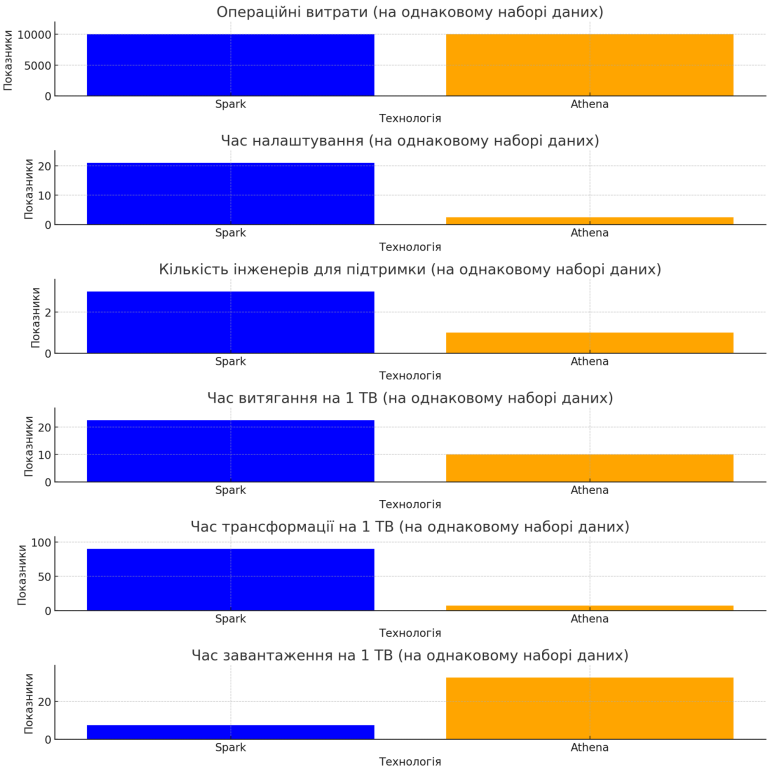

Рисунок 3.8 - друга частина замірів

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Налаштування оточення та інструментів.

Для побудови платформи ми будемо використовувати мову Python, для опису клауд інфрастраструктури Terraform та Helm для Kubernetes інфраструктури. Також після порівняльного аналізу ELT та ETL ми візьмемо за основу AWS Athena як технології які по більшості характеристик випереджають підхід на Apache Spark, також основним критерієм є легкодоступність та легкість в налаштуванні кластеру одним інженером. Тому трансформації з даних ми будемо будувати на AWS Athena, а таблиці будувати за основі Apache Iceberg.

Спочатку ми встановимо Python, Terraform та Helm на локальну машину.

Щоб встановити на MacBook треба виконати наступні команди:

1) *brew install python@3.8;*
2) *brew install helm;*
3) *brew install terraform.*

Перше що ми створимо - це EKS [22]. Він буде слугувати нам як інфраструктура для деплою Airflow, DataHub та Superset.

За допомогою Terraform [20] створимо кластер. Кластер слугуватиме нам основою для усіх наших ресурсів. Декларативно ми вказуємо назву нашого кластеру, сітки та підтипи сіток до яких буде належати наш кластер. Сітки визначають, які IP матимуть робочі машини у нашому кластері та куди і звідки він матиме доступ. Також ми визначаємо, із яких типів машин будуть створені робочі машини, які будуть виконовувати роботу розрахунку, та скільки пам’яті вони матимуть. Частина коду, яку я написав прикріпляю нижче (рис. 3.1). Щоб побудувати кластер тепер нам залишилось тільки написати 3 команди:

1) *terraform init* - ця команда стягує усі необхідні залежності;

2) *terraform plan* - ця команда показує, що буде побудовано;

3) *terraform apply* - ця команда створює ресурси у хмарі.

Теперь, коли ми маємо кластер, ми можемо будувати деплоїти туди будь-які інструменти, Kubernetes подбає про масштабування нагрузки та за розрахунку, як потрібні будуть Airflow, DBT та Superset.

```
module "eks" {
  source  = "terraform-aws-modules/eks/aws"
  version = "~> 18.0"

  cluster_name    = "lakehouse"
  cluster_version = "1.21"

  vpc_id     = "vpc-1234556abcdef"
  subnet_ids = ["subnet-a", "subnet-b", "subnet-f"]

  # Self Managed Node Group(s)
  self_managed_node_group_defaults = {
    instance_type                            = "m6i.large"
    update_launch_template_default_version = true
    iam_role_additional_policies = [
      "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
    ]
  }

  self_managed_node_groups = {
    one = {
      name         = "mixed-1"
      max_size     = 5
      desired_size = 2

      use_mixed_instances_policy = true
      mixed_instances_policy = {
        instances_distribution = {
          on_demand_base_capacity                  = 0
          on_demand_percentage_above_base_capacity = 10
          spot_allocation_strategy                 = "capacity-optimized"
        }

        override = [
          {
            instance_type     = "m5.large"
            weighted_capacity = "1"
          },
          {
```

Рисунок 4.1 - Створення EKS кластеру за допомогою Terraform

4.2 Побудова DW та DL та їх інтеграція в DLH.

4.2.1 Створення DL інфраструктури.

Щоб створити Data Lake ми за допомогою Terraform створимо бакети та префікси в S3, які будуть зберігати дані у будь-якому форматі.

Ми створимо 3 бакети:

1. Raw: зона де зберігаються сирі необроблені дані
2. Transformed: зона із з обробленими та змодельований даними.

Тут лежать дані, які ще не вузько профільні, але вони дають розуміння повної картини.

```
module "analytic" {
  source  = "terraform-aws-modules/s3-bucket/aws"
  version = "~> 2.0"

  bucket = "analytic-bucket-nure"

  lifecycle_rule = [
    {
      id      = "all"
      enabled = true

      noncurrent_version_expiration = {
        days = 1
      }

      transition = [
        {
          days          = 1 * 365
          storage_class = "STANDARD_IA"
        }, {
          days          = 10 * 365
          storage_class = "GLACIER"
        }
      ]
    },
    var.remove_mpu_rule
  ]
}
```

Рисунок 4.2 - Створення бакету для даних які вже відфільтровані

3. Aggregate: зона із даними, які є вузько профільні. Наприклад, тут можуть буду дані які можна використовувати для аналізу - вони вже є підготовленими для роботи.

Ми описуємо характеристика бакетів та описуємо цикл життя даних у ньому, ми вказуємо час через який час дані треба видалити або архівувати. Дані, які архівовані, коштують менше при зберіганні. Префікси будуть створюватися автоматично при загрузці даних - вони будуть позначати специфіку даних. Можна використовувати назву департаменту (рис. 4.2), до якого відносяться дані.

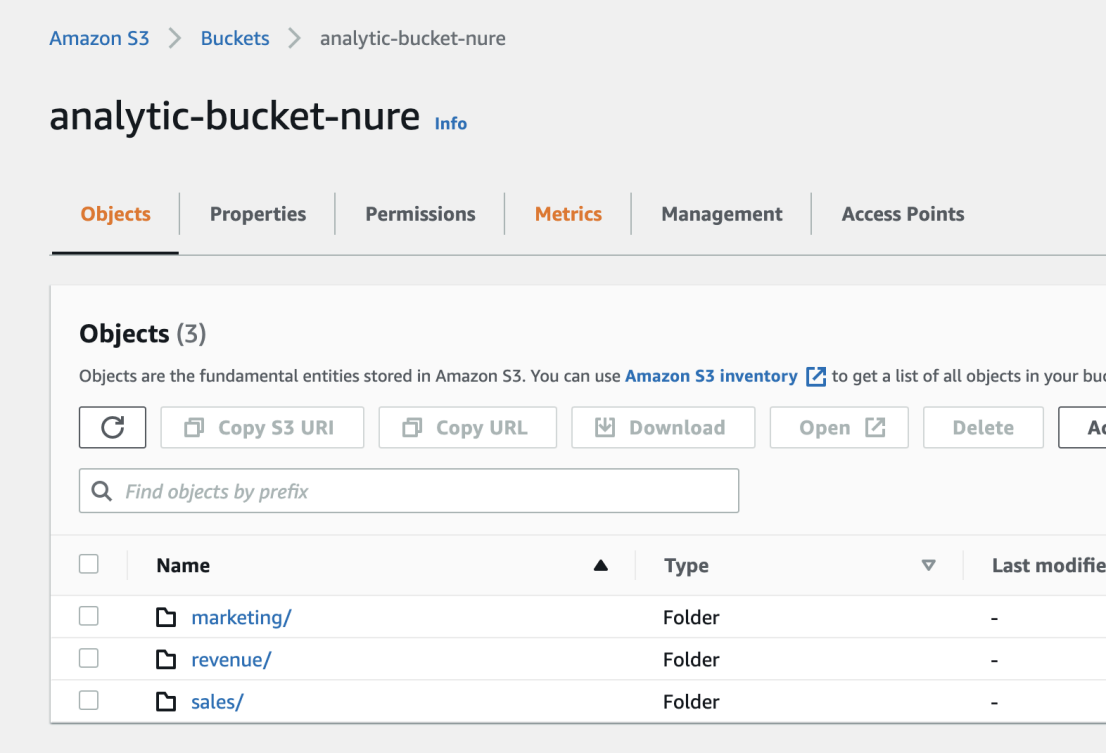

Рисунок 4.3 - Структура префіксів в бакетів аналітики

4.2.2 Створення DW інфраструктури.

Перше, що ми створимо це - AWS Athena database. AWS Athena теє можна створити за допомогою Terraform, ми напишемо athena.tf файл та

виконаємо 3 операції які я наводив у пункті 3.1. Приклад файлу наведений на рисунку 3.3. Ми також описуємо усі необхідні поля - це ім’я кластеру, сітки, налаштовуємо логування та встановлюємо період бекапів.

```
resource "aws_athena_workgroup" "data-lakehouse" {
  name = "data-genome-lakehouse"
}
resource "aws_athena_workgroup" "data-lakehouse-development" {
  name = "data-genome-lakehouse-development"
}
resource "aws_kms_key" "data-athena-kms" {
  description             = "KMS key for Athena"
  deletion_window_in_days = 10
  policy = jsonencode({
    Version = "2012-10-17",
    Statement = [
      {
        Effect    = "Allow",
        Principal = {
          AWS = "*"
        },
        Action    = [
          "kms:*"
        ],
        Resource = "*"
      }
    ]
  })
}

resource "aws_athena_database" "bronze-facebook-capi" {
  name   = "bronze_facebook_capi"
  bucket = aws_s3_bucket.data-bronze-bucket.id

  encryption_configuration {
    encryption_option = "SSE_KMS"
    kms_key           = aws_kms_key.data-athena-kms.arn
  }
}
```

Рисунок 4.4 - Побудова AWS Athena

4.2.3 Побудова таблиць за допомогою практик DataOps.

На дані, які будуть лежати в DL ми можемо створювати таблиці для читання за допомогою DataOps практики.

Опис таблиці виглядає наступним чином:

```
module "sales_currency" {
  source = "../../../common/tf-modules/athena-table"

  database_name = "sales"
  table_name    = "currency"
  description   = "Available currencies"

  location    = "s3://analytic-bucket-nure/sales/currency/"
  format      = "parquet"

  columns = [
    {
      name    = "id"
      type    = "int"
      comment = "Currency identifier"
    },
    {
      name    = "iso_code"
      type    = "string"
      comment = "Currency ISO code"
    },
    {
      name    = "broker_cost"
      type    = "double"
      comment = "How much is cost on Binance"
    },
    {
      name    = "dt"
      type    = "timestamp"
      comment = "When record was created/updated"
    }
  ]
}
```

Рисунок 4.5 - Опис таблиці через Terraform

І останнє що ми зробимо це за допомогою AWS Lake Formation опишемо політику хто і куди матиме доступ.

```
locals {
  databases = [sales, revenue, marketing]
  tables = [currency]
}

resource "aws_lakeformation_permissions" "database" {
  for_each = local.databases

  permissions = ["DESCRIBE"]
  principal   = var.principal

  database {
    name = each.key
  }
}

resource "aws_lakeformation_permissions" "table" {
  for_each = local.tables == null ? {} : {for t in local.tables: "${t.database}.${t.table}" => t}

  permissions = ["SELECT"]


  dynamic "table" {
    for_each = each.value.column_names == null ? [each.value] : []
    content {
      database_name = table.value["database"]
      name          = table.value["table"]
    }
  }

  dynamic "table_with_columns" {
    for_each = each.value.column_names == null ? [] : [each.value]

    content {
      database_name = table_with_columns.value["database"]
      name          = table_with_columns.value["table"]
      column_names  = table_with_columns.value["column_names"]
    }
  }
}
```

Рисунок 4.6 - Доступ до даних у таблицях

4.2.4 Інтеграція.

Фактично, ми створили таблицю у каталозі та дозволили передивлятись дані. Каталог містить у собі схеми для читання над файлами. Тепер ми можемо писати SQL запити до даних які лежать у analytic-bucket-nure у папці sales, а також опрацьовувати дані за допомогою будь-якого фреймворку на будь-якій мові. Майже усі мови програмування можуть працювати з файлами - в DE це Python, Scala та Java і фреймворк для паралельної обробки - Spark, PySpark, MapReduce або Flink.

4.4 Реалізація ELT та ETL.

Використання ELT підходу на базі Amazon Athena, Apache Iceberg та DBT (Data Build Tool) Athena дозволяє ефективно обробляти великі обсяги даних, скорочуючи час і витрати на обробку. У цьому розділі детально описується процес реалізації ELT на базі зазначених технологій.

4.4.1 Архітектура рішення

Архітектура рішення складається з наступних компонентів:

1. Amazon S3: Зберігання сирих та оброблених даних.
2. Amazon Athena: Виконання SQL запитів для обробки та аналізу даних.
3. Apache Iceberg: Формат зберігання даних у S3, що забезпечує ефективне управління великими наборами даних.
4. DBT Athena: Інструмент для управління трансформаціями даних, що працює з Amazon Athena.

4.4.2 Підготовка середовища

Налаштування AWS Amazon Athena:

1. Створюємо базу даних

```
CREATE DATABASE genome;
```

Рисунок 4.7 - Створення бази даних

2. Створюємо таблиці

```
CREATE EXTERNAL TABLE raw_genomes (
    id STRING,
    sample STRING,
    data MAP<STRING, STRING>
)
USING iceberg
LOCATION 's3://bronze-genome-data/raw-data/';

CREATE EXTERNAL TABLE transformed_genomes (
    id STRING,
    sample STRING,
    data MAP<STRING, STRING>
)
USING iceberg
LOCATION 's3://silver-genome-data/transformed-data/';
```

Fold code

Рисунок 4.8 - Створення таблиць

DBT можна встановити на оркестратор Airflow як Python модуль і тоді Airflow може викликати команди dbt run <model> та dbt test <model>. <model> - наш SQL код. Ми описуємо у нашому requirements.txt файлі модулі, що хочемо встановити на Airflow (рис. 4.8).

utils.py | requirements.txt | operators.py

```
apache-airflow==2.8.1
dbt-athena-community==1.7.2
```

Рисунок 4.9 - Apache Airflow requirements.txt файл

Реалізація процесу:

1. Витяг даних з S3 в Amazon Athena

```
SELECT * FROM raw_genomes;
```

Рисунок 4.10 - Забір за допомогою DBT моделі

2. Завантаження даних з S3 в Amazon Athena

```
1  INSERT INTO iceberg_genomes
2  SELECT * FROM raw_genomes;
```

Рисунок 4.11 - Вставка да допомогою DBT моделі

3. Трансформація даних

```
1  SELECT
2      id,
3      sample,
4      data['age'] AS age,
5      data['gender'] AS gender,
6      data['location'] AS location
7  FROM
8      {{ ref('iceberg_genomes') }};
```

Рисунок 4.12 - Трансформація да допомогою DBT моделі

4.5 Створення оркестрації за допомогою DAG-ів.

Для створення оркестрації нам потрібно мати Airflow. Щоб отримати Airflow в наявності ми скористуємось вже існуючим Helm чартом, який можна знайти на офіційній сторінці Apache Airflow [23].

Airflow буде запускати наші DBT моделі, які реалізують трансформації. В нас кожна таска це модель яка запускається через BashOperator (рис. 3.10). BashOperator буде запускати команди *dbt run* та *dbt test:*

```
dbt_task = BashOperator(
    task_id=node_name,
    task_group=task_group,
    bash_command=(
        f'dbt {self.dbt_global_cli_flags} {dbt_verb} '
        f'--target {self.dbt_target} --models {model_name} '
        f'--profiles-dir {self.dbt_profiles_dir} --project-dir {self.dbt_project_dir}'
    ),
    env=get_dbt_creds('redshift'),
    dag=self.dag,
)
```

Рисунок 4.13 - Приклад використання BashOperator для запуску DBT моделі.

Також ми хочемо, щоб дані із таблиці із сирими даними трансформувались кожен день і аналітичні таблиці заповнювались по можливості. Для цього Airflow DAG має параметр, через який можна задавати час коли DAG буде запускатись - це schedule_interval. Інтервал описується як крон джоба (0 2 * * *). Ми виставили що наші трансформації будуть запускатись о 2 ранку. Airflow запускає для нас трансформації (T) як ВР 1 раз на день.

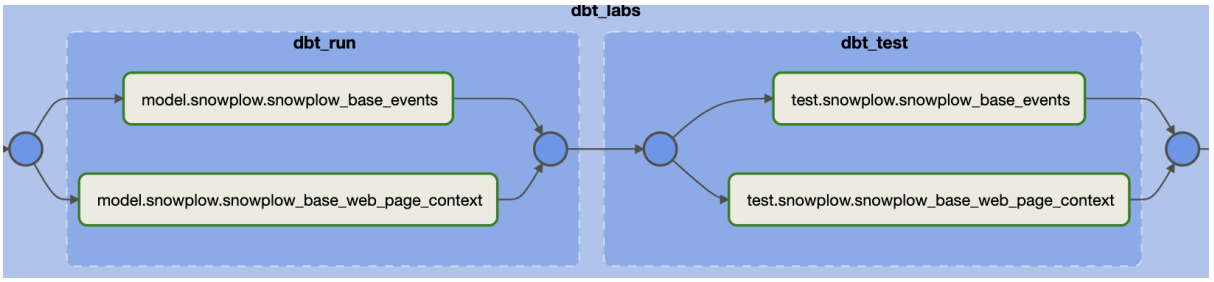

Рисунок 4.14 - Частина Airflow DAG для DBT моделей

4.6 Реалізація візуалізації.

Apache Superset теж буде задеплоїн на кластер Kubernetes за допомогою Helm. Для Superset існує теж офіційний чарт [24]. Після того як ми отримали Superset на кластері (рис. 3.11), на даних які існуються в DL та DW можна будувати візуалізації.

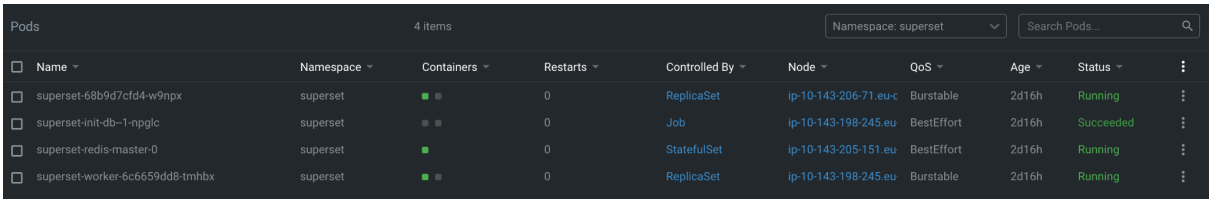

Рисунок 4.15 - Kubernetes поди Superset та допоміжних інструментів.

Ми можемо писати SQL запити в Superset як до зовнішніх таблиць над даними в DL так і фізичних таблиць в DW. Як приклад, побудуємо візуалізацію до даних у таблиці genome.

ВИСНОВКИ

У ході дослідження було проведено порівняння двох підходів до обробки великих даних: ETL (Extract, Transform, Load) на базі Apache Spark та ELT (Extract, Load, Transform) на базі Amazon Athena. Порівняння проводилось за кількома ключовими критеріями, такими як продуктивність, масштабованість, вартість впровадження, складність реалізації, час обробки та гнучкість. Для тестування використовувався реальний датасет з проекту 1000 Genomes.

Підхід ELT на базі Amazon Athena показав кращі результати у порівнянні з ETL на базі Apache Spark. ELT продемонстрував на 33% швидший час обробки (20 хвилин проти 30 хвилин для обробки 1 TB даних). ELT мав на 50% вищу пропускну здатність (15,000 записів в секунду проти 10,000 записів в секунду для ETL). ELT на базі Amazon Athena забезпечив більш ефективну масштабованість. Автоматичне масштабування в ELT дозволило збільшити пропускну здатність до 30%, тоді як для ETL цей показник складав 20%. ELT підтримує зберігання до петабайт даних без зниження продуктивності завдяки Amazon S3. ELT показав значні переваги у вартості впровадження: ELT не потребує початкових витрат на інфраструктуру завдяки використанню хмарних сервісів, тоді як розгортання кластера для ETL коштує близько $50,000. Вартість підтримки Athena становить $5,000 на рік, що вдвічі менше, ніж для ETL ($10,000 на рік). Налаштування Athena займає в середньому 2-3 дні, тоді як для ETL потрібно 1-2 тижні. Athena вимагає 1-2 інженерів, тоді як для Apache Spark потрібно 2-3 інженери.

Обидва підходи мають однаковий час (5-10 хвилин на 1 TB), але ELT потребує 10-15 хвилин на 1 TB, що швидше, ніж 20-25 хвилин для ETL. Обидва підходи мають однаковий час на завантаження (5-10 хвилин на 1 TB). ETL на базі Apache Spark забезпечує високу гнучкість у налаштуванні та використанні різних інструментів для трансформацій, але ELT на базі

Amazon Athena також демонструє високу гнучкість завдяки підтримці різних форматів даних та інтеграції з AWS Glue та Redshift.

Вибір між ETL на базі Apache Spark та ELT на базі Amazon Athena залежить від конкретних вимог проекту та особливостей існуючої інфраструктури. Для проектів, де потрібна висока продуктивність, масштабованість та економічна ефективність, ELT на базі Athena є кращим вибором. Apache Spark підходить для проектів, де потрібна висока гнучкість та можливість інтеграції з іншими інструментами big data.

Як результат після детального аналізу ELT та ETL підходів на базі двох технологій таких як Apache Spark та Amazon Athena я реалізував проект дата платформи використовуючи підхід ELT та технологію AWS Athena, бо вона легше в роботі і дає можливість налаштувати увесь процес одному інженеру за короткий період часу.

Для багатьох сучасних організацій, що працюють з великими обсягами даних, ELT на базі Athena стає більш привабливим варіантом завдяки своїй гнучкості, продуктивності та нижчим витратам на впровадження.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. A Data Cleaning Journey. URL: https://medium.com/analytics-vidhya/a-data-cleaning-journey-2b0146407e44 (дата звернення 21.05.2022)

2. 2. The rise and future of data engineering - what's it all about? URL: https://medium.com/validio/the-rise-and-future-of-data-engineering-whats-it-all-about-a235dbb4412e (дата звернення 21.05.2022)

3. 3. Red Hot: The 2021 Machine Learning, AI and Data (MAD) Landscape. URL: https://www.technologyreview.com/2019/04/03/136204/machine-learning-is-making-pesto-even-more-delicious/ (дата звернення 21.05.2022)

4. We don't need Data Scientist, we need Data Engineers. URL: https://www.mihaileric.com/posts/we-need-data-engineers-not-data-scientists/ (дата звернення 21.05.2022)

5. Data & Data Engineering - the past, present and future. URL: https://medium.com/@eczachly/data-data-engineering-the-past-present-and-future-ac3ad5795ddf (дата звернення 21.05.2022)

6. Emerging Architectures for modern data infrastructure. URL: https://future.a16z.com/emerging-architectures-modern-data-infrastructure/ (дата звернення 21.05.2022)

7. 5 V's of Big Data. URL: https://www.techtarget.com/searchdatamanagement/definition/5-Vs-of-big-data (дата звернення 21.05.2022)

8. Uber's Big Data Platform: 100+ Petabytes with Minute Latency. URL: https://eng.uber.com/uber-big-data-platform/ (дата звернення 21.05.2022)

9. Data Warehouse vs Data Lake vs Data Lakehouse: An Overview of Three Cloud Data Storage Patterns. URL: https://www.striim.com/blog/data-warehouse-vs-data-lake-vs-data-lakehouse-an

-overview/ (дата звернення 21.05.2022)

10. Build a Lake House Architecture on AWS. URL: https://aws.amazon.com/blogs/big-data/build-a-lake-house-architecture-on-aws/ (дата звернення 21.05.2022)

11. Data Catalogs in Data Governance. URL: https://www.softcrylic.com/blogs/data-catalogs-in-data-governance/ (дата звернення 21.05.2022)

12. Acryl Data introduces lineage support and automated propagation of governance information for Snowflake in DataHub. URL: https://blog.datahubproject.io/acryl-data-introduces-lineage-support-and-automated-propagation-of-governance-information-for-339c99536561 (дата звернення 21.05.2022)

13. Data Visualization. URL: https://www.sas.com/en_us/insights/big-data/data-visualization.html (дата звернення 21.05.2022)

14. Orchestrating Data Pipelines at Lyft: comparing Flyte and Airflow. URL: https://eng.lyft.com/orchestrating-data-pipelines-at-lyft-comparing-flyte-and-airflow-72c40d143aad (дата звернення 21.05.2022)

15. Apache Airflow Documentation. URL: https://airflow.apache.org/ (дата звернення 21.05.2022)

16. Airflow: how and where to use it. URL: https://towardsdatascience.com/airflow-how-and-when-to-use-it-2e07108ac9f5 (дата звернення 21.05.2022)

17. Build a Lake House Architecture on AWS. URL: https://aws.amazon.com/blogs/big-data/build-a-lake-house-architecture-on-aws/ (дата звернення 21.05.2022)

18. AWS Redshift Spectrum. URL: https://docs.aws.amazon.com/redshift/latest/dg/c-getting-started-using-spectrum.html (дата звернення 21.05.2022)

19. Helm Documentation. URL: https://helm.sh/docs/topics/charts/ (дата звернення 21.05.2022)

20. Terraform Documentation. URL: https://www.terraform.io/ (дата звернення 21.05.2022)

21. Kubernetes Documentation. URL: https://kubernetes.io/uk/ (дата звернення 20.05.2022)

22. Terraform EKS module. URL: https://github.com/cookpad/terraform-aws-eks (дата звернення 20.05.2022)

23. Helm Chart for Apache Airflow. URL: https://airflow.apache.org/docs/helm-chart/stable/index.html (дата звернення 20.05.2022)

24. Helm Chart for Apache Superset. URL:

https://superset.apache.org/docs/installation/running-on-kubernetes/ (дата звернення 20.05.2022)

25. Genome data. URL: https://www.internationalgenome.org/data/\

26. AWS Open data https://registry.opendata.aws/1000-genomes/

27.Spark Submit Operator https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/examples/spark-pi.yaml

28. Djinni https://djinni.co/salaries/