КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА АВТОМАТИЗОВАНОЇ ELT СИСТЕМИ ДЛЯ АНАЛІТИКИ ВЕЛИКИХ ДАНИХ В РЕАЛЬНОМУ ЧАСІ

Виконав студент 4-го курсу
Микита ОЛЕКСІЄНКО _____
(підпис)

Науковий керівник:
асистент, кандидат фіз.-мат. наук
Костянтин ЖЕРЕБ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
« __ » _____ 2023 р., протокол № __
Завідувач кафедри
Олександр ПРОВОТАР _____
(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 48 сторінок , 17 ілюстрацій, 42 джерела посилання.

AWS, BIG DATA, CI/CD, CLOUDFORMATION, ELT, PIPELINE, SPARK, АВТОМАТИЗАЦІЯ РОЗГОРТАННЯ, АНАЛІТИКА В РЕАЛЬНОМУ ЧАСІ, КЛАСТЕР, ХМАРНІ ОБЧИСЛЕННЯ.

Мета роботи: розробка автоматизованого та масштабованого ELT pipeline-у для обробки та аналітики великих даних в реальному часі та демонстрація його роботоспроможності на штучно створеному прикладі.

Об’єкт роботи: Об’єктом розробки є система з типовою ELT архітектурою, здатна до обробки та аналітики великих даних в реальному часі та буде підтримувати автоматичне розгортання всіх її компонентів.

Інструменти розробки: розподілений рушій Spark, розподілена система потокової передачі даних Kafka, хмарні сервіси AWS, сховище даних Amazon S3, кластерна платформа Amazon EMR, аналітичний застосунок Metabase, сервіс шаблонів AWS CloudFormation, CI/CD pipeline-и GitHub Actions, контейнерна платформа Docker, мова програмування Python, бібліотека boto3, середовище розробки PyCharm,.

Результати роботи: Розроблено типову ELT систему для обробки та аналітики великих даних в реальному часі з підтримкою автоматизації розгортання всіх її складових та перевірено її придатність на штучно створеному прикладі.

Створений програмний продукт може бути адаптований під схожий ELT процес обробки та аналітики даних або використаний в навчальних цілях для ознайомлення з дистрибутивним обчислювальним рушієм Spark, ELT парадигмою та сервісом хмарних обчислень AWS. Система може бути покращена за допомогою інтеграції оркестратора для автоматичного запуску Spark задач.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

AWS – Amazon Web Services, платформа хмарних сервісів від Amazon;

API – Application Programming Interface, прикладний програмний інтерфейс;

BI – Business Intelligence, інструмент для проведення бізнес аналітики

CI/CD – Continuous Integration/Continuous Delivery, набір методик, що дозволяє розробникам частіше і надійніше розгортати зміни в програмному забезпеченні;

EC2 – Elastic Compute Cloud, сервіс AWS, що надає в оренду віртуальні машини та обчислювальні потужності;

ELT – Extract, Load, Transform, абревіатура для позначення процесу, який займається витягом, завантаженням та перетворенням даних;

ETL – Extract, Transform, Load, абревіатура для позначення процесу, який займається витягом, перетворенням та завантаженням даних;

EMR – Amazon Elastic MapReduce, керована кластерна платформа Amazon;

IaC – Infrastructure as Code, Інфраструктура як код;

IDE – Integrated Development Environment, інтегроване середовище розробки;

JAR – Java ARchive, архів виконуваної Java програми;

JSON – JavaScript Object Notation, формат обміну даними;

NYC – New York City, місто Нью-Йорк;

RDS – Relational database service, сервіс AWS для роботи з реляційними базами даних у хмарі;

S3 – Amazon Simple Storage Service, сховище даних Amazon;

SQL – Structured Query Language, мова структурованих запитів для роботи з реляційними базами даних;

YARN – Yet Another Resource Negotiator, менеджер ресурсів у Hadoop 2 і 3.

ВСТУП

Оцінка сучасного стану об’єкта дослідження або розробки. Сьогодні ми живемо в світі, де щосекунди наші смартфони, ноутбуки, комп’ютери та інші пристрої генерують неймовірні за кількістю обсяги даних. Для обробки цих даних та проведення над ними аналізу в режимі реального часу, з метою отримання цінної інформації, потрібні системи, архітектура та логіка яких здатні забезпечити запит на такого ж неймовірного розміру датацентри та обчислювальні потужності, підтримка та масштабованість яких можуть бути складними задачами, як з технічної так і з фінансової точки зору. В цій ситуації на допомогу приходять ELT pipeline-и, Big Data та хмарні обчислення.

Актуальність роботи та підстави для її виконання. ELT парадигма побудови систем достатньо молода відносно своєї попередниці ETL і стрімко розвивається. Із стрімким збільшенням розмірів оброблюваних даних та потребою в аналізі отриманої інформації в режимі реального часу ELT архітектури активно застосовуються, часто в комбінації з хмарними сервісами, що надають масштабованість, підтримку та відмовостійкість системи.

Мета й завдання роботи. Метою кваліфікаційної роботи є розробка типової ELT системи для обробки та аналітики великих даних в реальному часі з підтримкою автоматизації розгортання всіх її складових та перевірити її придатність на штучно створеному прикладі.

Завданням кваліфікаційної роботи було розробити набір інфраструктурних компонентів, які разом утворюють ELT pipeline для аналітики великих даних в реальному часі, та реалізувати механізм автоматичного розгортання цих компонентів. Також була поставлена задача протестувати здатність розробленої системи до обробки даних в реальному часі на штучно створеному прикладі.

Об’єкт, методи й засоби дослідження та розробки. Об’єктом розробки є автоматизований ELT pipeline з підтримкою аналітики даних в реальному часі.

Роль потокової системи передачі даних виконує Apache Kafka. Функцію сховища даних виконує Amazon S3. Роль кластерної платформи з обчислювальними ресурсами була покладена на Amazon EMR. Для імплементації операцій над оброблюваними даними було обрано розподілений рушій Apache Spark. Роль аналітичного засобу виконує BI застосунок Metabase. Для автоматизації розгортання компонентів системи було використано CloudFormation шаблони, Docker контейнери та CI/CD процеси GitHub Actions. Для імплементації Spark задач та сервісних script-ів(скриптів) було обрано мову програмування Python. В якості інструментів реалізації програмного засобу було обрано PyCharm IDE Community version 2021.3.

Можливі сфери застосування. Створений програмний продукт може бути адаптований під схожий ELT процес обробки та аналітики даних або використаний в навчальних цілях для ознайомлення з ELT парадигмою, дистрибутивним обчислювальним рушієм Spark та сервісом хмарних обчислень AWS. Система може бути покращена за допомогою інтеграції оркестратора, наприклад Apache Airflow, для автоматичного запуску Spark задач.

РОЗДІЛ 1 ВЕЛИКІ ОБСЯГИ ДАНИХ. ETL та ELT

1.1 “Великі Дані” (Big Data)

Великі дані – це колекція наборів даних, які мають складну структуру і об’ємні у своїй природі, за рахунок чого їх стає складно обробляти та аналізувати за допомогою звичайних баз даних. Великі дані можуть бути отримані з різних джерел: медичні дані, записи про грошові транзакції, дописи в соціальних мережах, цифрові зображення та відео, сигнали GPS мобільних телефонів, тощо. Крім того, великі дані допомагають бізнесу зорієнтуватися на потенційних клієнтів і рекомендувати послуги, які вони бажають. Великі дані в основному класифікуються[1] на три типи:

- Структуровані дані.
- Неструктуровані дані.
- Напівструктуровані дані.

Будь-які дані, які можна зберігати, отримувати доступ і обробляти у формі фіксованого формату, називають структурованими даними. Сьогодні існує багато ефективних методів роботи з такими даними (де формат добре відомий заздалегідь), а також у отриманні з них цінності. Однак у наш час ми передбачаємо проблеми, коли розмір таких даних значно зростає, типові розміри досягають кількох зетабайтів[2]. Дивлячись на ці цифри, можна легко зрозуміти, чому дано назву Big Data, і уявити собі проблеми, пов’язані з їх зберіганням і обробкою. Завдяки визначеним розмірам складових та однорідному формату даних, що обробляються, потрібно дуже мало підготовки для забезпечення сумісності всіх джерел.

Не всі дані мають структурований та добре організований вигляд з інструкціями щодо використання. Всі невпорядковані дані класифікуються, як неструктуровані дані. Щоб отримати реальну цінність

з даних, набори даних повинні бути відповідним чином інтерпретованими. Складність аналізу неструктурованих даних полягає в тому, щоб навчити програму розуміти інформацію, яку вона витягує. Часто потрібно перетворити її у структуровану форму, що не є легким завданням і залежить від форматів, у яких зберігається інформація та кінцевих цілей. Деякі методи перетворення полягають у використанні синтаксичного аналізу тексту, обробки природної мови, тощо.

Напівструктуровані дані є чимось середнім між структурованими та неструктурованими даними. Здебільшого це означає, що ми маємо справу з неструктурованими даними, до яких прикріплено метадані. Розглянемо приклад електронного листа. Час надсилання електронного листа, адреси електронної пошти відправника та одержувача, IP-адреса пристрою, з якого було надіслано електронний лист, та інша відповідна інформація пов’язані зі змістом електронного листа. Хоча фактичний вміст сам по собі не структурований, ці компоненти дозволяють групувати дані структурованим чином.

Вищезазначені три типи Big Data технічно застосовні на всіх рівнях аналітики. Важливо зрозуміти джерело сирих даних та їх попередню обробку перед тим, як приступати до опрацювання та аналізу великих обсягів даних. Оскільки даних неймовірно багато, необхідно ефективно проводити видобування інформації для отримання найбільшої користі від даних. Навантаження на ELT/ETL процеси для кожної структури даних відрізняється.

1.2 Проблеми та виклики при обробці великих даних. Характеристики Big Data – “7V”

Поняття «Великі Дані» описав у 2001 році Doug Laney[3] з точки зору його атрибутів, відомих як “3V”: Volume(обсяг),

Variety(різноманітність) та Velocity(швидкість) . Пізніше в 2014 році IBM окреслила ще два інших атрибути “V”: Value(цінність) і Veracity(достовірність), таким чином створюючи “5V” великих даних [4]. Далі з часом були запропоновані ще 2 атрибути: Variability(мінливість) та Visibility(візуалізація)[5]. Розглянемо детальніше кожен із них:

- **Volume (обсяг).** Обсяги даних сьогодні швидко зростають. Для різних організацій об’єми даних можуть коливатись від десятків терабайт даних до сотень петабайт. Згідно з прогнозами, до 2025 року світ створить 181 зетабайт даних[2]. Тож, з кожним разом аналізувати їх без спеціальних технологій стає неможливим.
- **Velocity (швидкість).** Великі дані постійно оновлюються. Наприклад, щохвилини виконується 9 млн запитів у пошуковій системі Google[6]. Завдання Big Data полягає в тому, щоб впоратися зі швидкістю, з якою ці дані створюються, та аналізувати їх у режимі реального часу. В іншому випадку, поки ви аналізуватимете одні дані, оновляться інші, і ваш аналіз стане вже неактуальним.
- **Variety (різноманітність).** Big Data може мати різні формати, залежно від джерела інформації – структуровані та неструктуровані дані, текстова інформація, графіка, дані електронної пошти, інформація із соцмереж, ЗМІ, відео, дані про транзакції, бази даних, архіви. Кожен формат потрібно аналізувати по-різному. Наприклад, користувач може зв’язатися з компанією через соціальні мережі за допомогою комп’ютера, переглядати вебсайт компанії на смартфоні, робити покупки за допомогою планшета та зв’язуватися зі службою підтримки клієнтів електронною поштою. Таким чином, усі дані генеруються від однієї особи, але мають різні форми.
- **Veracity (достовірність).** Це важлива характеристика, адже дані безумовно мають бути правдивими. Якщо компанія працює з недостовірною інформацією, то вона не зможе приймати вірні

управлінські рішення, а її ініціативи зазнають невдачі. Прикладом можна вважати контакти клієнтів з будь-якою неточною інформацією (номери телефонів, імена, дати народження тощо). Тож, через це можна надсилати пропозиції нерелевантній аудиторії та не ефективно використовувати рекламний бюджет.

- **Variability (мінливість).** Треба розуміти, що одні й ті самі дані можуть швидко змінювати свій контекст. З часом залежно від різноманітних обставин, наприклад, економічних, соціальних або політичних, дані можуть переінакшити свою поведінку. Необхідно якомога частіше фіксувати ці зміни та підлаштовувати свою стратегію під це.
- **Visibility (візуалізація).** Якісне відображення аналізу великих даних робить звіти доступними для сприйняття. Це не просто електронні таблиці або текстові файли з числами та формулами. Зрозумілі діаграми, кольорові графіки та інтерактивні карти допомагають легко опрацьовувати дані і робити висновки на їх основі.
- **Value (цінність).** Необхідно не просто аналізувати великі дані, а й отримувати максимум користі від результатів роботи з інформацією. Завдяки цьому, наприклад, компанія може зосередитись, на розширенні асортименту товарів, які актуальні саме для її клієнтів. Це може стати значною конкурентною перевагою.

Фактично, ці концепти і є тими викликами та потребами, на які Big Data намагається відповісти. Також можна побачити, що з плином часу збільшується кількість великих даних, вони урізноманітнюються. Не виключено, що найближчим часом кількість “V” у характеристиці Big Data знову зросте.

1.3 Аналітика даних в реальному часі

Аналітика в режимі реального часу включає впорядкування та підготовку оброблених потоків даних, щоб мати можливість отримувати інформацію з них і негайно діяти на основі цих даних. Аналітика в режимі реального часу використовує ці потоки даних для вирішення проблем і прийняття рішень за лічені секунди.

Сьогодні поведінка людей, компаній, цілих ринків може залежати від одного допису Ілона Маска[7], що показує, який великий вплив можуть мати незначні події і до яких кардинальних змін це може призвести. Важливо, що ці зміни відбуваються дуже швидко, що потребує такої ж швидкої реакції на ці зміни.

Традиційні інструменти пакетної аналітики не спроможні вирішувати ті задачі, які вирішує аналітика в реальному часі. Вони не здатні підтримувати потокову обробку даних, що не дає можливості автоматично отримувати інформацію про зміни в цих даних.

Аналітика даних у реальному часі, може бути використана для наступних потреб:

- Швидке реагування на зміни ринку: завдяки аналітиці великих даних у реальному часі підприємства можуть швидко виявляти ринкові тенденції, уподобання клієнтів і нові можливості та реагувати на них.
- Покращення взаємодії з клієнтами: аналітика в режимі реального часу дозволяє компаніям персоналізувати взаємодію з клієнтами в режимі реального часу.
- Операційна ефективність: аналітика в режимі реального часу надає розуміння операційних процесів, дозволяючи організаціям оптимізувати ефективність і зменшити витрати.
- Управління ризиками та кібербезпека. Аналітика великих даних у режимі реального часу відіграє вирішальну роль у виявленні та

зменшенні ризиків, зокрема загроз кібербезпеці. Відстежуючи мережевий трафік, системні журнали та поведінку користувачів у режимі реального часу, організації можуть проактивно виявляти та реагувати на порушення безпеки, забезпечуючи цілісність і конфіденційність своїх даних.

1.4 ETL та ELT

ELT (extract, load, transform) і ETL (extract, transform, load) – це процеси інтеграції даних, які переміщують необроблені дані з вихідної системи в цільову базу даних, таку як озеро даних або сховище даних[8]. Ці джерела даних можуть бути в кількох різних сховищах або в застарілих системах, які потім передаються за допомогою ELT або ETL до цільового розташування даних.

ETL розшифровується як “витяг, перетворення, завантаження”. Саме в цьому порядку він виконує операції над даними. За допомогою ETL неструктуровані дані витягуються з вихідної системи, а конкретні точки даних і потенційні «ключі» ідентифікуються перед завантаженням даних у цільові системи. У традиційному сценарії ETL вихідні дані витягуються в проміжну область і переміщуються в цільову систему. У проміжній області дані проходять процес перетворення, який впорядковує та очищає всі типи даних. Цей процес трансформації дозволяє тепер структурованим даним бути сумісними з цільовими системами зберігання даних.

ELT розшифровується як “витяг, завантаження, перетворення”. Так само, як в ETL порядок слів в назві відповідає порядку операцій виконуваних над даними. За допомогою ELT неструктуровані дані витягуються з вихідної системи та завантажуються в цільову систему для подальшого перетворення за потреби. ELT використовує сховища даних для базових перетворень даних, таких як перевірка даних, видалення

дубльованих даних, трансформація або агреграція до потрібного вигляду. Ці процеси оновлюються в режимі реального часу та використовуються для великих обсягів необроблених даних. ELT – це новий процес, який не повністю реалізував свій потенціал у порівнянні зі своєю старшою сестрою ETL.

Підхід ELT забезпечує швидшу імплементацію, ніж процес ETL, хоча дані безладні після їх переміщення. Фундаментальна властивість ELT: перетворення відбувається після функції завантаження, запобігаючи уповільненню міграції, яке може статися під час цього процесу. ELT роз'єднує етапи трансформації та завантаження, гарантуючи, що помилка кодування (або інша помилка на етапі трансформації) не призведе до зупинки процесу міграції. Крім того, ELT дозволяє уникнути проблем із масштабуванням сервера, використовуючи обчислювальну потужність і розмір сховища даних для забезпечення трансформації (або масштабованих обчислень) у великих масштабах. ELT також працює з рішеннями хмарного сховища даних для підтримки структурованих, неструктурованих, напівструктурованих і необроблених типів даних.

З ELT процес спрощується тим, що не потрібні «ключі» чи інші ідентифікатори для даних, які передаються та використовуються. Час завантаження коротший, оскільки процес не має стільки етапів. Рішення ELT для систем аналітичних походить від необхідності мати можливість швидко завантажувати неструктуровані дані. Хмарне автоматизоване рішення ELT також може вимагати відносно низьких витрат на обслуговування.

РОЗДІЛ 2 ДИСТРИБУТИВНИЙ РУШІЙ APACHE SPARK.

Apache Spark – це розподілена система обробки даних з відкритим вихідним кодом, яка використовується для великих робочих навантажень даних[9]. Він використовує кешування в пам’яті та оптимізоване виконання запитів для швидких аналітичних запитів щодо даних будь-якого розміру. Він надає API для розробки на Java, Scala, Python і R, а також підтримує повторне використання коду в кількох робочих навантаженнях – пакетна обробка, потокова передача даних, інтерактивні запити, аналітика в реальному часі, машинне навчання та обробка графіків. Apache Spark став однією з найпопулярніших інфраструктур розподіленої обробки великих даних, його використовують організації з будь-якої галузі, зокрема FINRA, Yelp, Zillow, DataXu, Urban Institute і CrowdStrike[11]. Spark може бути в 100 разів швидшим за Hadoop[12] для великомасштабної обробки даних завдяки використанню обчислень пам’яті та інших оптимізацій. Spark також працює швидко, коли дані зберігаються на диску, і наразі є світовим рекордсменом для великомасштабного сортування[13].

Spark було створено для усунення обмежень MapReduce[14] шляхом виконання обробки в пам’яті, зменшення кількості кроків у завданні та повторного використання даних у кількох паралельних операціях. У MapReduce кожна дія над даними вимагає операції читання-запису диску. у У випадку Spark достатньо один раз зчитати дані в пам’ять, виконати всі обчислення і перетворення, та один раз записати результати, що забезпечує набагато швидше виконання. Spark також повторно використовує дані за допомогою кешу в пам’яті, що значно пришвидшує, наприклад, роботу алгоритмів машинного навчання, які неодноразово викликають функцію для того самого набору даних.

2.1 Складові Apache Spark

Структура Spark включає в себе: Spark Core як основа для платформи, Spark SQL для інтерактивних запитів, Spark Streaming для обробки потокових даних та аналітики в реальному часі, Spark MLlib для машинного навчання, Spark GraphX для обробки графів(див. рис. 2.1)

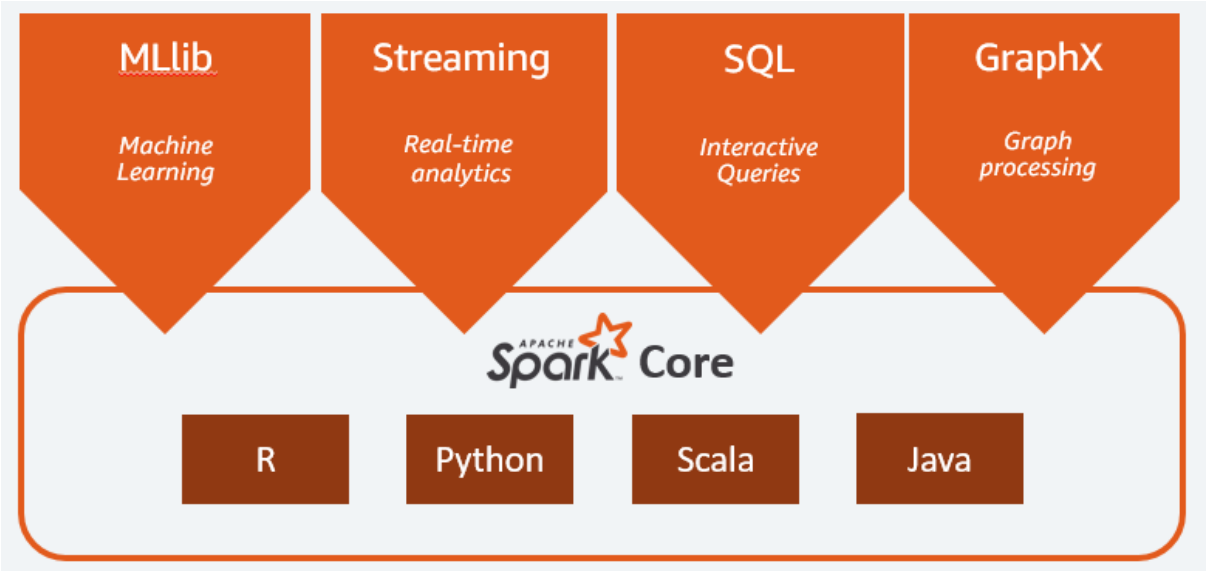

Рисунок 2.1 – Структура Apache Spark[15]

Spark Core є основою платформи. Він відповідає за керування пам’яттю, відновлення після помилок, планування, розподіл і моніторинг завдань, а також взаємодію із системами зберігання. Spark Core доступний через інтерфейс прикладного програмування (API), створений для Java, Scala, Python і R. Ці API приховують складність розподіленої обробки за простими операторами високого рівня.

Spark Streaming – це рішення в режимі реального часу, яке використовує можливості швидкого планування Spark Core для аналізу потокової передачі. Він приймає дані в міні-пакетах і виконує аналітику цих даних за допомогою того самого програмного коду, написаного для пакетної аналітики. Це покращує продуктивність розробки, оскільки

можна використовувати той самий код для пакетної обробки та потокових додатків.

Spark SQL – це розподілена система запитів, яка забезпечує інтерактивні запити з низькою затримкою, і працює до 100 разів швидше, ніж MapReduce. Вона включає в себе оптимізатор на основі витрат, стовпчасте сховище та генерацію коду для швидких запитів із масштабуванням до тисяч вузлів.

2.2 Менеджмент задач і обчислень в Spark

Програми Spark запускаються як незалежні набори процесів у кластері, які координуються об’єктом SparkContext у головній програмі (так званій програмі-драйвері)[16].

Зокрема, щоб працювати в кластері, SparkContext може підключатися до кількох типів менеджерів кластерів (це може власний автономний менеджер, YARN або Kubernetes), які розподіляють ресурси між програмами. Після підключення Spark отримує виконавців(executor-ів) на вузлах у кластері, що є процесами, які виконують обчислення та зберігають дані для програми. Потім він надсилає код програми (визначений файлами JAR або Python, переданими до SparkContext) виконавцям. Нарешті, SparkContext надсилає завдання виконавцям для виконання(див. рис. 2.2).

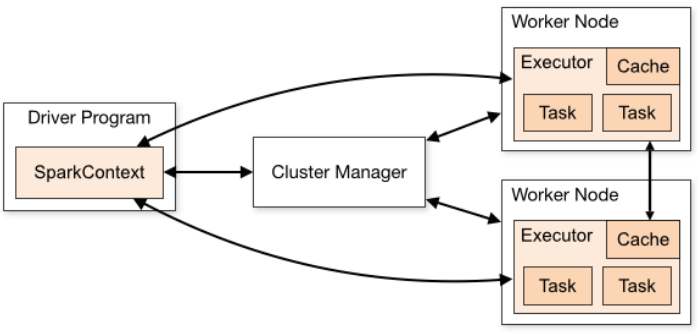

Рисунок 2.2 – Принцип роботи Spark кластеру[16]

Є кілька речей, які слід зазначити про цю архітектуру:

- Кожна програма отримує власні процеси-виконавці, які залишаються активними протягом усієї програми та виконують завдання в кількох потоках. Це надає перевагу ізоляції програм одна від одної як на стороні планування (кожен драйвер планує власні завдання), так і на стороні виконавця (завдання з різних програм виконуються в різних JVM). Однак це також означає, що дані не можна обмінювати між різними програмами Spark (екземплярами SparkContext), не записавши їх у зовнішню систему зберігання.
- Spark не залежить від менеджера кластера. Поки він може отримати процеси-виконавці, і вони взаємодіють один з одним, його відносно легко запустити навіть у менеджері кластерів, який також підтримує інші програми (наприклад, YARN/Kubernetes).
- Програма драйвера повинна прослуховувати та приймати вхідні з'єднання від своїх виконавців протягом усього терміну роботи. Таким чином, програма-драйвер повинна мати доступ до мережі з вузлів виконавців.

На даний момент система підтримує кілька менеджерів кластерів:

- Автономний – власний менеджер кластерів, що входить до складу Spark, який спрощує налаштування кластера.
- Hadoop YARN – менеджер ресурсів у Hadoop 2 і 3[17].
- Kubernetes – система з відкритим кодом для автоматизації розгортання, масштабування та керування контейнерними програмами[18].

РОЗДІЛ 3 ХМАРНІ ОБЧИСЛЕННЯ ТА ЇХ ВИКОРИСТАННЯ ПРИ РОБОТІ З ВЕЛИКИМИ ДАНИМИ І ELT .

3.1 Хмарні обчислення

Загалом, хмарні обчислення – це надання комп'ютерних послуг, зокрема серверів, сховищ, баз даних, мереж, програмного забезпечення, аналітики, тощо – через Інтернет («хмару»), щоб запропонувати швидшу інноваційність, гнучкі ресурси та економію при масштабуванні[19]. Зазвичай користувач платить лише за ті послуги, якими користуєтеся, що допомагає вам знизити операційні витрати, ефективніше керувати інфраструктурою та масштабуватись за потреби. Наразі найпоширенішими сервісами хмарних обчислень є Amazon Web Services(34% ринку), Microsoft Azure(21% ринку) та Google Cloud Platform(11% ринку)[20].

Хмарні обчислення – це значна зміна у порівнянні з традиційним уявленням компаній про IT-ресурси. Ось поширені причини, чому організації звертаються до послуг хмарних обчислень:

- **Вартість.** Перехід на хмару допомагає компаніям оптимізувати IT-витрати. Це пояснюється тим, що хмарні обчислення усувають капітальні витрати на купівлю обладнання та програмного забезпечення, а також встановлення та роботу локальних центрів обробки даних – стійки серверів, цілодобову електроенергію для живлення та охолодження, а також IT-експертів для керування інфраструктурою.
- **Швидкість.** Більшість служб хмарних обчислень надаються за запитом, тому навіть величезні обсяги обчислювальних ресурсів можуть бути надані за лічені хвилини, як правило, лише кількома

клацаннями миші, що дає підприємствам велику гнучкість і зменшує навантаження на планування потужностей.

- **Масштабованість.** Переваги послуг хмарних обчислень включають можливість еластичного масштабування. З точки зору хмари це означає надання необхідної кількості IT-ресурсів – наприклад, більшої чи меншої обчислювальної потужності, пам'яті, пропускної здатності – саме тоді, коли вони потрібні, і з потрібного географічного розташування.
- **Продуктивність.** Локальні датацентри, як правило, вимагають багато зусиль на їх підтримку— налаштування апаратного забезпечення, оновлення програмного забезпечення тощо. Хмарні обчислення усувають необхідність виконувати багато з цих завдань, тому IT-команди можуть витрачати час на досягнення більш важливих бізнес-цілей.
- **Ефективність.** Найбільші сервіси хмарних обчислень працюють у всесвітній мережі захищених центрів обробки даних, які регулярно оновлюються до останнього покоління швидкого та ефективного обчислювального обладнання. Це пропонує кілька переваг порівняно з єдиним корпоративним центром обробки даних, зокрема зменшену затримку мережі для додатків і більшу економію при масштабуванні.
- **Надійність**. Хмарні обчислення спрощують і здешевлюють резервне копіювання даних, аварійне відновлення та забезпечення безперервності роботи системи, оскільки дані можуть бути репліковані на кількох резервних хостах у мережі хмарного провайдера.
- **Безпека.** Багато хмарних провайдерів пропонують широкий набір політик, технологій і елементів керування, які допомагають у налаштуванні і посиленні безпеки, допомагаючи захистити дані, програми та інфраструктуру від потенційних загроз.

Якщо казати про хмарні обчислення у комбінації з ELT та Big Data, то вищеперелічені якості cloud computing набувають ще більшого сенсу. Можна помітити, що вони перегукуються з “7V” Big Data, які були описані у пункті 1.2. Наприклад, великі масиви даних потребують інфраструктури для зберігання та виконання операцій на ними, що або неможливо зробити за рахунок кількості оброблюваної інформації, або дуже ресурсозатратно з точки зору підтримки. Хмарні сервіси вирішують цю проблему.

В ELT pipeline-ах забезпечення швидкої масштабованості системи у випадку різкого стрибка навантаження відіграє важливу роль. Реалізація цього процесу у власному середовищі може потребувати неабияких зусиль, як технічних так і фінансових. У випадку розгортання інфраструктури у хмарі, всі ці процеси керуються нею, що дозволяє зосередитись на програмній розробці і витрачати кошти лише на ресурси, які справді використовуються.

Дослідження Forrester у 2017 році[21] показало, що рішення для великих даних за допомогою хмарних обчислень збільшаться приблизно в 7,5 разів швидше, ніж локальні варіанти цих рішень.

3.2 Типи хмарних сервісів

Більшість сервісів хмарних обчислень поділяються на чотири великі категорії[19]:

- **Інфраструктура як сервіс (IaaS).** Модель «Інфраструктура як сервіс», включає основні елементи для створення хмарної IT-системи. У рамках цієї моделі користувачі отримують доступ до мережевих ресурсів, віртуальних комп’ютерів або спеціального обладнання та сховищ даних. Модель «Інфраструктура як сервіс» забезпечує найвищий рівень гнучкості в роботі та управлінні

IT-ресурсами. Вона майже ідентична сучасній моделі IT-ресурсів, яку використовують IT-персонал і розробники.

- **Платформа як сервіс (PaaS).** Модель «Платформа як сервіс» не вимагає від організації керування основною інфраструктурою (як правило, включаючи апаратне забезпечення та операційні системи) і дозволяє їй присвятити всі свої зусилля розробці та управлінню програмами. Це покращує продуктивність, усуваючи необхідність турбуватися про придбання серверів, планування потужностей, обслуговування програмного забезпечення, оновлення безпеки та інші трудомісткі завдання, необхідні для запуску програм.
- **Програмне забезпечення як сервіс (SaaS).** Відповідно до моделі програмного забезпечення як сервісу, користувач отримує готовий продукт, яким керує постачальник цієї послуги. Зазвичай в цьому випадку мова йде про програми для кінцевих користувачів. Працюючи з моделлю SaaS, не потрібно турбуватися про підтримку служби чи керування основною інфраструктурою, і з’являється можливість повністю зосередитися на використанні конкретного програмного забезпечення. Добре відомим прикладом програми SaaS є веб-служба імейлів, яка дозволяє надсилати і отримувати електронну пошту без необхідності керувати додатковими компонентами програмного продукту чи підтримувати сервери й операційні системи, на яких працює ця служба.
- **Безсерверні обчислення.** Аналогічно до PaaS, безсерверні обчислення зосереджені на розробці функціональних можливостей програми, не витрачаючи час на постійне керування серверами та інфраструктурою, необхідною для цього. Хмарний постачальник виконує налаштування, планування потужностей і керування сервером за вас. Безсерверні архітектури мають високу

масштабованість і керуються подіями: вони використовують ресурси лише тоді, коли виникає певна функція чи тригер.

3.3 Інфраструктура як код

Інфраструктура як код (Infrastructure as Code, IaC) — це управління і надання інфраструктури за допомогою коду, а не за допомогою ручних процесів[22]. За допомогою IaC створюються файли конфігурації, які містять специфікації розроблюваної інфраструктури, що полегшує редагування та розповсюдження конфігурацій. Це також гарантує, що щоразу розгортається одне й те саме середовище. Кодифікуючи та документуючи специфікації конфігурації, IaC допомагає керувати конфігурацією та допомагає уникнути незадокументованих, спеціальних змін конфігурації.

Контроль версій є важливою частиною IaC, файли конфігурації повинні перебувати під системою контролю версій, як і будь-який інший файл вихідного коду програмного забезпечення. Розгортання інфраструктури як коду також означає, що з’являється можливість розділити інфраструктуру на модульні компоненти, які потім можна комбінувати різними способами за допомогою автоматизації.

Автоматизація надання інфраструктури за допомогою IaC означає, що розробникам не потрібно вручну розгортати та налаштовувати сервери, операційні системи, сховища даних та інші компонентами інфраструктури кожного разу, коли вони розробляють або розгортають програму.

Завдяки хмарним обчисленням кількість компонентів інфраструктури зросла, щодня випускається більше додатків, а інфраструктуру потрібно часто розгортати, масштабувати та знімати. Без практики IaC керувати масштабом сучасної інфраструктури стає все важче. З найбільших переваг IaC:

- Зниження витрат;
- Збільшення швидкості розгортань;
- Зменшити кількість помилок;
- Покращення узгодженості інфраструктури;
- Усунення розбіжностей в конфігурації.

3.4 AWS

Amazon Web Services (AWS) – це найпоширеніший cloud(клауд) у світі, який пропонує понад 200 повнофункціональних послуг із центрів обробки даних у всьому світі. Мільйони клієнтів, включаючи стартапи, найбільші підприємства та провідні державні установи, використовують AWS, щоб знизити витрати, стати більш гнучкими та швидше впроваджувати інновації[23].

Amazon надає новим користувачам широкий спектр сервісів у безкоштовне користування у рамках free-tier[24]. Це дає змогу розробникам без досвіду у роботі з хмарними обчисленнями спробувати щось нове чи протестувати вже створене рішення в форматі взаємодії з хмарою.

3.4.1 Amazon S3

Amazon Simple Storage Service (Amazon S3) – це сховище зберігання об’єктів, яка пропонує масштабованість, доступність даних, безпеку та продуктивність[25]. Клієнти будь-якого розміру та галузі можуть використовувати Amazon S3 для зберігання та захисту будь-якої кількості даних для різноманітних випадків використання, таких як озера даних, веб-сайти, мобільні програми, резервне копіювання та відновлення, архівування, корпоративні програми, пристрої IoT та великі дані, аналітика. Amazon S3 надає функції керування, для оптимізації,

упорядкування та налаштування доступу до даних відповідно до конкретних організаційних і бізнес-вимог(див. рис. 3.1).

Рисунок 3.1 – Опис сховища даних Amazon S3

3.4.2 Amazon EMR

Amazon EMR (раніше називалася Amazon Elastic MapReduce) – це керована кластерна платформа, яка спрощує запуск інфраструктур великих даних, наприклад Apache Hadoop і Apache Spark, на AWS для обробки й аналізу великих обсягів даних[26]. За їх допомогою можна обробляти дані для аналітичних і бізнес цілей(див. рис. 3.2). Amazon EMR також дозволяє трансформувати та переміщувати великі обсяги даних у та з інших сховищ даних і баз даних AWS, таких як Amazon Simple Storage Service (Amazon S3) і Amazon DynamoDB.

Рисунок 3.2 – Опис кластерної платформи Amazon EMR

Центральним компонентом Amazon EMR є кластер. Кластер – це сукупність екземплярів віртуальних машин Amazon Elastic Compute Cloud (Amazon EC2)[27]. Кожен екземпляр у кластері називається вузлом(node). Кожен вузол має роль у кластері, яка називається типом вузла. Amazon EMR також встановлює різні програмні компоненти на кожен тип вузла, надаючи кожному вузлу роль у розподіленій програмі. За цією поведінкою він схожий на кластер Apache Hadoop[13].

Нижче наведено типи вузлів Amazon EMR(див. рис. 3.3):

- Головний вузол(Primary node): вузол, який керує кластером, запускаючи програмні компоненти для координації розподілу даних і завдань між іншими вузлами для обробки. Основний вузол відстежує стан завдань і контролює працездатність кластера. Кожен кластер має основний вузол, і можна створити одновузловий кластер лише з основним вузлом.
- Основний вузол(Core node): вузол із програмними компонентами, які виконують завдання та зберігають дані в файловій системі кластеру. Багатовузлові кластери мають принаймні один основний вузол.
- Вузол завдань(Task node): вузол із програмними компонентами, який лише виконує завдання і не зберігає дані у файловій системі. Вузли завдань необов’язкові.

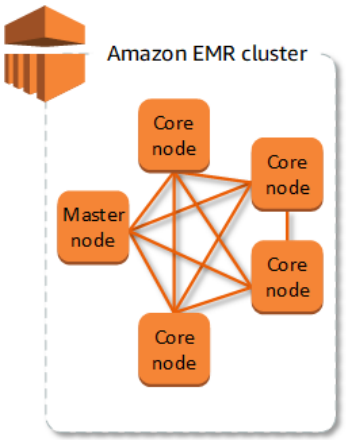

Рисунок 3.3 – Приклад архітектури вузлів EMR кластеру

РОЗДІЛ 4 ПОСТАНОВКА ВИРІШУВАНОЇ ЗАДАЧІ. ОГЛЯД ВИКОРИСТАНИЙ ТЕХНОЛОГІЙ. ОПИС АРХІТЕКТУРИ РОЗРОБЛЕНОЇ СИСТЕМИ.

4.1 Постановка задачі.

Ідеєю роботи була розробка типового ELT pipeline-у, який здатний обробляти дані в реальному часі, зберігати їх в сховище та агрегувати відповідним чином, для отримання з них цінної інформації. Також передбачена автоматизація розгортання всіх компонентів системи, що дозволить відновити, копіювати або масштабувати її без великої кількості зусиль витрачених на налаштування.

Для тестування коректності роботи системи було обраний відкритий набір даних служби таксі Нью-Йорку[28]. Цей датасет являє собою інформацію про всі поїздки різних служб таксі, розділені по місяцям. Було реалізовано script, який імітує передачу даних про нові поїздки в реальному часі до системи. Після збору переданих даних вони були збережені і агреговані для отримання цінної статистичної інформації: кількість поїздок залежно від локації, середня виручка кожної служби залежно від години, тощо.

4.2 Опис архітектури та використаних інструментів

При розробці програмного продукту були використані наступні технології:

- Apache Kafka – платформа із відкритим кодом призначена для розподіленої потокової передачі даних[29]. Вона здатна до роботи з високою пропускною здатністю, стійкістю до збоїв і потоковими даними в реальному часі. За своєю суттю Kafka функціонує як

розподілена pub-sub система з повідомленнями. Це дозволяє producer-ам публікувати дані в topic- ах, а consumer-и можуть підписуватися на ці topic-и, щоб отримувати дані в режимі реального часу. Повідомлення в Kafka організовані за topic-ами, які поділені на розділи, розподілені між кількома broker-ами або серверами.

- Apache Spark (див. розділ 2)
- Amazon S3 (див. пункт 3.3.1)
- Amazon EMR. (див. пункт 3.3.2)
- Amazon RDS – Сервіс AWS, що спрощує налаштування, запуск і масштабування реляційних баз даних у хмарі[30]. Для розробленої системи було обрано варіацію RDS з Postgres рушієм.
- Metabase – інструмент бізнес-аналітики з відкритим кодом[31]. Metabase дозволяє ставити запитання про дані та відображає відповіді в зрозумілих форматах, таких, як гістограма, мапа чи детальна таблиця.
- Apache Parquet – формат файлів, розроблений для ефективного зберігання та пошуку даних[32]. Він забезпечує ефективне стиснення даних і схеми кодування з підвищеною продуктивністю для масової обробки складних даних. Parquet – це формат із стовпчастою схемою. Файли Parquet складаються з груп рядків, верхнього та нижнього колонтитулів. Кожна група рядків містить дані з однакових стовпців. Ті самі стовпці зберігаються разом у кожній групі рядків:
- Docker – платформа з відкритим кодом, яка дозволяє розробникам створювати, розгортати, запускати, оновлювати та керувати контейнерами – стандартизованими виконуваними компонентами, які поєднують вихідний код програми з бібліотеками операційної системи (ОС) і залежностями, необхідними для запуску цього коду в будь-якому середовищі[33].

- Amazon EC2 – Сервіс, що забезпечує масштабовану обчислювальну потужність у хмарі Amazon Web Services (AWS). Використання віртуальних машин Amazon EC2 позбавляє необхідності інвестувати в апаратне забезпечення, тож з'являється можливість швидше розробляти та розгортати програми.
- AWS CloudFormation – сервіс, який допомагає моделювати та налаштовувати ресурси AWS, щоб можна було витрачати менше часу на керування цими ресурсами та більше часу, зосереджуючись на розробці програм, які працюють у AWS[34]. CloudFormation працює з шаблоном, який описує всі потрібні ресурси AWS (наприклад, екземпляри Amazon EC2 або екземпляри Amazon RDS). Після отримання шаблону він сам подбає про надання та налаштування цих ресурсів.
- boto3 – бібліотека мови Python, що надає API для взаємодії з сервісами AWS[35].

Як же всі ці інструменти працюють в сукупності? Загальна архітектурна діаграма системи зображена на рисунку 4.1. Структура pipeline-у відповідає тому, який шлях проходять дані від їх збору до їх візуалізації:

1. EC2 машина, на якій розгорнутий Docker контейнер Kafka. Дані у JSON форматі потрапляють у чергу.
2. EMR кластер, на якому в реальному часі працює Spark Structured Streaming задача, що "підписана" на відповідний topic всередині Kafka, читає нові дані, як тільки Kafka сповістить її про наявність ще непрочитаних даних. Після того, як Spark зчитав дані з Kafka він додає їх в S3 до вже наявних даних у форматі Parquet. Ця складова системи відповідає за "load" операцію ELT.
3. S3 bucket в якому зберігається вже оброблена інформація, залежності та виконуваний код для Spark задач.

4. Ще один EMR кластер, на якому працює Spark Batch задача, що агрегує потрібні метрики з наявних в сховищі даних і оновлює їх в базі даних. Ця складова системи відповідає за “transform” операцію ELT.
5. RDS Postgres база даних, де в різних таблицях знаходяться статистичні дані про оброблену інформацію.
6. Ще одна EC2 машина, на якій розгорнутий Docker контейнер Metabase. BI інструмент підключається до бази даних і формує візуалізації на основі отриманих даних: карти, графіки тощо.

Окремо від компонентів системи, що безпосередньо працюють з даними, були розроблені сервісні процеси, які забезпечують автоматичне розгортання інфраструктурних і програмних складових системи та допомагають підтримувати pipeline в справному стані і з останньою версією виконуваного коду:

1. CloudFormation шаблон, який являє собою параметризований набір конфігурацій всіх складових системи, що спрощує її створення і не змушує розробника вручну створювати кожен сервіс і окремо налаштовувати сумісність між ними.
2. GitHub Actions CI/CD пайплайн, який при коміті в головну гілку репозиторію оновлює код та конфігурації для Spark задач до останньої версії[36].
3. GitHub Actions пайплайни, які при початковому створенні системи розгортають потрібні Docker контейнери на EC2 машинах і створює потрібні topic-и для даних у Kafka.
4. Python script, який за допомогою boto3 бібліотеки запускає на кластерах Spark задачі та моніторить їх стан.

Фактично, CloudFormation шаблон та реалізовані CI/CD пайплайни утворюють модель IaC, згадану у пункті 3.3, з усіма описаними там перевагами. Усі конфігураційні файли і виконувані файли CI/CD

пайплайнів зберігаються в репозиторії з кодом[37] і ведуться системою контролю версій Git.

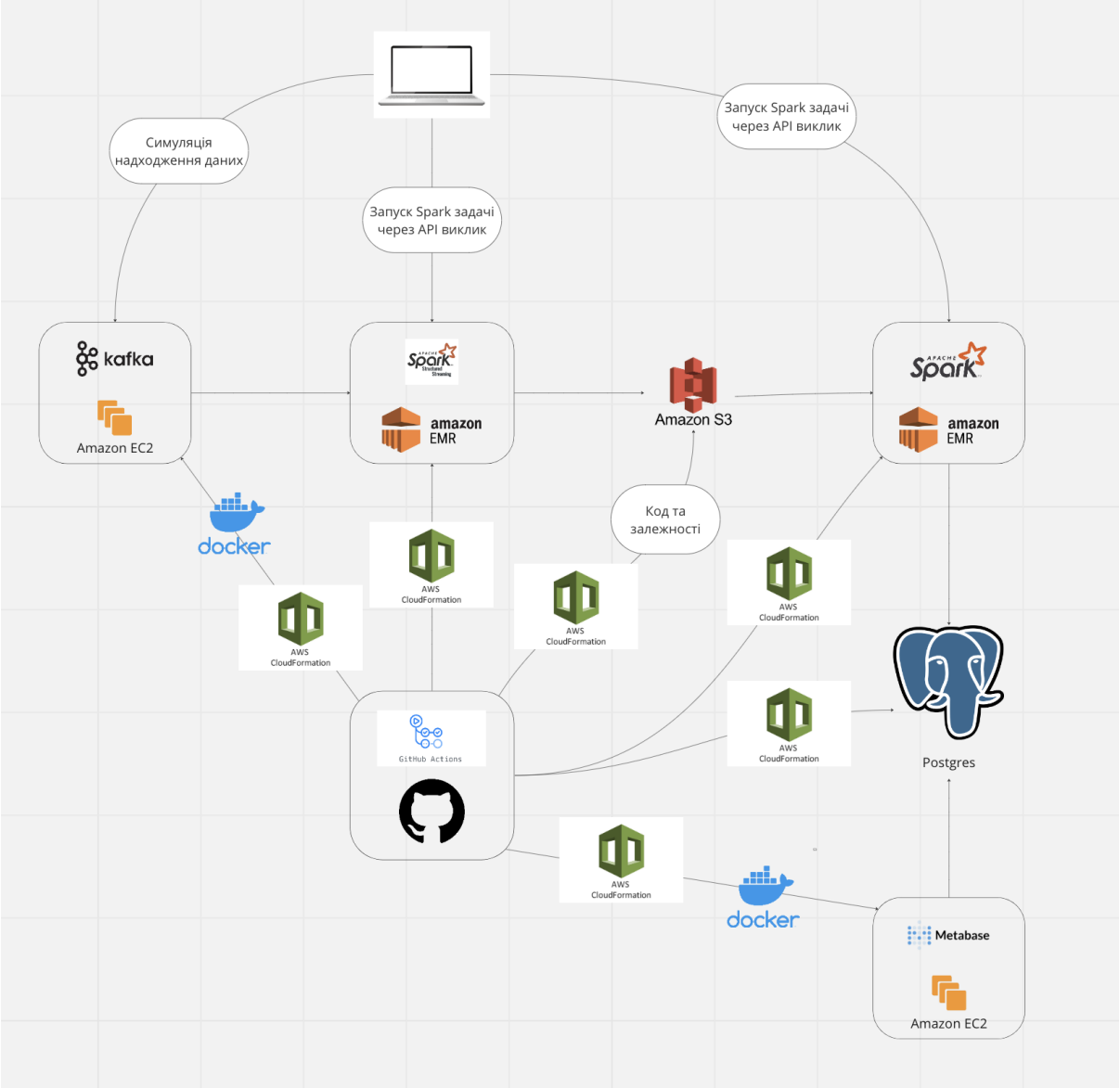

Рисунок 4.1 Архітектурна діаграма системи

РОЗДІЛ 5 ОСОБЛИВОСТІ КОНФІГУРАЦІЇ СИСТЕМИ. СИМУЛЯЦІЯ НАДХОДЖЕННЯ ДАНИХ. ІМПЛЕМЕНТАЦІЯ ПРОГРАМНИХ СКЛАДОВИХ PIPELINE-У

5.1 Особливості конфігурації системи

CloudFormation шаблон параметризований не лише щодо імен складових і мережевих налаштувань. Також у вигляді параметрів передаються типи і кількість EC2 машин, що використовуються або як хости, або як складові кластера. Це забезпечує легку масштабованість системи, лише за допомогою зміни кількох параметрів. Потрібен більший розмір диску для зберігання даних у Kafka чи більші обчислювальні потужності кластера, бо трансформаційна логіка ускладнилась – потрібно лише вказати відповідні параметри і AWS візьме на себе реалізацію масштабування системи.

Також Spark потребує особливих залежностей при роботі з Kafka, S3 та Postgres. Ці залежності вказані в коді(див. рис. 5.1) і автоматично завантажуються на кластер при першому запуску Spark задачі. У випадку, зміни версії Spark, треба мати на увазі, що треба буде узгодити нову версію ПЗ з версією використовуваних залежностей.

```
packages = [
    'org.apache.spark:spark-sql-kafka-0-10_2.12:3.2.1',
    'org.apache.kafka:kafka-clients:2.8.0',
    'org.postgresql:postgresql:42.5.1',
    'org.apache.hadoop:hadoop-aws:3.2.1',
    'software.amazon.awssdk:bundle:2.15.40',
    'software.amazon.awssdk:url-connection-client:2.15.40',
]
```

Рисунок 5.1 – Список залежностей Spark

5.2 Підготовка та симуляція надходження даних

У вигляді тестових даних було обрано датасет NYC Taxi Trip Data[28]. Цей набір даних надає детальну інформацію про поїздки на таксі у місті Нью-Йорк. Він включає в себе локації початку та закінчення поїздки, протяжності, вартості цих поїздок та інші атрибути[38]. Для отримання максимальної користі з отриманих даних потрібно зробити декілька підготовчих дій.

У наборі даних кожна поїздка містить поле Hvfhs_license_num – ідентифікатор служби таксі, яка здійснювала поїздку і поля PULocationID, DOLocationID – зони же почалась і закінчилась поїздка відповідно. В самому датасеті ці поля виконують роль ідентифікаторів, тому для отримання повної користі з даних було створено допоміжні набори даних:

- licenses.csv – файл де вказано відповідність між ідентифікатором Hvfhs_license_num і службою, що йому відповідає.
- zones.csv – файл де вказано відповідність між ідентифікатором LocationID і характеристиками зони, що йому відповідає.

Ці файли зберігаються у репозиторії з кодом і завантажуються автоматично CI/CD процесом при початковому створенні інфраструктури.

Також після того, як розгорнеться Docker для Metabase треба буде завантажити в нього geojson файл з картою Нью-Йорку[39] вручну, бо цієї карти немає в Metabase за замовчуванням. Сам файл також знаходиться в репозиторії в папці infrastructure/metabase.

Симуляція надходження даних в реальному часі імплементована у вигляді звичайного Python script-у, який через певні проміжки часу надсилає в Kafka topic пакет даних про поїздки.

5.3 Особливості імплементації Spark

В цьому підпункті буде розглянуто особливості імплементації тієї частини pipeline-у, яка безпосередньо обробляє дані: Spark задач.

5.3.1 Spark Streaming задача

Логіка роботи Spark Streaming задачі складається фактично з 4-х частин:

1. Зчитати і розпарсити конфігураційний файл
2. На основі отриманих конфігурацій підписатись на потік Kafka topic-у
3. Після приходу в буфер topic-у нових даних зчитати їх і записати в S3
4. Після успішного збереження даних перейти в режим очікування до того моменту, як в Kafka знову надійдуть дані. Коли це трапиться повторити починаючи з кроку 2.

На рисунку 5.2 показана імплементація функції, що запускає власне Streaming процес. В коді можна побачити наступний рядок: “.option('checkpointLocation', config['checkpoint_location'])” . Ця опція означає, що Spark зберігає місце, де він зупинився при читанні даних з Kafka в заздалегідь зазначене місце. Така властивість буде корисною, якщо, з якоїсь причини відбудеться збій Spark задачі, або EMR кластеру, на якому вона працює. В цьому випадку Spark не буде читати всі дані з Kafka з самого початку, а почне з того моменту, де зупинився.

Також на рисунку присутній рядок: “.foreachBatch(writer)”. Це означає, що для кожного пакету даних, який Spark зчитує з Kafka, він виконує дію описану у writer. Фактично, це є частина коду, яка відповідає за десеріалізацію даних із Kafka, проведення первинної обробки даних і збереження даних в сховище S3(див. рис). З коду видно, що функція повністю параметризована, тобто, вона може працювати з будь-якими даними, що знаходяться у Kafka, якщо вони передаються у туди JSON

форматі і зберігаються в сховищі у Parquet форматі. Єдиний момент, Spark бере JSON-схему даних з конфігураційного файлу, тому зміни в структурі даних повинні бути там відображені.

```
def run_streaming(config, args, spark):
    logging.info('staring streaming query')

    trigger_once = args.trigger_once

    if trigger_once:
        args = dict(once=True)
    else:
        args = dict(processingTime='60 seconds')

    writer = Writer(
        spark=spark,
        config=config
    )

    stream = (
        spark
        .readStream
        .format('kafka')
        .option('kafka.bootstrap.servers', ','.join(config['kafka_brokers']))
        .option('subscribePattern', '|'.join(config['topics']))
        .option('startingOffsets', 'earliest')
        .load()


        .writeStream
        .foreachBatch(writer)
        .option('checkpointLocation', config['checkpoint_location'])
        .trigger(**args)
        .start()
        .awaitTermination()
    )
```

Рисунок 5.2 – Імплементація запуску Streaming задачі

При зміні вхідного JSON формату, наприклад, на AVRO[40] потрібно буде змінити механізм десеріалізації отриманої з Kafka інформації. При зміні вихідного Parquet формату, наприклад, на Delta Lake[41] потрібно буде імплементувати свій механізм запису інформації в сховище.

```
class Writer:
    def __init__(self, config, spark):
        self.spark = spark
        self.config = config

    def __call__(self, df: DataFrame, epoch_id):
        output_table = os.path.join(config['root'], config['output_table'])

        df = utils.add_date_cols((df.where(df.value.isNotNull())
                .select(fn.from_json(fn.col("value").cast("string"), self.config['schema']).alias("value"))
                .select('value.*')
                ))

        (df.write.partitionBy(config['partition_key'])
            .mode('append')
            .parquet(output_table))
```

Рисунок 5.3 – Імплементація функції читання/запису даних з Kafka в S3

Тому можна сказати, що частина pipeline-у, яка відповідає за збір нових даних(“load” операцію ELT) і зберігання їх у S3 – уніфікована. Окрім операцій читання/запису Spark код та інфраструкурні налаштування не будуть потребувати внесення інших змін, якщо буде залишатись сталим сценарій поведінки Spark Streaming задачі: забрати дані з Kafka і зберегти їх у сховище S3.

5.3.2 Spark Batch задача

Spark Batch задача на відміну від Streaming задачі не працює постійно. Її призначення полягає в тому, що зчитати дані, які є у сховищі в даний момент часу, порахувати потрібні метричні дані, оновити їх у базі даних і завершитись. Метрики, які вираховує Spark:

1. загальна кількість поїздок для кожної зони, де ця зона була точкою початку поїздки
2. загальна кількість поїздок для кожної зони, де ця зона була точкою закінчення поїздки
3. загальна кількість поїздок для кожної служби таксі, згруповані по годинам
4. середня кількість чайових залежно від довжини подорожі
5. середня кількість чайових залежно від тривалості подорожі

6. середня кількість чайових залежно від швидкості подорожі

Ці метрики при правильній візуалізації можуть дати цінну інформацію.

Наприклад, на рисунку 5.4 можна побачити карти, які ілюструють перші 2 метрики. З них видно, що в Нью-Йорку люди в найбільше користуються таксі щоб доїхати від і до аеропортів(відтінок синього у цих зон найбільш темний).

На рисунку 5.5 можна побачити поруч графіки залежності чайових від різних факторів(метрики 4-6). З них, наприклад, можна зробити висновок, що, якщо поїздка триває від нуля до 2-х годин, то чим довше вона буде йти, тим більші будуть чайові.

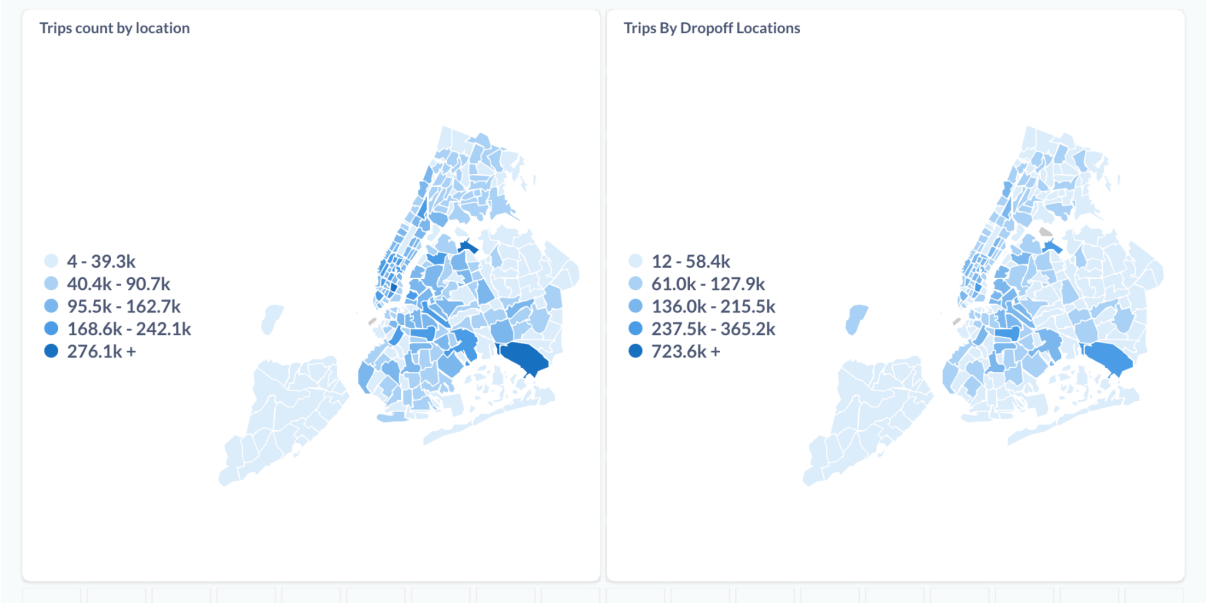

Рисунок 5.4 – Візуалізація метрик 1-2 у Metabase

Так як Batch задача не працює постійно, її треба постійно перезапускати, щоб метричні дані в базі даних були актуальними відносно тих, що знаходяться сховищі. У пункті 4.2 був згаданий Python script, який за допомогою boto3 звертається до кластера і запускає Spark задачу. У цьому script-і, крім самої функції, яка запускає задачу, імплементований декоратор, який задається виконуваною функцією та двома параметрами(див. рис. 5.6):

- max_consecutive_errors – кількість збоїв задачі підряд, після досягнення якої перестати перезапускати задачу. Очевидно, що якщо задача падає велику кількість разів підряд, то існує проблема, з якою треба розбиратись.
- delay – кількість часу у секундах, яку, у разі збою попередньої задачі, script чекає перед тим, як перезапускати задачу

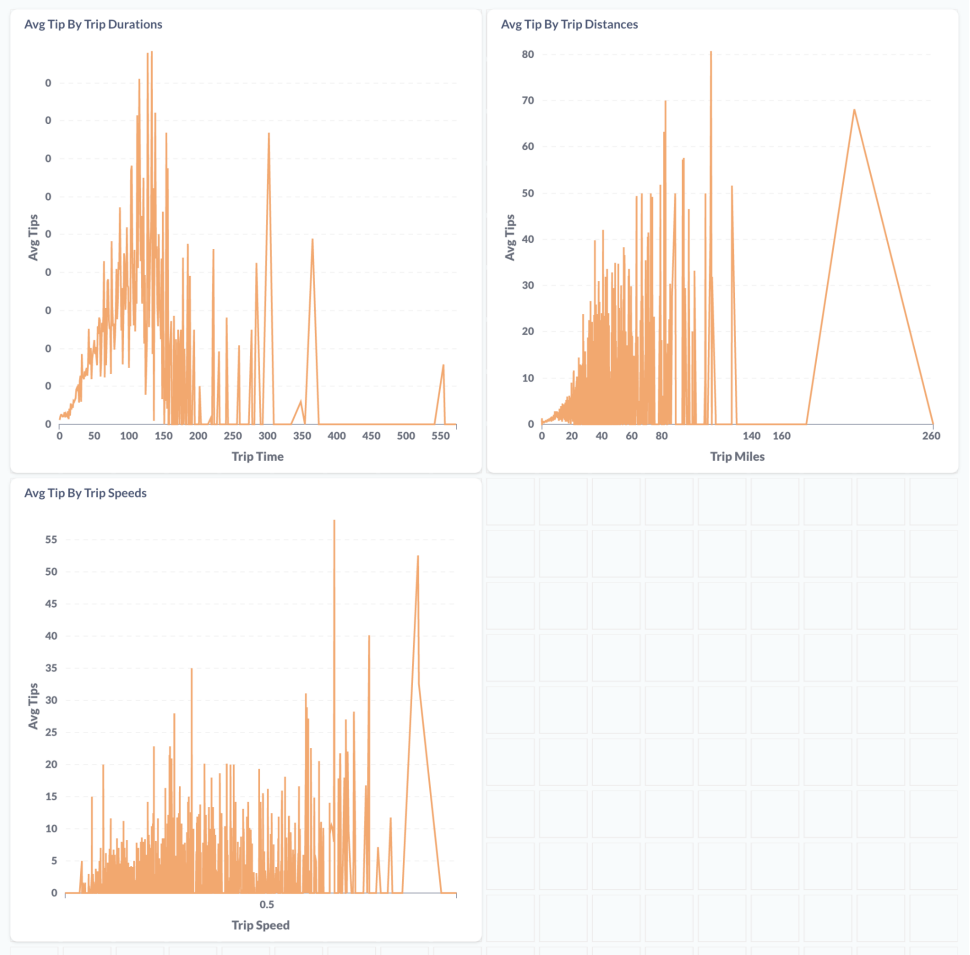

Рисунок 5.5 – Візуалізація метрик 4-6 у Metabase

У випадку Streaming задачі цей декоратор скоріше запобігає простоюванню ресурсів на кластері і виконує моніторингову роль. Мережеве підключення може давати збій з різних причин і потрібно мати механізм автоматичного відновлення синхронізації даних. Знову ж таки, якщо задача дає збій декілька разів підряд, то це стає проблемою, яку

треба вирішувати. Варто зазначити, що це не змінює того факту, що запуск Spark задач потребує ручного запуску Python script-у. Додаткова інтеграція в систему оркестратора, наприклад Apache Airflow[42], автоматизує роботу Spark обчислень.

```python
def retry(max_consecutive_errors=3, delay=30):
    """
    Retry calling the decorated function which collect executed batches state
    Args:
        max_consecutive_errors: Number of times to try (not retry) before
                                giving up.
        delay: Initial delay between retries in seconds.
    """

    def retry_decorator(f):
        @wraps(f)
        def steps_restart(*args, **kwargs):
            remaining_errors = max_consecutive_errors

            while True:
                try:
                    f(**kwargs)
                    remaining_errors = max_consecutive_errors
                except Exception as e:
                    msg = f'{e}. Retrying in {delay} seconds...'
                    logging.warning(msg)
                    time.sleep(delay)
                    remaining_errors -= 1

                if (remaining_errors < 1 and max_consecutive_errors > 0) or remaining_errors < 0:
                    raise Exception("No errors remaining, failed to start new Spark job")
                elif remaining_errors < 1:
                    break

        return steps_restart
    return retry_decorator
```

Рисунок 5.6 – Імплементація декоратора для перезапуску Spark задач

РОЗДІЛ 6 ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ. ОНОВЛЕННЯ СТАТИСТИЧНИХ ПОКАЗНИКІВ В РЕАЛЬНОМУ ЧАСІ

Покажемо спроможність системи до обробки даних в реальному часі. Оберемо зону JFK Airport, згідно словника зон її LocationID дорівнює 132. На рисунку 6.1 видно, що на початку симуляції в сховищі існують дані про 346138 поїздок, які в цій локації починались. Додамо в Kafka topic новий запис(див. рис. 6.2), бачимо, що PULocationID у цієї поїздки дорівнює 132. На рисунку 6.3 видно, що дані про поїздку з'явились в Kafka topic-у. Рисунок 6.4 демонструє, що дані були взяті в обробку Spark Streaming задачею. Рисунок 6.5 демонструє, що цикл оновлення метрик був успішно виконаний Spark Batch задачею. І, зрештою, на рисунку 6.6 бачимо, оновлене значення кількості поїздок на мапі.

Рисунок 6.1 – Показник кількості поїздок в початковий момент часу

Рисунок 6.2 – Вивід програми про додавання нового запису в Kafka topic

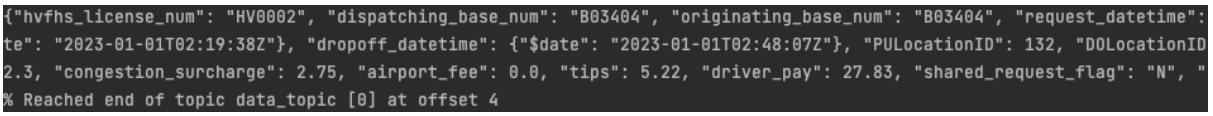

Рисунок 6.3 – Вивід консолі про останні дані, що потрапили в Kafka topic

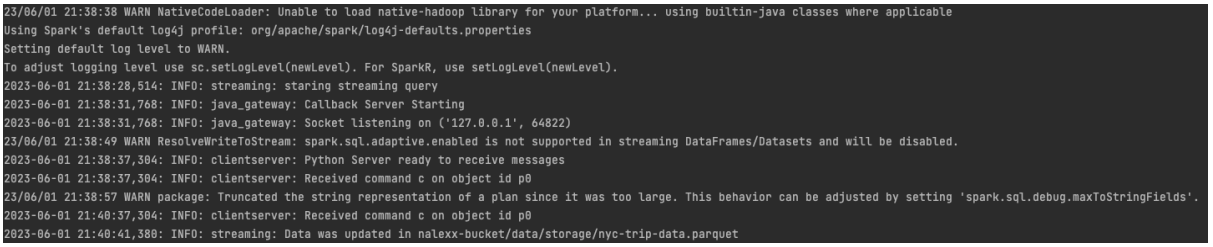

Рисунок 6.4 – Вивід програми про обробку отриманих даних Spark Streaming задачею

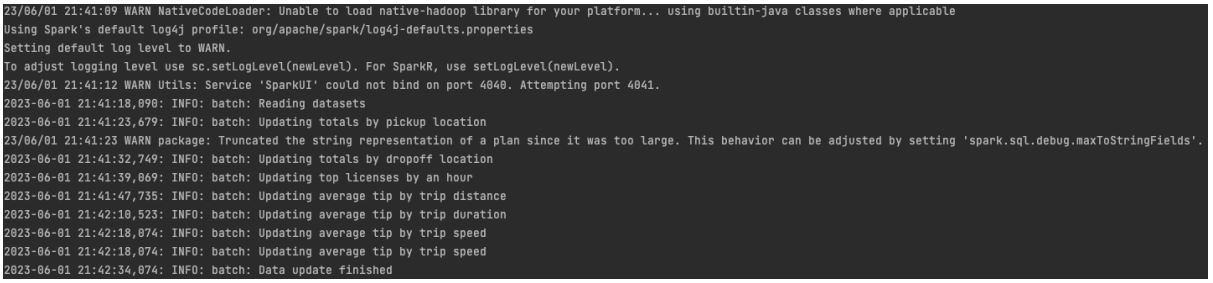

Рисунок 6.5 – Вивід програми про оновлення метрик Spark Batch задачею

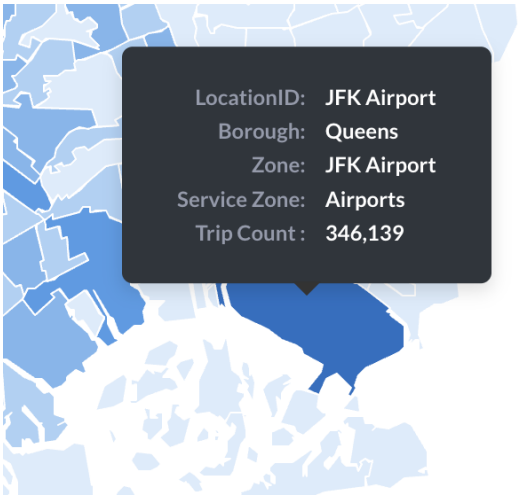

Рисунок 6.6 – Показник кількості поїздок в кінцевий момент часу

ВИСНОВКИ

В рамках виконання роботи було розроблено типову ELT систему, яка дозволяє проводити аналітику даних в реальному часі. Дані надсилаються до потокової системи Kafka, збираються Spark Streaming задачею і зберігаються в сховище даних Amazon S3. Після цього Spark Batch задача агрегує потрібні метричні дані і зберігає їх в базу даних, звідки вони стають доступними Metabase – інструменту бізнес аналітики, для побудови відповідних візуалізацій: графіків, карт, тощо.

Спроєктовано систему, що підтримує автоматичне розгортання всіх її апаратних та програмних компонентів за допомогою CloudFormation шаблонів та CI/CD pipeline-ів Github Actions. CloudFormation шаблон параметризований відносно фізичних характеристик інфраструктури, що надає можливість легкого її масштабування. CI/CD процес реалізований таким чином, що все програмне забезпечення, яке не керується сервісами AWS розгортається за допомогою окремих pipeline-ів і не потребує додаткового ручного налаштування. Також CI/CD процес забезпечує автоматичне оновлення конфігурацій, залежностей та виконуваного коду для Spark задач при коміті в головну гілку репозиторію.

Система була протестована на штучному сценарії, що показав її придатність до обробки та аналітики даних в реальному часі. За допомогою Python script-у було симульовано надходження нових даних до Kafka і було підтверджено успішну обробку цих даних у вигляді зміни метричних показників діаграми. Код “load” складової ELT pipeline-у не залежить від схеми оброблюваної інформації, що робить його уніфікованим для подібних сценаріїв роботи з даними.

Розроблена система може бути адаптована під схожий ELT процес обробки та аналітики даних або використана для ознайомлення з

розподіленим рушієм Spark, ELT парадигмою, великими даними та сервісами платформи AWS. Система може бути покращена за допомогою інтеграції оркестратора, наприклад Apache Airflow, для автоматичного запуску Spark задач.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is Big Data?, Inelliapat Hadoop Tutorial – Complete Hadoop Guide in 2023 [Електронний ресурс]. – Режим доступу: URL: https://intellipaat.com/blog/tutorial/hadoop-tutorial/big-data-overview/?US#no2
2. What Happens in an Internet Minute in 2023: 90 Fascinating Online Stats [Електронний ресурс]. – Режим доступу: URL: https://localiq.com/blog/what-happens-in-an-internet-minute/
3. 3D Data Management: Controlling Data Volume, Velocity, and Variety / Laney Doug / 2001
4. IBM Journal of Research and Development – 2014 – N58 – C.4:1-4:9
5. On the Impact of High Performance Computing in Big Data Analytics for Medicine / Virginia Niculescu / 2020
6. 25+ Impressive Big Data Statistics for 2023 [Електронний ресурс]. – Режим доступу: URL: https://techjury.net/blog/big-data-statistics/
7. 10 Elon Musk Tweets That Created Waves In Crypto World [Електронний ресурс]. – Режим доступу: URL: https://www.outlookindia.com/business/10-elon-musk-tweets-that-created-waves-in-crypto-world-news-233190
8. ETL vs ELT, IBM Cloud Education [Електронний ресурс]. – Режим доступу: URL: https://www.ibm.com/cloud/blog/elt-vs-etl-whats-the-difference
9. Spark: The Definitive Guide / Bill Chambers, Matei Zaharia / O'Reilly Media, Inc, 2018
10. Документація Apache Spark[Електронний ресурс]. – Режим доступу: URL: https://spark.apache.org/docs/latest/

11. Project and product names using "Spark" [Електронний ресурс]. – Режим доступу: URL: https://spark.apache.org/powered-by.html
12. Документація Apache Hadoop 3.3.5 [Електронний ресурс]. – Режим доступу: URL: https://hadoop.apache.org/docs/stable/index.html
13. Apache Spark Officially Sets a New Record in Large-Scale Sorting [Електронний ресурс]. – Режим доступу: URL: https://www.databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html
14. What is MapReduce in Hadoop? Big Data Architecture [Електронний ресурс]. – Режим доступу: URL: https://www.guru99.com/introduction-to-mapreduce.html
15. Introduction to Apache Spark [Електронний ресурс]. – Режим доступу: URL: https://aws.amazon.com/big-data/what-is-spark/
16. Spark Cluster Mode Overview [Електронний ресурс]. – Режим доступу: URL: https://spark.apache.org/docs/latest/cluster-overview.html
17. Документація Apache Hadoop YARN [Електронний ресурс]. – Режим доступу: URL: https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html
18. Документація Kubernetes [Електронний ресурс]. – Режим доступу: URL: https://kubernetes.io/
19. What is cloud computing? A beginner’s guide [Електронний ресурс]. – Режим доступу: URL: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-cloud-computing
20. Top 10 biggest cloud providers in the world in 2023 [Електронний ресурс]. – Режим доступу: URL: https://technologymagazine.com/top10/top-10-biggest-cloud-providers-in-the-world-in-2023

21. Move Big Data To The Public Cloud With An Insight PaaS / Brian Hopkins. – Forrester blog, 2017 [Електронний ресурс]. – Режим доступу: URL: https://www.forrester.com/blogs/insight-paas-accelerate-big-data-cloud/
22. What is Infrastructure as Code (IaC)? [Електронний ресурс]. – Режим доступу: URL: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac
23. Cloud computing with AWS [Електронний ресурс]. – Режим доступу: URL: https://aws.amazon.com/what-is-aws/
24. AWS Free Tier [Електронний ресурс]. – Режим доступу: URL: https://aws.amazon.com/free/
25. Документація Amazon S3 [Електронний ресурс]. – Режим доступу: URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
26. Документація Amazon EMR [Електронний ресурс]. – Режим доступу: URL: https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-overview.html
27. Документація Amazon EC2[Електронний ресурс]. – Режим доступу: URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html
28. TLC Trip Record Data [Електронний ресурс]. – Режим доступу: URL: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page
29. Документація Apache Kafka [Електронний ресурс]. – Режим доступу: URL: https://kafka.apache.org/documentation/
30. Документація Amazon RDS [Електронний ресурс]. – Режим доступу: URL:

https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html

31. Документація Metabase [Електронний ресурс]. – Режим доступу: URL: https://www.metabase.com/docs/latest/
32. Документація Apache Parquet [Електронний ресурс]. – Режим доступу: URL: https://parquet.apache.org/docs/overview/
33. Документація Docker [Електронний ресурс]. – Режим доступу: URL: https://docs.docker.com/get-started/
34. Документація Amazon CloudFormation [Електронний ресурс]. – Режим доступу: URL: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html
35. Документація бібліотеки boto3 [Електронний ресурс]. – Режим доступу: URL: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
36. Документація GitHub Actions [Електронний ресурс]. – Режим доступу: URL: https://docs.github.com/en/actions
37. Репозиторій з кодом розробленої системи [Електронний ресурс]. – Режим доступу: URL: https://github.com/Nalexx6/Diploma
38. Data Dictionary – High Volume FHV Trip Records [Електронний ресурс]. – Режим доступу: URL: https://www.nyc.gov/assets/tlc/downloads/pdf/data_dictionary_trip_records_hvfhs.pdf
39. Карта зон таксі Нью-Йорку у форматі geojson: [Електронний ресурс]. – Режим доступу: URL: https://maps.princeton.edu/catalog/nyu-2451-36743
40. Документація Apache Avro [Електронний ресурс]. – Режим доступу: URL: https://avro.apache.org/docs/ (переглянуто 27.05.2023)

41. Документація Delta Lake format [Електронний ресурс]. – Режим доступу: URL: https://docs.delta.io/latest/delta-intro.html
42. Документація Apache Airflow [Електронний ресурс]. – Режим доступу: URL: https://airflow.apache.org/docs/apache-airflow/stable/index.html