НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проектування

До захисту допущено:

Завідувач кафедри

_____ Вадим МУХІН

«___»_____2022 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою

“Інтелектуальні сервіс-орієнтовані розподілені обчислювання”

зі спеціальності 122 "Комп'ютерні науки"

на тему: «Використання сервіс-орієнтованої архітектури при обробці великих об’ємів даних»

Виконав:
студент IV курсу, групи ДА-82
Муравльов Андрій Дмитрович _____

Керівник:
доцент, к.т.н.
Булах Богдан Вікторович _____

Консультант з економічного розділу:
доцент, к.е.н.
Рощина Надія Василівна _____

Рецензент:
д.т.н., проф.
Бідюк Петро Іванович

_____

Засвідчую, що у цій дипломній роботі
немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2022

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проектування

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 122 "Комп'ютерні науки"

Освітньо-професійна програма – "Інтелектуальні сервіс-орієнтовані розподілені обчислювання"

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Вадим МУХІН

«___»_____2022 р.

ЗАВДАННЯ

на дипломну роботу студенту

Муравльову Андрію Дмитровичу

1. Тема роботи «Використання сервіс-орієнтованої архітектури при обробці великих об’ємів даних», керівник роботи Булах Богдан Вікторович, кандидат технічних наук, доцент, затверджені наказом по університету від «06» червня 2022 р. № 906-с

2. Термін подання студентом роботи – 15 червня 2022 р.
3. Вихідні дані до роботи

Середовища розробки MS Visual Code, аккаунт в Google Cloud Platform, онлайн література та статті щодо розробки сервіс-орієнтованих додатків та обробки великих даних.

4. Зміст роботи: Огляд парадигми сервіс-орієнтованої архітектури, її застосування при роботі з великими даними, дослідження хмарних рішень, що надають сервіси для роботи з великими даними. Розробка додатку, що опрацьовує великі дані.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо) : діаграми опису архітектури, знімки екрану етапів побудови програмного рішення.

6. Консультанти розділів роботи[1]*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Рощина Н.В.		

7. Дата видачі завдання 02.01.2022

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	02.01.2022	
2	Аналіз літературних джерел по використанню СОА при роботі з великими даними	02.05.2022	
3	Пошук відкритих датасетів	09.05.2022	
4	Дослідження інструментальних засобів роботи з великими даними, що орієнтовані на СОА	16.05.2022	
5	Дослідження концепції управління потоками даних та оркестрування сервісів	23.05.2022	
6	Реалізація додатку, що оброблює великі дані з використанням СОА	03.06.2022	
7	Оформлення дипломної роботи	12.06.2022	
8	Отримання допуску до захисту та подача роботи в ДЕК	20.06.2022	

Студент Муравльов А. Д.

Керівник Булах Б. В.

[1]* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

АНОТАЦІЯ

до бакалаврської дипломної роботи Муравльова Андрія Дмитровича

на тему: «Використання сервіс-орієнтованої архітектури при обробці великих об’ємів даних»

Дана робота присвяченна дослідженню принципів роботи з великими даними орієнтованих на використання сервіс-орієнтованої архітектури.

У роботі проведено загальний огляд та описано механізми взаємодії, обробки та отримання даних в Big Data, в тому числі класичних рішень та рішень, орієнтованих на використання сервіс-орієнтованої архітектури.

Розглянуто найбільш поширені провайдери хмарних провайдерів та сервісів, що вони надають, які використовуються при роботі з великими даними — Microsoft Azure, Amazon Web Services, Google Cloud Platform.

Реалізовано додаток, що працює з великими даними. В додатку реалізовані різні методи обробки та взаємодії з великими даними.

Дана робота та її результати буде корисна для розробників та проектів, які натрапляють на швидкий ріст об’єму даних, що зберігаються та надходять до продукту.

Загальний обсяг роботи: 89 с., 38 рис., 6 таблиць, 21 джерело.

Ключові слова: Google Cloud Platform, Service-Oriented Architecture, BaSOA, AaaS, SaaS, FaaS, провайдери хмарних сервісів, Великі Дані, Enterprise Service Bus.

ABSTRACT

for the bachelor's thesis of Muravlov Andrii Dmytrovych

on the topic: «Use of service-oriented architecture in big data processing»

This paper investigates the principles of Big Data, oriented on the use of service-oriented architecture.

The paper provides a general overview and describes the mechanisms of interaction, processing and receiving data in Big Data, including classic solutions and solutions oriented to the use of service-oriented architecture.

The most common cloud providers and their services used when working with Big Data are considered - Microsoft Azure, Amazon Web Services, Google Cloud Platform.

An application that works with big data has been implemented. The application implements different methods to process and interact with big data.

This work and its results will be useful for developers and projects that fall on the rapid growth of the amount of data stored and coming into the product.

Overall scope of work:89 pages, 38 figures, 6 tables, 21 sources.

Keywords: Google Cloud Platform, Service-Oriented Architecture, BaSOA, AaaS, SaaS, FaaS, Cloud Service Providers, Big Data, Enterprise Service Bus.

ЗМІСТ

Перелік умовних позначень, символів і скорочень

SQL — Structured Query Language

AWS — Amazon Web Services

GCP — Google Cloud Platform

REST — Representational state transfer

СОА — Сервіс-орієнтована архітектура

ESB — Enterprise Service Bus

ВСТУП

В наш час, як ніколи раніше, ми зіштовхнулися з неймовірним ростом розміру та потоку даних, що генеруються кожного дня. Генерація даних пережила деякий «вибух», викликаний, у тому числі, розповсюдженням інтернету по всьому світу та все більшою доступністю до інформаційних технологій. Кількість даних, які людство генерує кожен день за різними оцінками досягає в середньому 2.5 ЕБ (Ексабайт) або 2.5 мільйони терабайт [1].

Це, в свою чергу, призвело до зміни парадигми сприйняття даних — компанії, медики, уряди, журналісти вже розглядають дані не як побічний продукт своєї діяльності, а навпаки як головний актив, що дозволяє отримати уявлення про потреби користувачів, чи інших зацікавлених сторін, та про ефективність задоволення цих потреб.

Через це виникає потреба обробки та зберігання потоків великих даних найбільш швидко та ефективно. Існує багато парадигм та методів роботи з великими даними, у кожного є свої переваги та свої недоліки. Сервіс-орієнтована архітектура за своїм визначенням дозволяє повторне використання сервісів або додатків, достатньо просто масштабується, фокусується на використанні в бізнес функціоналі та надає стандарти комунікації.

Тому, метою даної роботи стало порівняння нині існуючих парадигм роботи з великими даними, порівняння їх з сервіс-орієнтованою архітектурою, розгляд методів взаємодії з великими даними в цій парадигмі та реалізація програмного рішення з використанням сервіс-орієнтованої архітектури.

РОЗДІЛ 1. Великі дані та парадигми роботи з ними

Великі дані (англ. Big Data) загалом описується як сукупність наборів даних, настільки великих та комплексних, що традиційне програмне забезпечення та системи баз даних не можуть опрацювати їх протягом потрібного часу [5]. Наприклад, зберігання та обробка дописів у Twitter/Reddit/Facebook потребують значних обчислювальних потужностей на процеси зберігання та обробки даних. В такі процеси може входити, наприклад, пошук кореляції між мільйонами дописів чи аналіз демографічних даних користувачів.

Хоча традиційні бази даних на основі SQL довели свою високу ефективність, надійність та послідовність у зберіганні та обробці структурованих (або реляційних) даних, вони не справляються з обробкою великих даних, які характеризуються великим обсягом, різноманітністю, швидкістю, відкритістю, невідповідною структурою та візуалізацією серед інших [2].

Великі дані будуть відігравати важливу роль у різних галузях, таких як наука, дослідження, інженерія, медицина, охорона здоров'я, фінанси, бізнес та, зрештою, структурі самого суспільство [3]. Вони можуть бути використані для аналізу та прогнозування тенденцій, прибутку та збитків у бізнесі, визначення дорожньої обстановки в режимі реального часу, охорони здоров'я, інформації про погоду тощо.

Великі дані, як правило, характеризуються трьома показниками: різноманітністю, обсягом та швидкістю. Різноманітність відноситься до природи та структури інформації, що становить великі дані. Швидкість відноситься до частоти генерування даних, і навіть динамічні аспекти даних. Різноманітність відноситься до багаторежимної природи даних, наприклад різні схеми джерел даних, структуровані дані, такі як онтології, та неструктуровані дані, такі як сигнали датчиків [4].

Обробка, наявність або отримання великих даних може бути класифікована за різними категоріями, включаючи пакетну обробку, обробку в реальному часі та гібридну обробку. Пакетна обробка - це ефективний спосіб обробки великих обсягів даних, які збираються протягом певного періоду. У цій схемі дані збираються, зберігаються/вставляються в джерела даних та обробляються. Потім видаються результати пакетної обробки. Проте деякі Однак деякі програми вимагають обробки даних (потоків) у реальному часі, які надходять із різнорідних джерел даних. Обробка в реальному часі передбачає безперервне введення, обробку та виведення даних [4]. Низька затримка є основною метою цієї парадигми обробки. Тобто дані мають бути оброблені за невеликий (або близький до реального) періоду. Області застосування включають розумні міста, розваги та управління стихійними лихами.

1.1. Властивості великих даних

Великі дані мають великий обсяг, високу швидкість та містять різноманітну інформацію, яка вимагають нових методів управління та обробки, щоб забезпечити більш ефективне прийняття рішень, прогнозування, бізнес-аналізу, підвищення якості обслуговування та лояльності клієнтів, а також оптимізації процесів у різних організаціях, галузях та онлайнових соціальних мережах. Традиційне програмне забезпечення не може керувати даними великого обсягу, швидкості та різноманітності (3Vs) [6]:

- Обсяг (volume) відноситься до розміру даних, що підлягають обробці. Обсяг Великих даних виходить далеко за традиційні межі мегабайтів або гігабайтів і досягає терабайтів або навіть петабайтів.
- Швидкість (velocity) відноситься не тільки до частоти генерації даних, але і динамічних аспектів даних, а також до необхідності генерувати результати у режимі реального часу.

– Різноманітність (variety) відноситься до багаторежимної природи даних. Тобто, різні джерела інформації та різні схеми даних кожного джерела, наприклад, структуровані дані, такі як онтології, та неструктуровані дані, такі як сигнали датчиків.

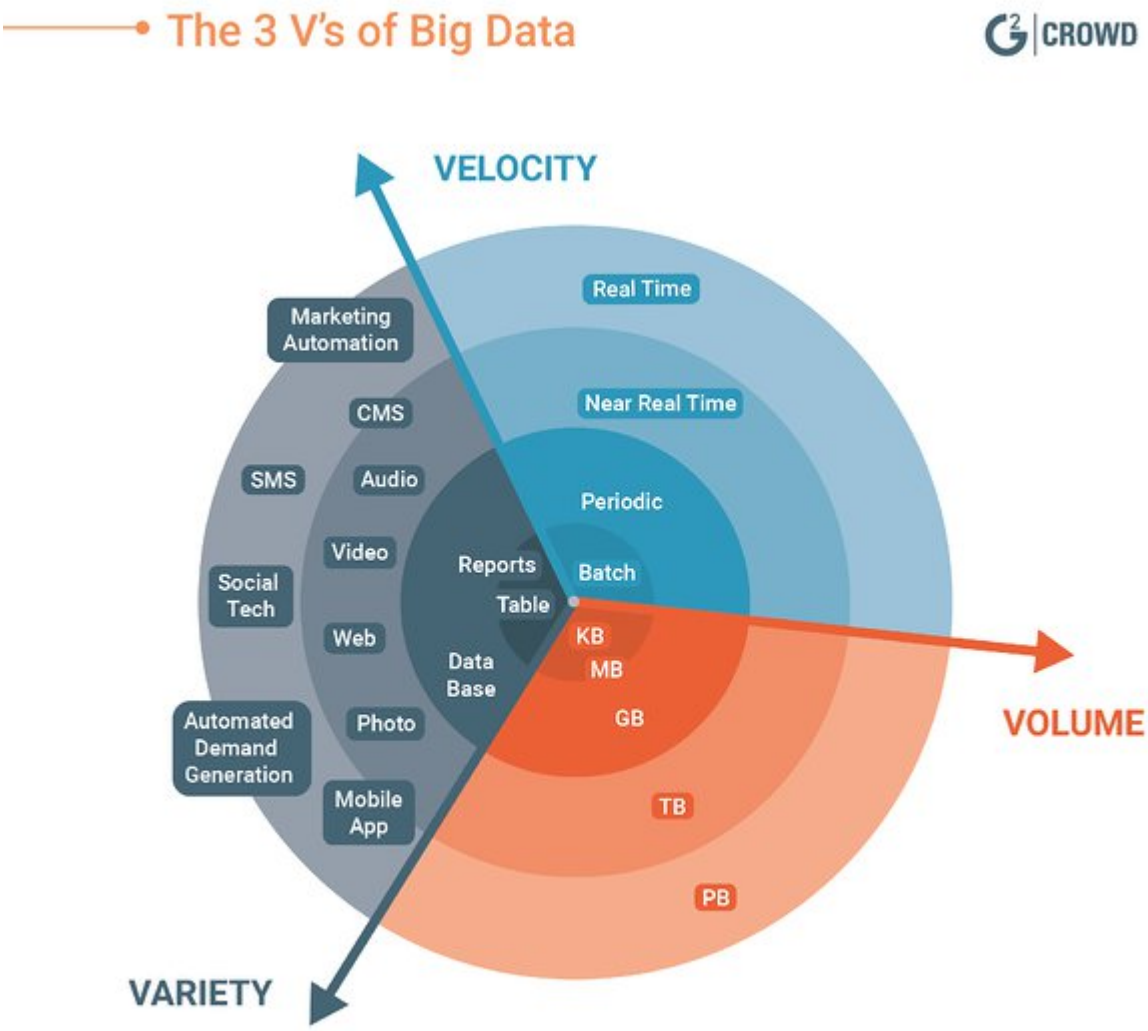


Рисунок 1.1. 3V великих даних [7]

Великі дані мають потенціал стати основним інструментом прийняття рішень, проникнувши у всі сфери нашого сучасного суспільства, включаючи роздрібну торгівлю, виробництво, охорону здоров'я, економіку, фінанси, науки та навколишнього середовища, дорожнього руху, погоди та інші. У таких областях щодня генерується величезний щодня генерується величезний

обсяг даних, наприклад, потоки даних від мережі датчиків, онлайн-дописи та обговорення, дані про навколишнє середовище, погоду тощо. огляди та обговорення, екологічні дані, дані про погоду та моніторинг дорожнього руху. Усвідомлюючи переваги та важливість Великих Даних, різні розробки та дослідницькі проекти були ініційовані промисловістю, академічними інститутами та урядовими організаціями.

1.2. Парадигми обробки великих даних

1.2.1. Пакетна обробка (Batch processing)

Пакетна обробка — це рішення для обробки великих об’ємів статичних даних, тобто тих даних, що вже знаходяться в системі. Ця парадигма не враховує нові дані, які надходять після початку пакетної обробки.

Головна особливість обробки системи пакетної обробки даних є висока масштабованість, для досягнення якої обробка використовує розподілену обробку, таку як фреймворк MapReduce.

MapReduce є стандартною технологією для, фактично, пакетної обробки. Вона має ряд переваг:

- Наявність функціоналу створення спрощеного та уніфікованого виду даних;
- Є масштабованою за своєю сутністю;
- Виносить механізми перетворення та обробки даних від кінцевого користувача, що дозволяє нехтувати неоднорідністю якості мережі та пристроїв.

MapReduce також має деякі обмеження та стримуючі фактори в певних умовах, які необхідно враховувати. Наприклад, багато завдань аналізу/обробки в системах або додатках реального часу повинні виконуватися ітеративно або кілька заходів. Це важко зробити в MapReduce.

Крім того, для ефективного аналізу даних необхідні додаткові розробки для потокових обчислень у реальному часі, а також оптимізація доступу до даних та індексування.

У цілому нині, парадигма пакетної обробки надійніша, але пакетні обчислення можуть займати більше часу. Таким чином, вони не підходять для програм з низькою затримкою. Крім того, пакетну обробку не можна перервати або змінити конфігурацію "на льоту" при надходженні нових даних. Нині пакетний аналіз великих даних застосовується у додатках соціальних мереж, графовому аналізі, наукових додатках та інших.

1.2.2. Обробка в режимі реального часу

Головна мета парадигми обробки в реальному часі — впоратися зі швидкістю великих даних, наприклад обробити потокові дані, але з низькою затримкою.

Ця парадигма обробки ґрунтується на більш-менш тих же принципах, що й пакетна обробка, таких як розподіл та паралелізм. Для досягнення низької затримки ця парадигма обробки аналізує невеликі набори даних, що зберігаються у пам'яті та відсіює дані які мають схожі показники. Таким чином, обробка в реальному часі являє собою щось подібне до нескінченної послідовності невеликих пакетних обробок, де інформація знаходиться в пам'яті, а не вторинних сховищах - іншими словами, використовується бездисковий підхід.

Прикладом обробки в реальному часі є визначення поточних або трендових тем у Twitter. Деякі програми вимагають обробки реального часу потоків даних з різнорідних джерел. Як приклади можна навести такі: Розумні міста - управління транспортом, енергопостачанням та прибиранням сміття; управління стихійними лихами - особливо за допомогою даних, зібраних із джерел управління надзвичайними ситуаціями, використання громадянами соціальних мереж та мобільних пристроїв; виробництво та логістика -

використання датчиків заводів для контролю якості, оптимізації продукції та економії ресурсів; розваги - аналіз потокових даних із музичних, телевізійних та ігрових платформ для рекомендацій, аналізу користувачів та реклами.

1.2.3. Гібридне обчислення

Багато програм вимагають поєднання парадигм обробки даних у реальному часі. Це досягається за допомогою гібридної моделі. Ця модель також відома як архітектура Lambda [8], яка складається з:

Пакетного шару (пакетна обробка) – обробляє основний набір даних, який не змінюється та зберігається в розподіленій файловій системі; сервісного шару (результати пакетної обробки) - завантажує пакетні уявлення у сховищі даних та робить їх доступними для пошуку;

Швидкісного шару (обробка в реальному часі) - обробляє тільки нові дані з низькою затримкою.

Щоб отримати повний результат, необхідно отримати пакетне представлення та подання у реальному часі та об'єднати результати. Синхронізація, компіляція результатів та інші нетривіальні питання вирішуються цьому етапі, що є частиною шару Combine.

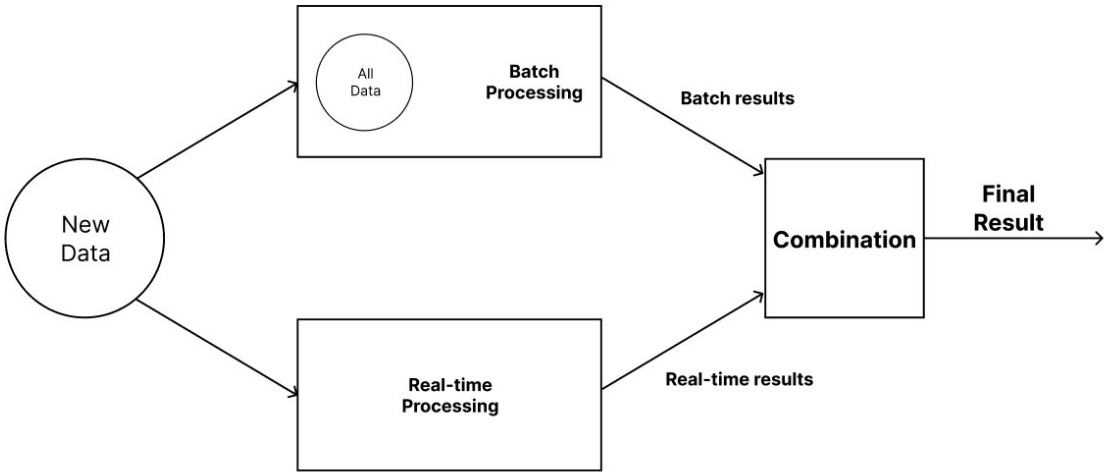


Рис 1.2. Гібридна модель обробки великих даних

На рисунку 1.2. показано високорівневу архітектуру моделі гібридної обробки [8]. У цій моделі є три шари: пакетна обробка, обробка в реальному часі та комбінована обробка. Нові дані дублюються та відправляються в пакетний шар та шар реального часу. Пакетний рівень обробляє весь набір даних за цикл. Однак пакетна робота займає багато часу, тому в процесі може надходити нова інформація, яка не враховується шаром накопичувача. Щоб компенсувати цю затримку, шар реального часу обробляє ті нові дані, які ще були проаналізовані шаром пакетної обробки. Окремі рівні зберігають свої часткові результати у базі даних, яка вивчається агрегованим рівнем для отримання остаточних, оновлених результатів у режимі реального часу.

Висновки до розділу 1

Великі дані є досить новою сферою в інформаційних технологіях, але вже можна зробити висновок про їх важливість у сучасних обчисленнях, підходах до розробки, аналітики продукту та їх цільових аудиторій, перспективах для повсякденного життя, медицини, містобудівництва і т.д.

Виходячи з визначення Big Data, можна сформулювати основні принципи роботи з такими даними [20]:

1. Горизонтальна масштабованість. Оскільки даних може бути скільки завгодно багато – будь-яка система, яка передбачає обробку великих даних, має бути розширюваною. У 2 рази зріс обсяг даних – у 2 рази збільшили кількість заліза у кластері та все продовжило працювати.

2. Відмовостійкість. Принцип горизонтальної масштабованості передбачає, що машин у кластері може бути багато. Це означає, що частина цих машин гарантовано виходитиме з ладу. Методи роботи з великими даними повинні враховувати можливість таких збоїв і переживати їх без будь-яких значущих наслідків.

3. Локальність даних. У великих розподілених системах дані розподілені за великою кількістю машин. Якщо дані фізично перебувають у одному сервері, а обробляються іншому – витрати на передачу даних можуть перевищити витрати на саму обробку. Тому одним із найважливіших принципів проектування BigData-рішень є принцип локальності даних - по можливості обробляємо дані на тій же машині, на якій їх зберігаємо.

Парадигма сервіс-орієнтованої архітектури дозволяє цих принципів дотримуватись та дає ряд своїх переваг, наприклад СОА дозволяє більшість обчислень та сервісів винести у хмарні сховища, що дозволить сервісам горизонтально масштабуватись без потреби закупки нових кластерів зі збільшенням об’єму даних, а також гарантує набагато більшу відмовостійкість.

РОЗДІЛ 2. Сервіс-орієнтована архітектура та її застосування у Big Data

1.1. Властивості сервіс-орієнтованої архітектури

Сервіс-орієнтована архітектура (далі СОА) — архітектурний шаблон програмного забезпечення для роботи з неоднорідними, динамічними, невизначеними та необмеженими інженерними системами.

СОА фокусується на:

- Сервісах понад властивостями компонентів
- Сумісності та кроссплатформеності
- Розподіленості та гнучкості
- Високому рівні абстракції

СОА використовується для інтеграції компонентів, що широко розходяться в функціоналі, надаючи їм спільні інтерфейси і набір протоколів, щоб ці компоненти могли комунікувати через сервісну шину (Service bus).

Багато даних які надходять з різноманітних джерел та сервісів неопрацьовані та не потрібні кінцевому юзеру, або у випадку збору датасетів для машинного навчання мають бути додатково опрацьовані щоб вони підходили під імплементацію моделі навчання. Для цього дані інкапсулюються, оскільки в обох випадках лише провайдер сервісу має знати логіку компоненту/моделі/тощо.

Не дивлячись на те що початкові дані інкапсульовані та не мають значення для кінцевого користувача, користувач має знати функцію яку дані виконують та як з ними взаємодіяти. Для цього в СОА імплементуються інтерфейси, що описують поля.

Сервіси використовують стандартизовані контракти взаємодії, визначені одним, або більше документами, що описують сервіси.

Кожен сервіс є незалежною одиницею і майже не залежить від інших сервісів, тобто будь який сервіс може приєднуватися або від’єденуватися від системи зберігаючи працездатність системи та інших сервісів та дозволяючи динамічне надання ресурсів та даних для збереження ефективного балансування навантаження.

Кожен сервіс в СОА є реюзабельним і незалежним від будь-якого іншого процесу до якого він під’єднується, що дозволяє сервісам бути використаними знову, зібраними та розібраними до різних конфігурацій.

1.2. СОА при роботі з великими даними

Протягом тривалого часу великі дані та пов’язана з ними аналітика були структуровані як єдина канонічна форма, яка включає в себе величезні окремі схеми, щоб охопити все [10]. Це жорсткий і монолітний підхід, який має на меті стандартизувати все на підприємстві. Навпаки, SOA розбиває цю жорсткість і монолітність підходів до систем і розділяє життєво важливі компоненти на частини послуг.

Тепер, враховуючи поточні потреби бізнесу, це дає змогу стати доступними для аналітики, щойно вони будуть створені. Крім того, немає потреби мати все стандартизоване на всьому підприємстві; натомість важливі лише ті спільні сегменти даних.

Крім того, структура SOA дозволяє кінцевим користувачам гнучко створювати власні інтерфейси для доступу до необхідної інформації, не чекаючи, поки IT створить її для них. Це має сенс, щоб великі дані були доступні також як сервісно-орієнтована архітектура або BaSOA (Big Data analytics SOA).

1.2.1. Компоненти СОА при роботі з великими об’ємами даних

Існує три основних компоненти в сервіс-орієнтованій архітектурі аналітики великих даних:

- Центральний юрокер послуг аналітики великих даних
- Запитувач послуг аналітики великих даних
- Постачальник послуг аналітики великих даних
- Оркестратори
- Шина сервісів

Брокери послуг аналітики великих даних – це організації, що сприяють розвитку послуг аналітики великих даних, до яких належать:

- Популярні видання
- Соціальні медіа
- Традиційні ЗМІ
- Консалтингові компанії
- Студенти університетів
- Вчені

і т.д.

Використання різних методів та прийомів допомагає краще зрозуміти послуги з аналітики великих даних загалом. Також надається огляд аналітики даних, бізнес-аналітики, веб-аналітики та послуг.

Замовники послуг з аналізу великих даних – це організації, до яких належать:

- Організації
- Уряди
- Особи, які приймають бізнес-рішення на всіх рівнях, такі як CEO, CIO та CFO, а також менеджери.
- Інформаційні системи для бізнесу

- Системи електронної комерції

Замовникам послуг з аналізу великих даних потрібні послуги з аналізу великих даних, які включають послуги інформаційної аналітики послуги з аналізу знань послуги бізнес-аналітики разом із методами візуалізації, вищезгадані послуги надають шаблони знань та інформацію для прийняття рішень. До послуг аналітики великих даних зазвичай входять особи, які приймають рішення, які отримують інформацію на основі аналітичних звітів, що надаються провайдерами послуг аналітики великих даних.

Провайдери послуг з аналітики великих даних включають:

- Розробників у сфері аналітики
- Постачальники аналітики
- Аналітичні системи або програмне забезпечення та інші посередники
- Провайдери послуг веб-аналітики (WAS).

Вищезгадані організації можуть надавати аналітичні послуги. Це може сприяти прийняттю стратегічних бізнес-рішень. Розробники аналітичних систем надають аналітичні інструменти з широкими можливостями вилучення даних, аналітики та звітності. Google є постачальником пошукових систем та постачальником WAS, оскільки Google надає Google Analytics, аналітику великих даних, з хорошими інструментами відстеження. Наприклад, Mobile App Analytics, частина Google Analytics, також є постачальником послуг аналізу великих даних для мобільних пристроїв, який допомагає клієнтам смартфонів виявляти нових та релевантних користувачів за допомогою звітів про джерела трафіку. Серед постачальників послуг аналізу великих даних в Інтернеті - Amazon, Google і Microsoft.

Оркестратори — це набір сервісів, що надають функціонал оркестрування сервісами.

Оркестрування сервісів - це управління певних бізнес-процесів веб-сервісів центральним контроллером [15].

Контролер, який також може бути веб-сервісом, координує асинхронні взаємодії, управління потоками, управління бізнес-транзакціями та моніторинг бізнес-процесів. Нотація моделювання бізнес-процесів (BPMN) використовується для визначення візуального представлення потоку, а мова виконання бізнес-процесів (BPEL) використовується для написання коду, який виконує служби.

Оркестрування сервісів грає важливу роль у сервіс-орієнтованій архітектурі (SOA).

Шина сервісів (англ. Enterprise Service Bus або ESB) — це програмна архітектура, яка об'єднує всі сервіси разом через шиноподібну інфраструктуру [16]. Вона функціонує як комунікаційний центр в SOA, дозволяючи підключати кілька систем, доданих і даних, а також об’єднує кілька систем без сбоїв.

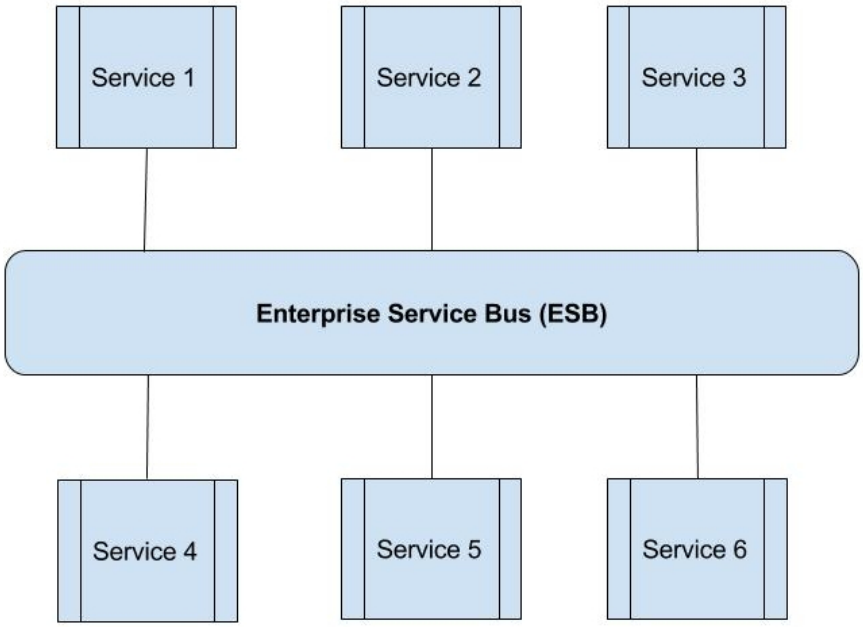


Рисунок 2.1. Представлення ESB

На рисунку 2.1. зображено зв'язок між програмними додатками в сервіс-орієнтованій архітектурі за допомогою ESB. Шина - це комунікаційна система, яка передає дані між комп'ютерами і об'єднує між собою жорсткі диски, CD-ROM, графічні адаптери та інші мікросхеми.

ESB може виконувати ролі менеджера транзакцій, безпеки, проксі, шлюзу, тощо, використовуючи певне проміжне програмне забезпечення, однак основний принцип роботи та взаємодії сервісів з шиною залишається незмінним.

Висновки до розділу 2

У розділі розглянуто парадигму сервіс-орієнтованої архітектури, її властивості та переваги. Можливість повторно використовувати сервіси та інкапсулювати дані що надходять до умовного додатку та повертаються користувачу дозволяє позбавити користувача від непотрібної інформації та затрат обчислювальної потужності.

Розглянуто застосування сервіс-орієнтованої архітектури при роботі з великими даними, основні компоненти такої взаємодії, серед яких особливо важливими є провайдери послуг з аналітики великих даних. Більш детально розглянемо переваги та недоліки найпоширеніших провайдерів після огляду інструментальних засобів роботи з великими даними.

РОЗДІЛ 3. Інструментальні засоби роботи з великими даними, орієнтовані на СОА

3.1. Map Reduce

MapReduce – це модель розподіленої обробки даних, запропонована Google для обробки великих обсягів даних на комп'ютерних кластерах [17].

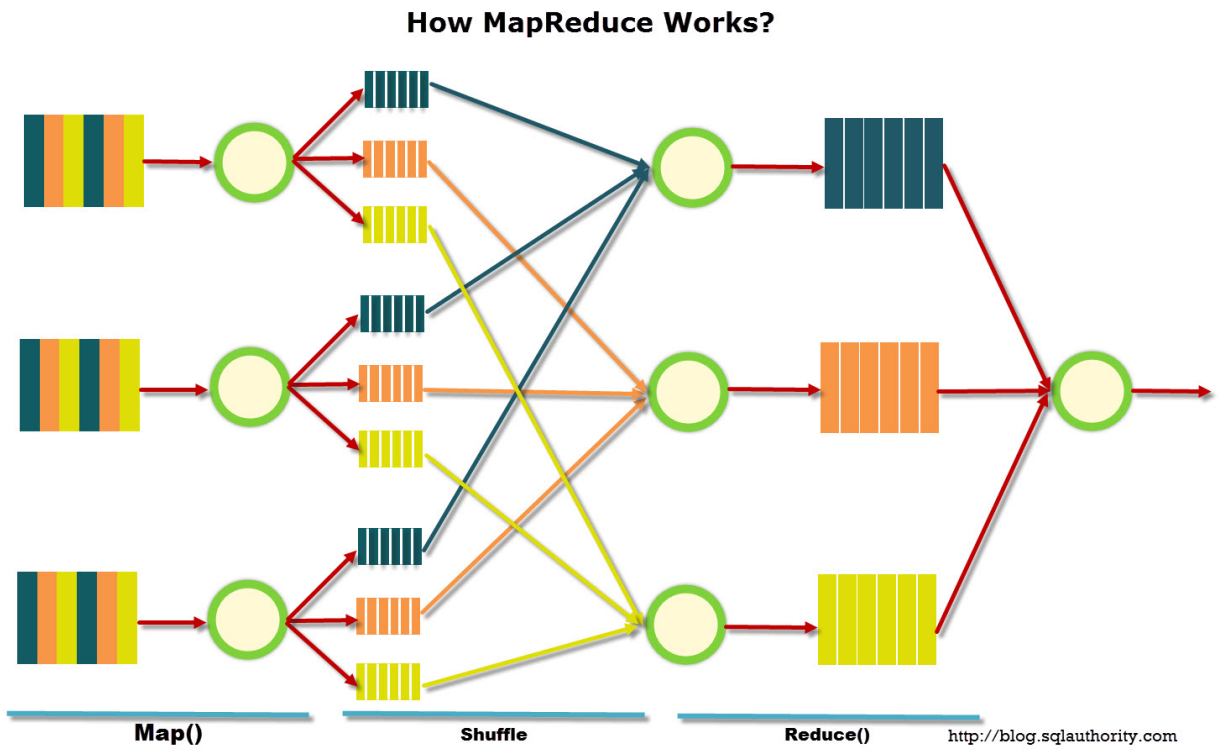


Рис. 3.1. Механізм роботи MapReduce

MapReduce передбачає, що дані організовані як деяких записів. Обробка даних відбувається у 3 стадії [21]:

1. Стадія Map. У цій стадії дані передобробляються з допомогою функції map(), яку визначає користувач. Робота цієї стадії полягає у передобробці та фільтрації даних. Робота дуже схожа на операцію map у функціональних мовах програмування – функція користувача застосовується до кожного вхідного запису.

Функція map() застосована до одного вхідного запису та видає безліч пар ключ-значення. Безліч – тобто. може видати лише один запис, може не

видати нічого, а може видати кілька пар ключ-значення. Що буде в ключі і в значенні – вирішувати користувачеві, але ключ – дуже важлива річ, оскільки дані з одним ключем у майбутньому потраплять до одного примірника функції reduce.

2. Стадія Shuffle. Проходить непомітно для користувача. У цій стадії виведення функції map "розбирається по кошиках" - кожен кошик відповідає одному ключові виведення стадії map. Надалі ці кошики послужать входом до reduce.

3. Стадія Reduce. Кожен «кошик» із значеннями, сформований на стадії shuffle, потрапляє на вхід функції reduce().

Функція reduce задається користувачем та обчислює фінальний результат для окремого «кошика». Багато всіх значень, повернутих функцією reduce(), є фінальним результатом MapReduce-завдання.

Декілька додаткових фактів про MapReduce:

1) Всі запуски функції map працюють незалежно і можуть працювати паралельно, у тому числі на різних кластерних машинах.

2) Всі запуски функції reduce працюють незалежно і можуть працювати паралельно, у тому числі на різних кластерних машинах.

3) Shuffle уявляє паралельне сортування, тому також може працювати на різних машинах кластера. Пункти 1-3 дозволяють виконати принцип горизонтальної масштабованості.

4) Функція map, зазвичай, застосовується тієї ж машині, де зберігаються дані – це дозволяє знизити передачу даних у мережі (принцип локальності даних).

5) MapReduce – це завжди повне сканування даних, жодних індексів немає. Це означає, що MapReduce погано застосовується, коли відповідь потрібна дуже швидко.

3.2. SOAP веб-сервіси

SOAP (Simple Object Access Protocol) – це специфікація протоколу обміну повідомленнями для обміну структурованою інформацією під час реалізації веб-сервісів у комп'ютерних мережах. Він використовує XML Information Set для формату повідомлень і покладається на протоколи прикладного рівня, найчастіше Hypertext Transfer Protocol (HTTP), хоча деякі застарілі системи обмінюються повідомленнями через Simple Mail Transfer Protocol (SMTP) для узгодження та передачі повідомлень.

SOAP дозволяє розробникам викликати процеси, запущені в різних операційних системах (таких як Windows, macOS і Linux), для автентифікації, авторизації та обміну даними з використанням мови розмітки (XML), що розширюється. Оскільки веб-протоколи, такі як HTTP, встановлені та працюють на всіх операційних системах, SOAP дозволяє клієнтам викликати веб-служби та отримувати відповіді незалежно від мови та платформ.

У світі існує величезна кількість додатків, створених різними мовами програмування. Наприклад, веб-додаток може бути розроблений на Java, інший - на .Net, а третій - на PHP.

Обмін даними між програмами має вирішальне значення в сучасному мережному світі. Але обмін даними між цими різнорідними програмами буде складним. Також складним буде код для здійснення цього обміну даними.

Одним із методів боротьби з цією складністю є використання XML (Extensible Markup Language) як проміжна мова для обміну даними між додатками.

Кожна мова програмування може розуміти мову розмітки XML. Отже, XML був використаний як базове середовище для обміну даними.

Але не існує стандартних специфікацій щодо використання XML у всіх мовах програмування для обміну даними. Саме тут на допомогу приходить програмне забезпечення SOAP.

SOAP був розроблений для роботи з XML за протоколом HTTP і має певну специфікацію, яку можна використовувати у всіх програмах. Докладніше ми розглянемо протокол SOAP у наступних розділах.

3.2.1. Переваги SOAP

SOAP - це протокол, який використовується для обміну даними між програмами. Нижче наведено деякі причини, з яких використовується SOAP.

- Під час розробки веб-служб на основі SOAP необхідно мати деяку мову, яка може бути використана для спілкування веб-служб із клієнтськими програмами. SOAP є ідеальним засобом, який був розроблений для досягнення цієї мети. Цей протокол також рекомендований консорціумом W3C, який є керівним органом для всіх веб-стандартів.
- SOAP - це легкий протокол, який використовується для обміну даними між програмами. Зверніть увагу на ключове слово "легкий". Оскільки програмування SOAP засноване на мові XML, яка сама собою є легковажною мовою обміну даними, отже, SOAP як протокол також потрапляє в цю ж категорію.
- SOAP розроблено так, щоб бути незалежним від платформи, а також від операційної системи. Таким чином, протокол SOAP може працювати з будь-якими програмами на основі мови програмування як на платформі Windows, так і на платформі Linux.
- SOAP працює на протоколі HTTP, який є протоколом за замовчуванням, що використовується всіма веб-програмами. Отже, не існує жодної

установки, яка потрібна для запуску веб-сервісів, побудованих на протоколі SOAP, для роботи у Всесвітній мережі.

3.2.2. Структура даних в SOAP

Слід зазначити, що SOAP повідомлення зазвичай автоматично генеруються веб-службою при її викликі.

Щоразу, коли клієнтська програма викликає метод у веб-сервісі, веб-сервіс автоматично генерує SOAP-повідомлення, що містить необхідні деталі даних, які будуть надіслані з веб-сервісу до клієнтської програми.

Як уже говорилося у попередній темі цього підручника з SOAP, просте SOAP-повідомлення складається з наступних елементів:

- Елемент конверту
- елемент заголовка
- елемент тіла

Давайте розглянемо наведений нижче приклад простого SOAP-повідомлення і подивимося, що робить елемент.

Рис. 3.2. SOAP повідомлення

Як видно з наведеного вище повідомлення SOAP, першою частиною повідомлення SOAP є елемент envelope, який використовується для інкапсуляції повідомлення SOAP.

Наступним елементом є тіло SOAP, яке містить деталі фактичного повідомлення.

Наше повідомлення містить веб-сервіс під назвою "Guru99WebService".

Guru99Webservice приймає параметр типу 'int' та має ім'я TutorialID.

Тепер вищезазначене повідомлення SOAP буде передано між веб-сервісом і клієнтським додатком.

Можна побачити, наскільки корисною для клієнтської програми є наведена вище інформація. Повідомлення SOAP повідомляє клієнтську програму, як називається веб-служба, які параметри вона очікує, а також який тип кожного параметра, який приймає веб-служба.

3.2.3. Комунікаційна модель SOAP

Вся комунікація з SOAP здійснюється через протокол HTTP. До появи SOAP багато веб-сервісів використовували для зв'язку стандартний стиль RPC (Remote Procedure Call). Це був найпростіший тип комунікації, але мав багато обмежень.

Тепер розглянемо наведену нижче діаграму, щоб побачити, як працює ця комунікація. У цьому прикладі припустимо, що на сервері розміщено веб-сервіс, який надає два методи у вигляді

GetEmployee - отримання всіх даних про співробітника.

SetEmployee – встановлює значення таких деталей, як посада співробітника, зарплата тощо. відповідно.

При звичайній взаємодії в стилі RPC клієнт просто викликає методи у своєму запиті та відправляє необхідні параметри на сервер, а сервер відправляє бажану відповідь.

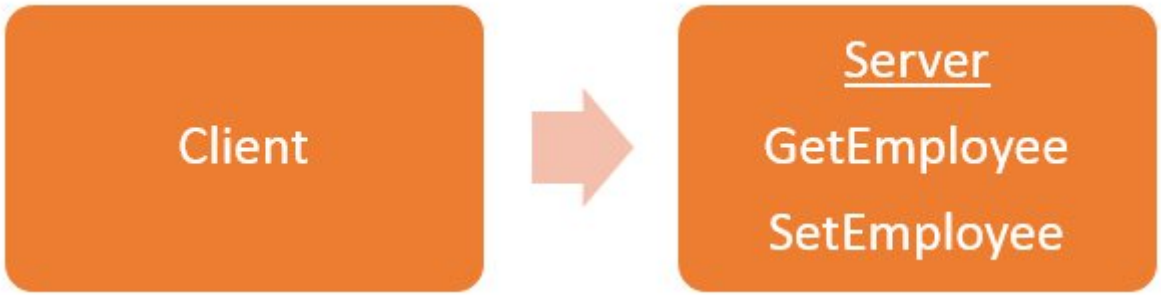


Рис. 3.3. Модель звичайної взаємодії

Наведена вище модель комунікації має такі серйозні обмеження

- Не залежить від мови - сервер, на якому розміщуються методи, працює певною мовою програмування, і зазвичай виклики на сервер здійснюються тільки цією мовою програмування.
- Не стандартний протокол – Коли здійснюється виклик віддаленої процедури, виклик здійснюється не за стандартним протоколом. Це було проблемою, оскільки переважно всі комунікації через Інтернет мали здійснюватися за протоколом HTTP.
- Брандмауери - Оскільки виклики RPC не відбуваються за звичайним протоколом, на сервері повинні бути відкриті окремі порти, щоб клієнт міг спілкуватися із сервером. Зазвичай всі брандмауери блокують такий трафік, і для того, щоб забезпечити роботу зв'язку між клієнтом і сервером, потрібно багато налаштувань.

Щоб подолати всі вищезгадані обмеження, SOAP використовує наступну модель комунікації:

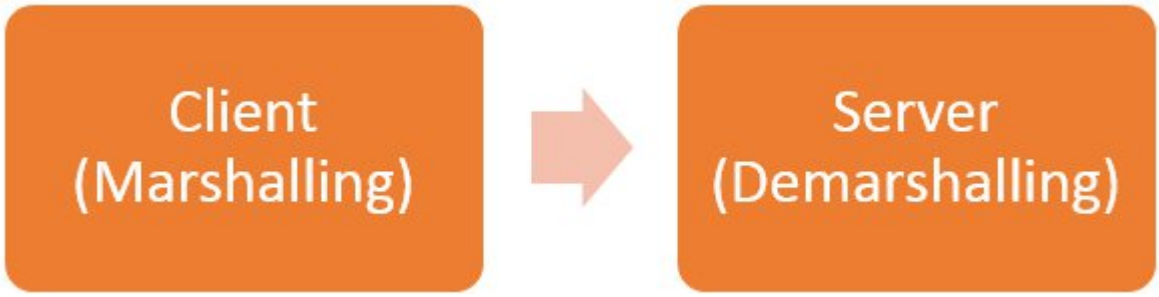


Рис. 3.4. Модаель SOAP взаємодії

Клієнт форматує інформацію про виклик процедури та будь-які аргументи в SOAP-повідомлення та надсилає його на сервер як частину HTTP-запиту. Цей процес інкапсуляції даних у SOAP-повідомленні відомий як Marshalling.

Потім сервер розгортає повідомлення, відправлене клієнтом, дивиться, що запитав клієнт, і відправляє відповідну відповідь клієнту у вигляді SOAP-повідомлення. Практика розгортання запиту, надісланого клієнтом, відома як Demarshalling.

3.3. REST мікросервіси

В архітектурі мікросервісів кожен додаток проектується як незалежний сервіс. REST є цінним архітектурним стилем для мікросервісів завдяки своїй простоті, гнучкості та масштабованості.

Однією з найсильніших переваг REST для мікросервісів є те, що послуги можуть взаємодіяти, не вимагаючи внутрішніх знань один про одного. Код кожного сервісу може розвиватися у своєму власному темпі, не торкаючись інших послуг. Тому REST особливо добре підходить для мікросервісів, які зазвичай спираються на невеликі автономні команди, що розробляють, розгортають та масштабують свої відповідні сервіси самостійно. Крім того, у мікросервісах RESTful API стандартизовані відповідно до специфікації OpenAPI, яка забезпечує документований контракт на те, як сервіси повинні взаємодіяти під час поточної розробки.

RESTful API використовують стандартні HTTP-дієслова для операцій створення, отримання, оновлення та видалення і повинні дотримуватися концепції ідемпотентності та безпеки. Вони повинні звертати особливу увагу на те, чи можна повторити операцію кілька разів без зміни результату та чи безпечно кешувати результати для подальшого використання. API REST реалізують дієслова HTTP відповідно до певних протоколів:

Операції POST використовуються для створення або оновлення ресурсів, але їх не можна викликати багаторазово. Якщо ресурс POST викликається кілька разів, то при кожному виклик створюється новий, унікальний ресурс.

Операції GET можуть викликатися багаторазово, не викликаючи побічних ефектів. Вони повинні використовуватися лише для отримання інформації.

Операції PUT оновлюють ресурси. Зазвичай вони включають повну копію ресурсу, щоб операцію можна було викликати кілька разів.

Операції DELETE можуть бути викликані кілька разів. Навіть якщо ресурс може бути видалений лише один раз, і при наступних викликах код відповіді буде різним, сервер залишається в тому самому стані при багаторазових викликах.

Добре спроектовані REST API дотримуються протоколу HTTP щодо ресурсів та URI, HTTP-дієслів та відсутності стану. Наприклад, REST API використовують HTTP-дієслова, щоб операції були ідемпотентними та безпечними, де це можливо.

Jakarta RESTful Web Services (JAX-RS) – це Jakarta EE API для розробки веб-сервісів, що відповідають принципам REST. Він визначає набір інтерфейсів та анотацій Java, які спрощують розробку REST-додатків. Деякі ключові особливості JAX-RS включають:

- Набір анотацій для оголошення класів ресурсів та типів даних, які вони підтримують.
- Набір інтерфейсів, що дозволяють розробникам програм отримати доступ до контексту часу виконання
- Розширювана структура для інтеграції користувальницьких обробників контенту.

Застосовуючи анотації до звичайних об'єктів Java, JAX-RS визначає структуру для прив'язки шаблонів URI та операцій HTTP до методів у класах програми. Ця основа забезпечує стандартизований спосіб взаємодії з ресурсами програм через Інтернет. Анотації підтримуються будь-яким сервером програм, сумісним з Jakarta EE.

Висновки до розділу 3

Було розглянуто основні інструменти роботи з великими даними, серед яких було розглянуто протоколи SOAP та REST.

SOAP має такі переваги в порівнянні з REST:

- Незалежність від мови, платформи та транспорту (REST вимагає використання HTTP).
- Добре працює в розподілених корпоративних середовищах (REST передбачає прямий зв'язок "крапка-крапка")
- Стандартизований
- Забезпечує значну розширюваність перед збиранням у вигляді стандартів WS*
- Вбудована обробка помилок
- Автоматизація під час використання з певними мовними продуктами

Переваги REST в порівнянні з SOAP

REST здебільшого простіше у використанні та є більш гнучким. Він має такі переваги перед SOAP:

- Для взаємодії з веб-службою не потрібні дорогі інструменти
- Менша крива навчання
- Ефективність (SOAP використовує XML для всіх повідомлень, REST може використовувати менші формати повідомлень)
- Швидкість (не потрібна тривала обробка)
- Ближче до інших веб-технологій з філософії проектування

Дуже небагато веб-служб, наприклад Amazon, підтримують обидва протоколи. При прийнятті рішення слід зосередитись на тому, який веб-сервіс найкраще відповідає потребам, а не на тому, який протокол використати.

РОЗДІЛ 4. Способи управління даними в СОА

4.1. Оркестрування сервісів

4.1.1. Визначення оркестрування сервісів

Оркестрування сервісів – це уніфіковане управління IT-сервісами та ресурсами з метою оптимізації наскрізних IT- та бізнес-процесів.

Платформа оркестрування сервісів розташовується на вершині IT-стеку, абстрагуючи складність від окремих систем та середовищ. Це дозволяє IT-відділам легко керувати процесами, контролювати ресурси і об'єднувати окремі завдання в наскрізні процеси з єдиної точки управління, не вдаючись до допомоги сценаріїв користувача.

4.1.2. Важливість оркестрування сервісів

Сучасні IT-процеси управляють даними на платформах, системах і середовищах, створених з використанням різних технологій від різних постачальників. Традиційно ці інструменти та системи існували як єдине ціле, кероване персоналом із вузькою, але дуже глибокою експертизою. Потім ці співробітники високого рівня повинні були досліджувати, розробляти і тестувати сценарії користувача для створення крос-платформеного процесу.

Цей метод забирає багато часу і може призвести до помилок, і навіть за наявності сценаріїв IT-відділ повинен покладатися на ряд ручних тригерів і передач. В епоху даних в реальному часі, підвищених споживчих очікувань і мінливих бізнес-цілей ручні методи і сценарії користувача не можуть забезпечити гнучкість і швидкість, які потрібні організаціям.

Обчислювальні середовища мали тенденцію підвищення рівня абстракції ще до появи FORTRAN. За останні 20 років IT-середовища ставали все більш різноманітними та складними, що робить необхідним новий рівень можливостей оркестрування для спрощення та оптимізації цих середовищ.

4.1.3. Переваги

Платформи оркестрування сервісів призначені для уніфікації управління розрізненими процесами та системами у локальних та хмарних середовищах. Це досягається за допомогою універсальних конекторів та API веб-сервісів, які спрощують керування даними та залежностями між цими розрізненими інструментами та середовищами без необхідності написання додаткових сценаріїв.

Надаючи єдиний інструмент для розробки наскрізних процесів, IT-команди можуть централізувати управління та моніторинг цих процесів та пов'язаних із ними ресурсів. Засоби моніторингу та автоматичного усунення несправностей у режимі реального часу дозволяють уникнути затримок та збоїв у наданні IT-сервісів, а поглиблені уявлення та звіти полегшують виявлення закономірностей використання та оптимізацію машинних ресурсів.

Існує довгий перелік переваг, які IT-відділ може отримати завдяки оркестровці сервісів. Ці ключові переваги включають:

Поліпшення SLA - При виникненні затримок, перевитрати, недовитрати або збоїв IT-відділ є єдиним місцем для виявлення першопричини, на відміну від перевірки декількох інструментів. Це може значно покращити час відгуку. Крім того, платформа оркестрування послуг може резервувати та надавати ресурси для забезпечення своєчасного завершення критично важливих для SLA робочих процесів.

Швидкий DevOps - Конструктори робочих процесів з перетягуванням дозволяють IT-фахівцям легко розробляти нові надійні сервіси та наскрізні процеси. Замість того, щоб створювати сценарії з'єднань між інструментами, API та існуючі сценарії можуть бути прийняті та перетворені на багаторазово використовувані компоненти, які, у свою чергу, можуть бути перетягнуті в наскрізні робочі процеси.

(В якості додаткової переваги шукайте рішення, що спрощують управління життєвим циклом сервісів за допомогою автоматизованої документації, історії змін та управління змінами).

Операційний спокій – інструменти оркестрування сервісів розроблені для спрощення IT-середовища. Це включає скорочення часу, яке IT-фахівцям доводиться витрачати на боротьбу з пожежами, завдяки можливостям автоматичного усунення наслідків, моніторингу та своєчасного оповіщення. Якщо завдання або робоче навантаження наражається на ризик збою, платформи оркестрованих сервісів можуть автоматично виконати робочий процес з усунення наслідків або надіслати попередження відповідним IT-командам та системам.

Підвищення ефективності та зниження витрат - Завдяки універсальним конекторам та доступності API, IT-спеціалісти можуть автоматизувати більше, а вручну керувати менше. Автоматизація сервісів означає, що IT-спеціалісти можуть більше зосередитися на важливіших проектах, таких як новітні бізнес-послуги та ініціативи з цифрової трансформації.

4.2. Хмарне оркестрування

Постачальники публічних хмар (включаючи Amazon AWS, Microsoft Azure, Google Cloud, VMware і т.д.) часто не пропонують інтеграції з хмарними платформами/хмарними сервісами, що конкурують. Така прив'язка до постачальників ускладнює управління даними в гібридних хмарних та мультихмарних середовищах.

Завдяки радикальній розширюваності, пропонованій платформами оркестрування сервісів, ці інструменти стали ідеальним рішенням для IT-команд, які прагнуть спростити управління хмарою серед різних постачальників.

Крім інтеграції хмарних процесів, інструменти оркестрування сервісів використовуються для управління IT-інфраструктурою, автоматизуючи надання та депровізування віртуальних машин. Якщо процес або робочий процес вимагає додаткових ресурсів, інструмент оркестрування сервісів може надати необхідну хмарну інфраструктуру, а потім депровізувати ці сервери після завершення робочого навантаження. Це заощаджує час IT-фахівців, запобігаючи необхідності ручного управління ресурсами, та знижує операційні витрати за рахунок скорочення незадіяних ресурсів машин.

Крім того, деякі інструменти оркестрування сервісів застосовують алгоритми машинного навчання (ML) до історичних даних про час виконання. Ці алгоритми виявляють тенденції та закономірності у використанні ресурсів, оптимізуючи розподіл ресурсів для майбутніх запусків.

4.3. Загальні функції та можливості провайдерів хмарного оркестрування

Можливість інтеграції розрізнених додатків та систем є ключовим фактором для організації IT та бізнес-процесів. SOAP (Service Orchestration and Automation Platforms) також підтримують ряд додаткових можливостей та функцій, які допомагають IT-фахівцям керувати та оптимізувати ці процеси з єдиного центру. SOAP із найвищим рейтингом від Gartner Peer Insights мають кілька спільних можливостей:

- Автоматизація на основі подій

SOAP надають тригери на основі подій, які можуть запускати автоматизовані робочі процеси, як тільки відбувається певна подія, покращуючи методи планування дати/часу для надання даних у режимі реального часу, підвищення гнучкості бізнесу та подальшого зниження потреб у ручному управлінні.

- Конструктор робочих процесів

SOAP включають графічні конструктори робочих процесів, які можна використовувати для збирання крос-платформних процесів. Інструменти робочих процесів дозволяють IT-фахівцям візуалізувати залежності та оркеструвати розрізнені завдання.

- Моніторинг та оповіщення

Можливості моніторингу та оповіщення допомагають скоротити середній час усунення неполадок, покращити SLA та запропонувати дієві ідеї для оптимізації процесів. Користувачі можуть відслідковувати робочі навантаження в режимі реального часу, отримувати оповіщення у разі виникнення проблем або використовувати ML/AI для аналізу історичних даних системи.

- Надання ресурсів

SOAP можуть використовуватися для оркестрування ресурсів у локальних та хмарних середовищах, забезпечуючи надання та депровізування кращих ресурсів у потрібний час. Це може бути зроблено динамічно з урахуванням попиту робоче навантаження, не вимагаючи ручного втручання.

- Автоматизація самообслуговування

Автоматизація самообслуговування дозволяє бізнес-користувачам та співробітникам служби підтримки керувати своїми власними процесами. За наявності контролю доступу на основі ролей користувачі, які не є IT-фахівцями, можуть виконувати та контролювати складні процеси, покращуючи IT-послуги та знижуючи навантаження на IT. Це демократизує автоматизацію процесів, зберігаючи контроль IT-відділу над безпекою, середовищами та системами.

- Мобільні додатки

Мобільні та веб-програми можуть використовуватися для управління та моніторингу складних процесів з будь-якого місця. Користувачі можуть

планувати та виконувати робочі процеси, отримувати push-повідомлення та швидко реагувати на проблеми або бізнес-запити.

4.4. Концепція управління потоками даних

Концепція надає одержувачу механізм для управління швидкістю передачі даних, щоб приймаючий вузол не був перевантажений даними від вузла, що передає. Управління потоком слід відрізняти від управління навантаженням, яке використовується для управління потоком даних, коли навантаження вже відбулося. Механізми управління потоком можна класифікувати за тим, чи посилає приймаючий вузол зворотний зв'язок передавального вузла чи ні.

Управління потоком важливо, тому що комп'ютер-відправник може передавати інформацію швидше, ніж комп'ютер-одержувач може її прийняти та обробити. Це може статися, якщо приймаючі комп'ютери мають велике навантаження в порівнянні з комп'ютером, що відправляє, або якщо приймаючий комп'ютер має меншу обчислювальну потужність, ніж комп'ютер, що відправляє.

4.4.1. Зупинка та очікування автоматичного повторного запиту

Управління потоком із зупинкою та очікуванням - це найпростіша форма управління потоком. У цьому методі повідомлення розбивається на кілька кадрів, і одержувач повідомляє про готовність прийняти кадр даних. Відправник очікує підтвердження отримання (ACK) після кожного кадру протягом певного часу (званого тайм-аутом). Приймач посилає ACK, щоб повідомити відправнику, що кадр даних було прийнято правильно. Відправник надсилає наступний кадр лише після отримання ACK.

Операції

1. Відправник: Передає один кадр за один раз.

2. Відправник очікує на отримання ACK протягом тайм-ауту.

3. Отримувач: Передає підтвердження (ACK) у міру отримання кадру.

4. Перехід до кроку 1 відбувається після отримання ACK або після закінчення тайм-ауту.

Якщо кадр або ACK втрачені під час передачі, кадр передається повторно. Цей процес повторної передачі відомий як автоматичний повторний запит (ARQ).

Проблема Stop-and-wait полягає в тому, що за один раз може бути передано лише один кадр, і це часто призводить до неефективної передачі, оскільки поки що відправник не отримає ACK, він не може передати жодного нового пакета. У цей час і відправник і канал не використовуються.

4.4.2. Протокол розсувного вікна

Метод управління потоком, при якому приймач дає передачу дозвіл на передачу даних до тих пір, поки не заповниться вікно. Коли вікно заповнено, передавач повинен припинити передачу, доки приймач не оголосить про вікно.

Керування потоком зі ковзним вікном краще використовувати, коли розмір буфера обмежений і заздалегідь встановлений. Під час типового обміну даними між відправником та одержувачем одержувач виділяє буферний простір для n кадрів (n – розмір буфера у кадрах). Відправник може надіслати, а одержувач може прийняти n кадрів, не чекаючи на підтвердження. Для відстеження кадрів, які отримали підтвердження, кадрам надається порядковий номер. Приймач підтверджує кадр, надсилаючи підтвердження, що містить порядковий номер наступного очікуваного кадру. Це підтвердження повідомляє, що приймач готовий прийняти n кадрів, починаючи із зазначеного номера. І відправник і одержувач підтримують так зване вікно. Розмір вікна менше або дорівнює розміру буфера.

Управління потоком за допомогою ковзного вікна має набагато кращі характеристики, ніж управління потоком за принципом "зупинися і чекай". Наприклад, у бездротовому середовищі, де швидкість передачі даних низька, а рівень шуму дуже високий, очікування підтвердження для кожного пакета, що передається, недоцільно. Тому передача даних масово забезпечить кращу продуктивність з погляду вищої пропускної спроможності.

Управління потоком зі ковзним вікном - це протокол "від точки до точки", що передбачає, що жодна інша організація не намагається встановити зв'язок, доки не завершиться поточна передача даних. Вікно, яке підтримує відправник, вказує, які кадри він може надіслати. Відправник посилає всі кадри у вікні і чекає на підтвердження (на відміну від підтвердження після кожного кадру). Потім відправник зсуває вікно на відповідний порядковий номер, вказуючи тим самим, що кадри у вікні, починаючи з поточного номера, можуть бути відправлені.

4.4.3. Програмне керування потоком

Програмне керування — метод керування потоком, що використовується в комп'ютерних лініях передачі даних. Він використовує спеціальні коди, що передаються всередині діапазону основного каналу зв'язку. Ці коди зазвичай називаються XOFF і XON (від "transmit off" та "transmit on", відповідно). Таким чином, програмне управління потоком іноді називають управління потоком XON/XOFF. Це відрізняється від керування потоком за допомогою спеціальних позасмугових сигналів - "апаратного керування потоком".

Коли один кінець передачі даних не може більше приймати дані (або наближається до цього моменту), він посилає XOFF на інший кінець. Інший кінець отримує код XOFF і зупиняє передачу. Як тільки перший кінець знову готовий до прийому даних, він посилає XON і інший кінець відновлює передачу.

Наприклад, комп'ютер, який надсилає дані на повільний принтер. Оскільки комп'ютер надсилає дані швидше, ніж принтер встигає їх роздрукувати, принтер відстає і наближається до ситуації, коли він перевантажиться даними. Принтер реагує на цю ситуацію, надсилаючи XOFF комп'ютеру, який тимчасово припиняє надсилання даних. Коли принтер знову готовий до прийому даних, він посилає XON на комп'ютер, який знову починає надсилати дані.

XOFF/XON можуть використовуватися в обох напрямках, наприклад, два підключені один до одного телепринтери.

Висновки до розділу 4

У даному розділі розглянуто поняття оркестрування сервісів, провайдери хмарного оркестрування, їх переваги та функціонал, а також концепцію управління потоками даних. Виходячи з даних цього розділу можна виділити три основні провайдери хмарного оркестрування сервісів:

- Azure DevOps
- Amazon Web Services
- Google Cloud

У наступному розділі розглянемо практичну задачу роботи з великими даними, для кожного з провайдерів розглянемо рішення, що надаються для роботи з big data та програмно реалізуємо реалізацію задачі для одного з сервісів.

РОЗДІЛ 5. Розробка додатку з відкритим датасетом великих даних та сервіс-орієнтованою архітектурою

5.1. Огляд хмарних рішень

5.1.1. Google Cloud Platform

GCP пропонує широкий спектр послуг в області Великих Даних, які можуть бути використані для управління та аналізу даних, у тому числі:

Google Cloud BigQuery

BigQuery дозволяє зберігати та виймати набори даних з величезними обсягами даних. Служба використовує табличну структуру, підтримує SQL та легко інтегрується з усіма службами GCP. Ви можете використовувати BigQuery як для пакетної, так і потокової обробки даних. Сервіс ідеально підходить для автономного аналізу та інтерактивних запитів.

Google Cloud Dataflow

Dataflow пропонує безсерверну пакетну та потокову обробку. Ви можете створювати власні конвеєри керування та аналізу, а Dataflow автоматично керує вашими ресурсами. Сервіс інтегрується із сервісами GCP, такими як BigQuery, та зі сторонніми рішеннями, такими як Apache Spark.

Google Cloud Dataproc

Dataproc дозволяє інтегрувати стек відкритих вихідних кодів та оптимізувати процес за рахунок автоматизації. Це повністю керована послуга, яка допоможе вам виконувати запити та потокову передачу даних за допомогою таких ресурсів, як Apache Hadoop у хмарі GCP Cloud. Dataproc може бути інтегрований з іншими GCP-сервісами, такими як Bigtable.

Google Cloud Pub/Sub

Pub/Sub – це асинхронна служба обміну повідомленнями, яка забезпечує зв'язок між різними програмами. Pub/Sub часто використовується для аналітичних потокових конвеєрів. Pub/Sub можна інтегрувати із системами на GCP або за його межами та виконувати спільне введення даних про події та дії за шаблонами розповсюдження. Це є реалізація enterprise service bus.

Google Cloud Composer

Composer - це повністю керована хмарна служба оркестрування робочих процесів, побудована з урахуванням Apache Airflow. Ви можете використовувати Composer для керування обробкою на кількох платформах та створення власного гібридного середовища. Composer дозволяє визначити процес мовою Python. Потім сервіс автоматизує завдання обробки даних, такі як ETL.

Google Cloud Data Fusion

Data Fusion – це повністю керована послуга інтеграції даних, яка дозволяє зацікавленим особам із різними навичками готувати, передавати та перетворювати дані. Data Fusion дозволяє створювати конвеєри даних ETL/ELT без коду за допомогою візуального інтерфейсу "вкажи та клацніть". Data Fusion - це проект з відкритим вихідним кодом, який забезпечує переносимість, необхідну для гібридних та мультихмарних інтеграцій.

Google Cloud Bigtable

Bigtable – це повністю керована служба баз даних NoSQL, розроблена для забезпечення високої продуктивності під час роботи з великими даними. Bigtable працює на стеку зберігання даних із низькою затримкою, підтримує API HBase з відкритим вихідним кодом та доступний у всьому світі. Послуга ідеально підходить для тимчасових рядів, фінансових даних, маркетингу,

графічних даних та IoT. На ньому працюють основні служби Google, такі як Analytics, Search, Gmail та Maps.

Каталог хмарних даних Google

Каталог даних дозволяє знайти дані, які можна використовувати для запису ділових та технічних метаданих. Щоб легко знаходити набори даних, можна використовувати теги схеми і створити каталог, що настроюється. Для захисту даних служба використовує контроль доступу. Служба інтегрується з Google Cloud Data Loss Prevention для класифікації конфіденційної інформації.

Google пропонує еталонну архітектуру для великомасштабної аналітики на Google Cloud, з більш ніж 100 000 подій за секунду або більше 100 МБ потокової передачі за секунду. Архітектура базується на Google BigQuery.

Архітектура роботи з великими даними

Google рекомендує будувати архітектуру великих даних із гарячими та холодними шляхами. Гарячий шлях – це потік даних, що вимагає обробки практично в режимі реального часу, а холодний шлях – це потік даних, який може бути оброблений після невеликої затримки.

Це має низку переваг, у тому числі:

- Можливість зберігати журнали для всіх подій без перевищення квот
- Зниження витрат, оскільки лише деякі події мають оброблятися як потокові вставки (які є дорожчими).

Наступна діаграма ілюструє архітектуру:

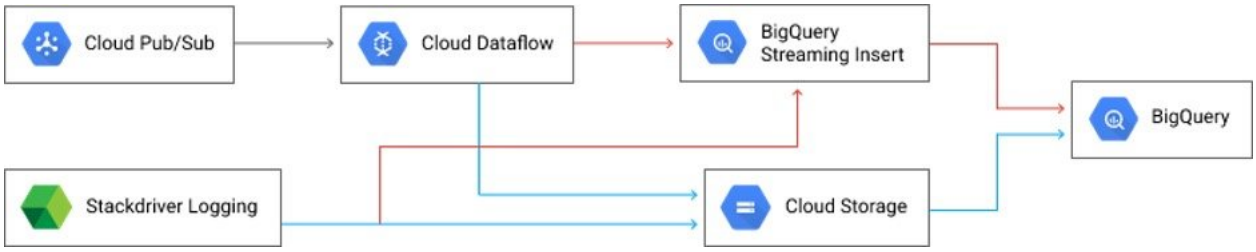


Рисунок 5.1. Архітектура роботи з великими даними в GCP

Дані надходять із двох можливих джерел - подій аналітики, опублікованих у Cloud Pub/Sub, та журналів Google Stackdriver Logging. Дані поділяються на два шляхи:

- Гарячий шлях (червоні стрілки) надходить у BigQuery за допомогою потокової вставки, щоб забезпечити безперервний потік даних.
- Холодний шлях (сині стрілки) надходить у Google Cloud Storage і звідти пакетно завантажується до BigQuery.

5.1.2. Microsoft Azure

Azure надає декілька платформ-сервісів (PaaS) для роботи з великими даними:

- Azure synapse analytics
- Azure HDInsight
- Azure Databricks

Azure synapse включає у себе наступні сервіси:

- **Synapse Pipelines —** допомагає отримувати та передавати дані з використанням візуальних воркфлоу.
- **Apache Spark —** інструмент для аналізу, обробки та перетворення великих даних
- **Synapse SQL —** сукупність кластерів обробки даних за допомогою SQL запитів
- **Synapse studio —** інструмент управління synapse сервісами
- **Azure Data Lake**

Архітектура Azure synapse analytics:

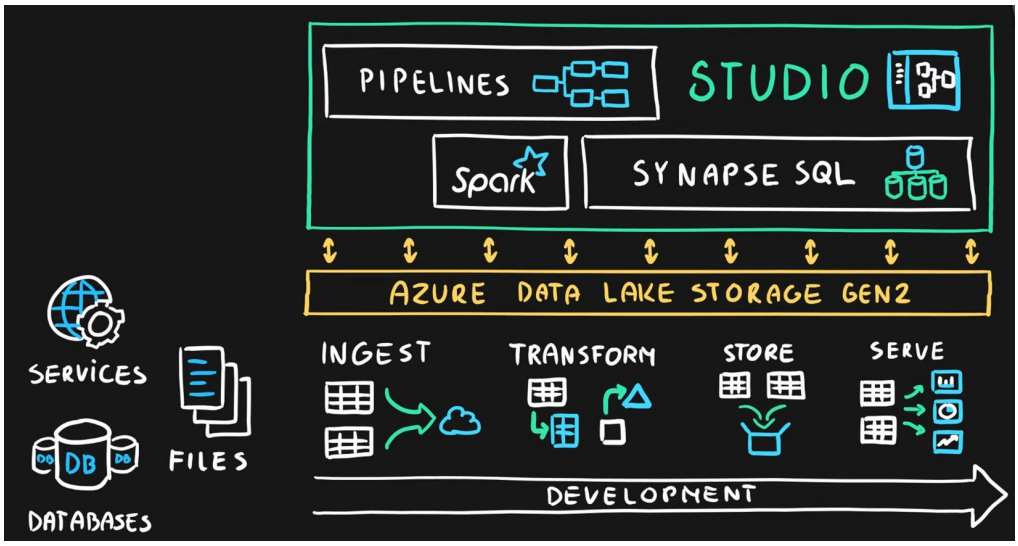


Рисунок 5.2. Архітектура роботи з великими даними в Azure synapse

Azure HDInsight надає big data кластери:

- Spark
- Kafka
- Hbase
- Hive
- Hadoop
- Apache Storm

Архітектура Azure HDInsight

Рисунок 5.3. Архітектура роботи з великими даними в HDInsight

5.1.3. Amazon web services

AWS пропонує колекцію ресурсів та рішень, розроблених для вирішення завдань, що виникають у кожній області великих даних:

Firehose

Amazon Kinesis Firehose виконує стиснення даних, пакетну обробку, шифрування та функції Lambda. Firehose передає потокові дані в реальному часі на S3 Amazon, надійно завантажуючи їх у озера даних, сховища даних або аналітичні інструменти. Firehose автоматично масштабується відповідно до продуктивної потужності даних будь-якої організації та не вимагає постійного адміністрування.

AWS Snowball

AWS Snowball - це ресурс для транспортування даних, який ефективно та безпечно переносить великі обсяги даних з ваших внутрішніх, локальних платформ зберігання та кластерів Hadoop у відра S3. Після створення завдання за допомогою консолі керування AWS буде автоматично доставлено пристрій Snowball.

Amazon S3

Високомасштабований, безпечний, довговічний ресурс об'єктного зберігання, який може зберігати будь-які типи даних, взяті з будь-якого місця. S3 зберігає дані, зібрані з корпоративних програм, веб-сайтів, мобільних пристроїв, а також IoT-пристроїв та датчиків. Він також може зберігати будь-які обсяги даних із неперевершеним ступенем доступності.

AWS Glue

Служба даних, яка зберігає метадані в центральному сховищі та спрощує процес ETL (вилучення, перетворення та завантаження). Аналітик даних може створити та запустити завдання ETL лише кількома клацаннями миші з

консолі керування AWS Management Console. AWS Glue постачається з вбудованим каталогом даних, який функціонує як постійне сховище метаданих для кожного активу даних, дозволяючи аналітикам шукати та вимагати всі дані в єдиному поданні.

Amazon EMR

Apache Spark та Hadoop – популярні платформи для обробки даних, тому розумно мати інструмент AWS, який добре з ними працює. EMR від Amazon відмінно підходить для цього, оскільки він надає керовану послугу, яка швидко та без зусиль обробляє величезні обсяги даних. EMR підтримує 19 різних проектів з відкритим вихідним кодом, включаючи згадані раніше Spark та Hadoop. Крім того, в нього входять керовані блокноти EMR Notebooks, які підходять для спільної роботи, проектування даних та розвитку науки про дані.

Redshift

Amazon Redshift дозволяє аналітикам виконувати складні аналітичні запити до величезних обсягів структурованих даних без значних фінансових витрат – майже на 90% менше, ніж традиційні рішення для обробки даних. Redshift поставляється з Redshift Spectrum, який дозволяє аналітикам даних виконувати SQL-запити безпосередньо до екзабайт структурованих або неструктурованих даних, що зберігаються в S3, не вимагаючи непотрібного переміщення даних.

Amazon Quicksight

Amazon Quicksight - це утиліта Amazon Web Services, яка створює візуалізації та інтерактивні панелі приладів, доступні з будь-якого мобільного пристрою або веб-браузера. Цей ресурс бізнес-аналітики використовує надшвидкий, паралельний, вбудований на згадку механізм обчислень AWS (SPICE) для швидкого виконання розрахунків даних та створення графіків.

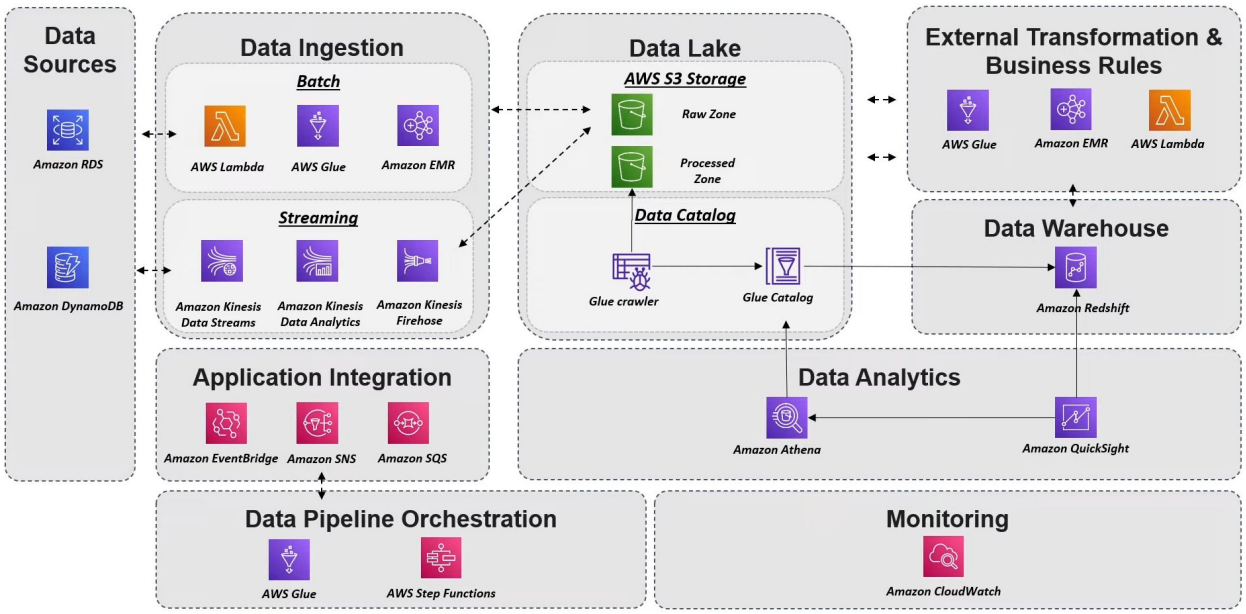


Рис 5.4. Архітектура роботи з великими даними в AWS

Проаналізувавши усі хмарні рішення, можна зробити висновок, що кожне з них надає схожі за функціоналом, серед відмінностей можна зазначити те що Azure сервіси не мають можливості роботи без серверу та автоматичного масштабування. Зараз в трендах великих даних набирає популярність та попит BigQuery від GCP, тому для реалізації програмного рішення було обрано саме цей хмарний провайдер.

5.2. Пошук та вибір відкритого датасету

В інтернеті у відкритому доступі знаходиться дуже багато датасетів, досить обширний список опубліковано в репозиторії GitHub:

https://github.com/awesomedata/awesome-public-datasets

Google Cloud Platform, обраний для реалізації, має свій набір нативних відкритих датасетів, у якому їх 236 одиниць. Ці датасети можна переносити одразу до BigQuery сервісу обраного проекту, що дозволяє їх досить просто інтегрувати. Є 48 датасетів під категорією Big Data:

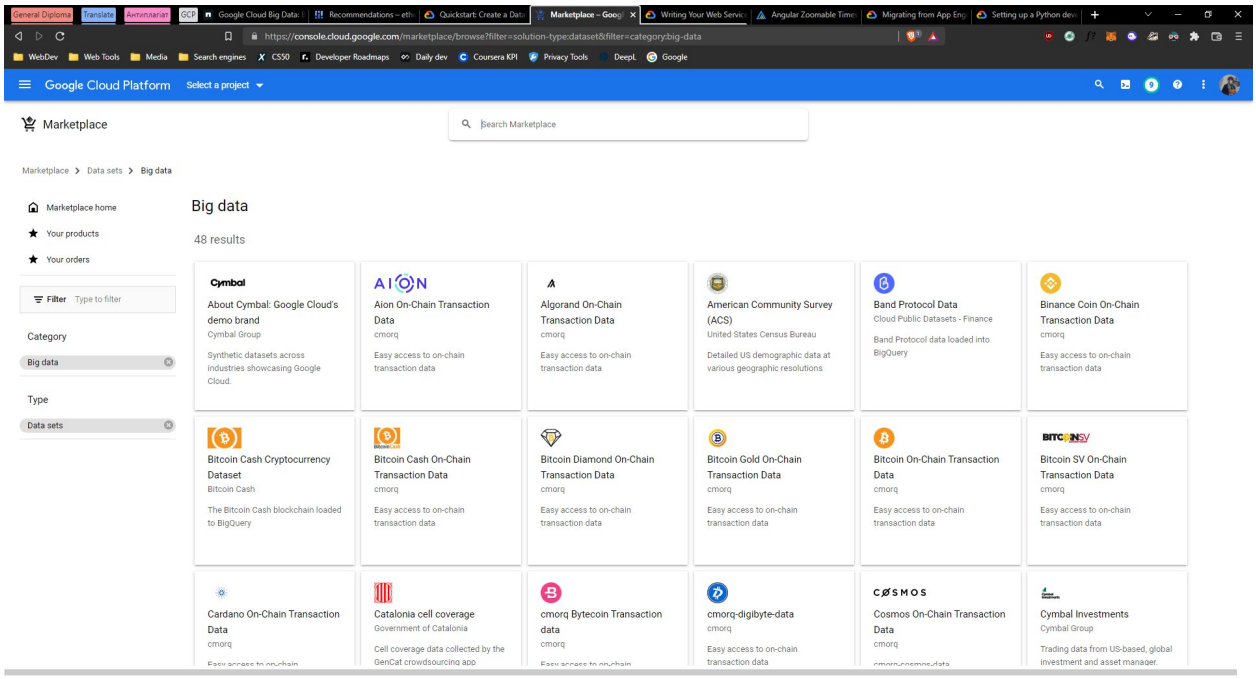


Рисунок 5.5. Відкриті датасети з категорією Big Data

Для обробки було обрано датасет Ethereum CryptoCurrency [19]. Датасет містить у собі 9 таблиць:

- balances - містить баланси Ether для всіх адрес, що оновлюються щодня.
- blocks - містить набір всіх блоків у блокчейні та їх атрибути.

  contracts: Деякі транзакції створюють смарт-контракти зі своїх вхідних байтів, і цей смарт-контракт зберігається за певною 20-байтовою адресою. Ця таблиця містить підмножину адрес Ethereum, які містять байт- код контракту, а також деякий базовий аналіз цього байт-коду.
- logs: для складання звітів про будь-який тип зареєстрованих подій на блокчейні Ethereum.
- sessions
- token_transfers:

  Найпопулярніший тип транзакцій на блокчейні Ethereum викликає контракт типу ERC20 для виконання операції передачі, переміщуючи кілька токенів з одного 20-байтового адреси на

інший 20-байтовий адресу. Дана таблиця містить підмножину цих транзакцій, дані були додатково оброблені та денормалізовані, щоб їх було легко використовувати для аналізу подій передачі токенів.

- tokens
- traces
- transaction - містить набір всіх транзакцій з усіх блоків та містить ідентифікатор блоку для отримання специфічної для блоку інформації, пов'язаної з кожною транзакцією.

Об’єм датасету на момент перенесення до Google Cloud BigQuery складав приблизно 5.8 терабайт. Датасет оновлюється кожен день. Виходячи зі змісту датасету та роботи з ним у додатку було обрано реалізацію blockchain explorer. Blockchain explorer — платформа на якій можна знайти транзакцію за хешем, токен у блокчейні, блоки та аккаунти.

5.3. Реалізація додатку

5.3.1. Налаштування середовища розробки

Додаток буде реалізовано у виді Single Page Application. Стек технологій для додатку:

- Google Cloud Platform (BigQuery, Pub/Sub, Dataproc)
- Angular — фронтенд фреймворк мови програмування JavaScript, розроблений Google.
- Flask — мікро веб фреймворк написаний на Python для сервер-сайд операцій.

Середовища розробки:

- Microsoft Visual Studio Code
- Google Cloud Platform Console

Створення проекту у GCP

Зайшовши у консоль GCP можна побачити перемикач проектів, нажавши на який з'явиться модальне вікно з вибором проекту або створенням нового. Нажавши на «New Project» з'явиться вікно з первинними налаштуваннями проекту:

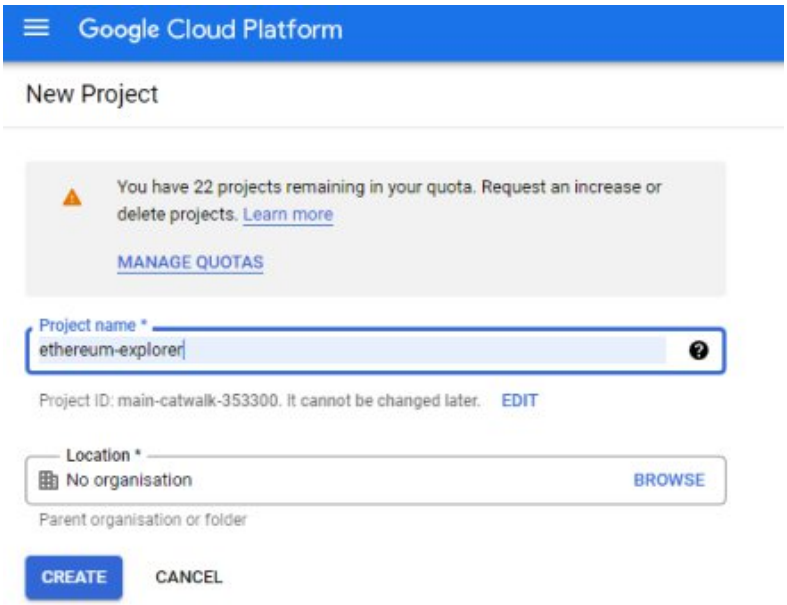


Рисунок 5.6. Створення проекту в GCP

Після створення проект відображається у вікні проектів:

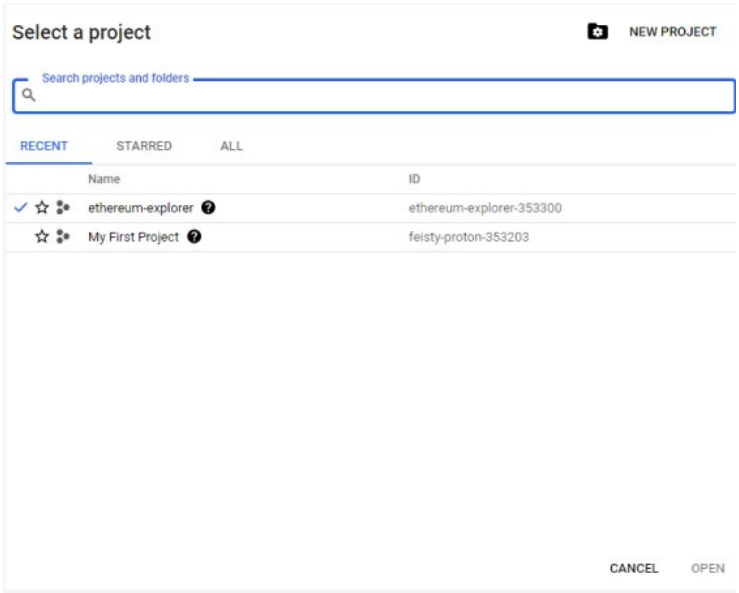


Рисунок 5.7. Створений проект

Для проекту автоматично створюється панель управління:

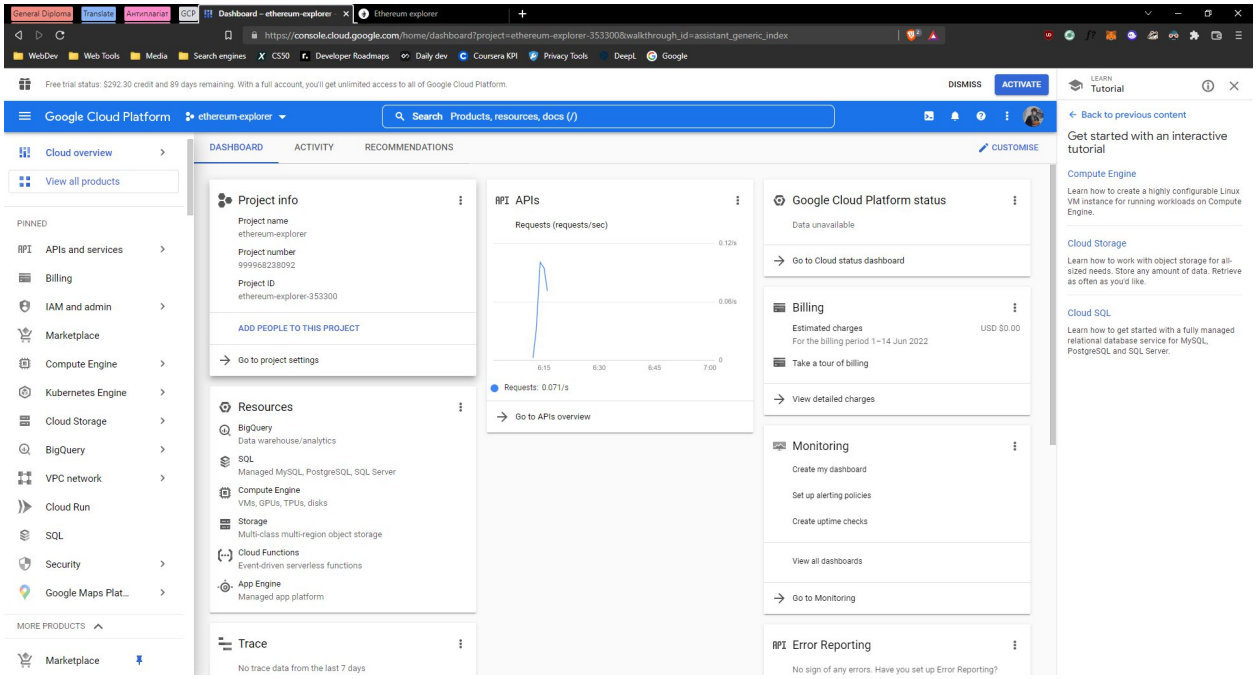


Рисунок 5.8. Панель управління проектом

З неї маємо доступ до всіх необхідних інструментів. Наступним кроком необхідно створити пустий датасет в BigQuery та перенести туди публічний датасет попередньо обраний для опрацювання.

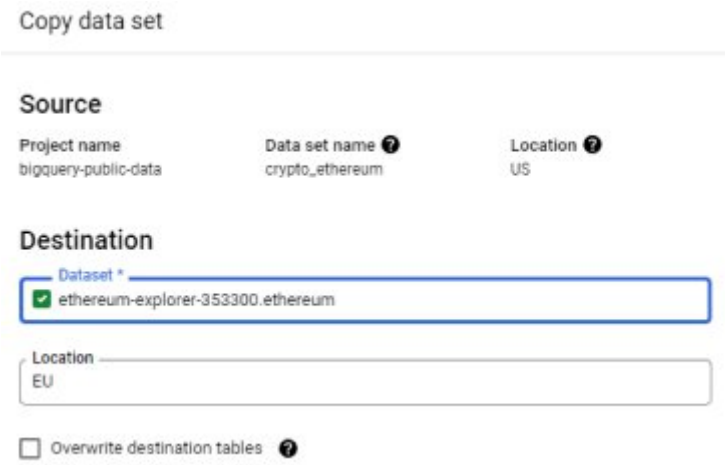


Рисунок 5.9. Перенос публічного датасету до проекту

Далі необхідно включити в налаштуваннях API наступні сервіси — Cloud Logging API, BigQuery API, Cloud Monitoring API, Dataflow API, BigQuery Storage API, Compute Engine API, Data pipelines API, Cloud Dataproc

API, BigQuery Reservation API, Analytics Hub API, Notebooks API, BigQuery Data Transfer API, BigQuery Migration API, Cloud Dataproc Control API, Cloud Debugger API, Cloud Deployment Manager V2 API, Cloud OS Login API, Cloud Pub/Sub API, Cloud Resource Manager API, Cloud Scheduler API, Cloud SQL, Cloud Storage, Cloud Storage API, Cloud Trace API, Google Cloud APIs, Google Cloud Storage JSON API, Service Management API, Service Usage API. На цьому базове налаштування проекту в google cloud завершено.

5.3.2. Реалізація продукту

Наступним кроком створимо базу веб додатку:

ng new ethereum-explorer

Команда вище генерує базовий додаток Angular. У згенерованому проекті створимо директорію backend, перейдемо в неї, створимо віртуальне середовище python та ініціалізуємо необхідні бібліотеки. Лістинг команд наведено нижче:

1. python -m venv env
2. .\venv\Scripts\activate
3. pip install flask flask_cors pandas google.cloud.bigquery db-dtypes

Кінцева структура проекту має наступний вигляд:

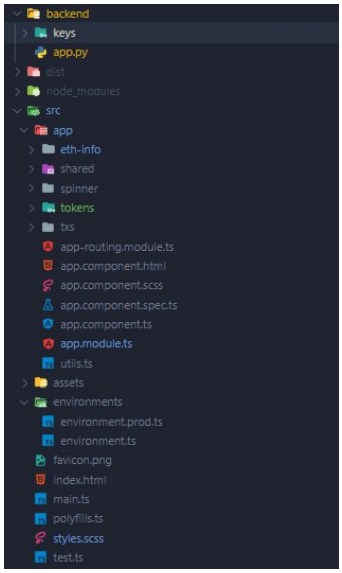
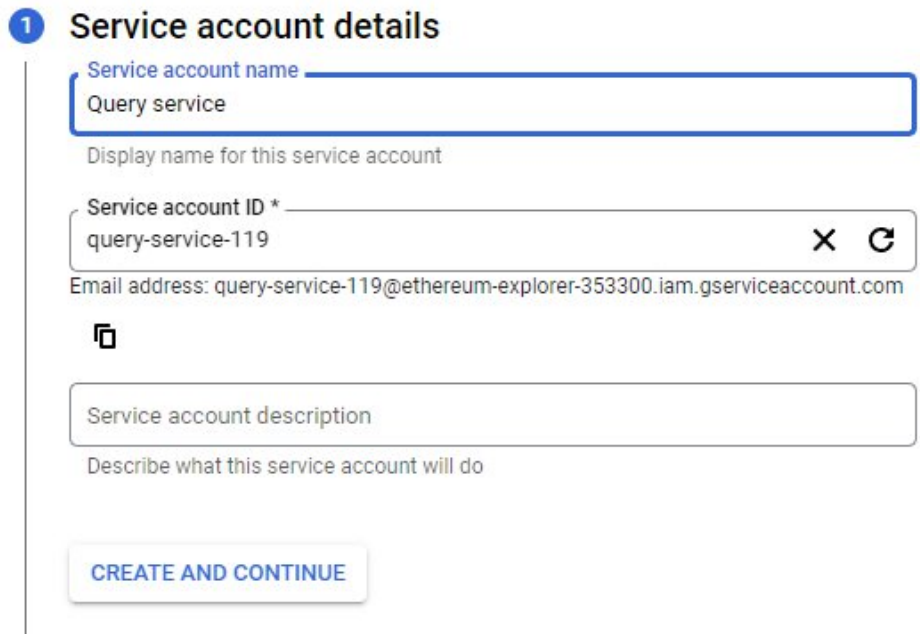


Рисунок 5.10. Структура проекту

Backend директорія — сервер, через який буде відбуватись взаємодія з GCP, директорія keys у ньому — ключі сервісів GCP, що генеруються автоматично при їх створені.

Src директорія — корінь клієнтської сторони додатку, яка робить запити на сервер та відображає інтерфейс користувача.

На головній сторінці відображаються останні транзакції та блоки на блокчейні. Датасет по цих таблицях не впорядкований, тобто треба додатково відсортувати значення по unix часових мітках. Цю задачу можна реалізувати через BigQuery API сервіс. Для цього в GCP необхідно створити сервіс акаунт:



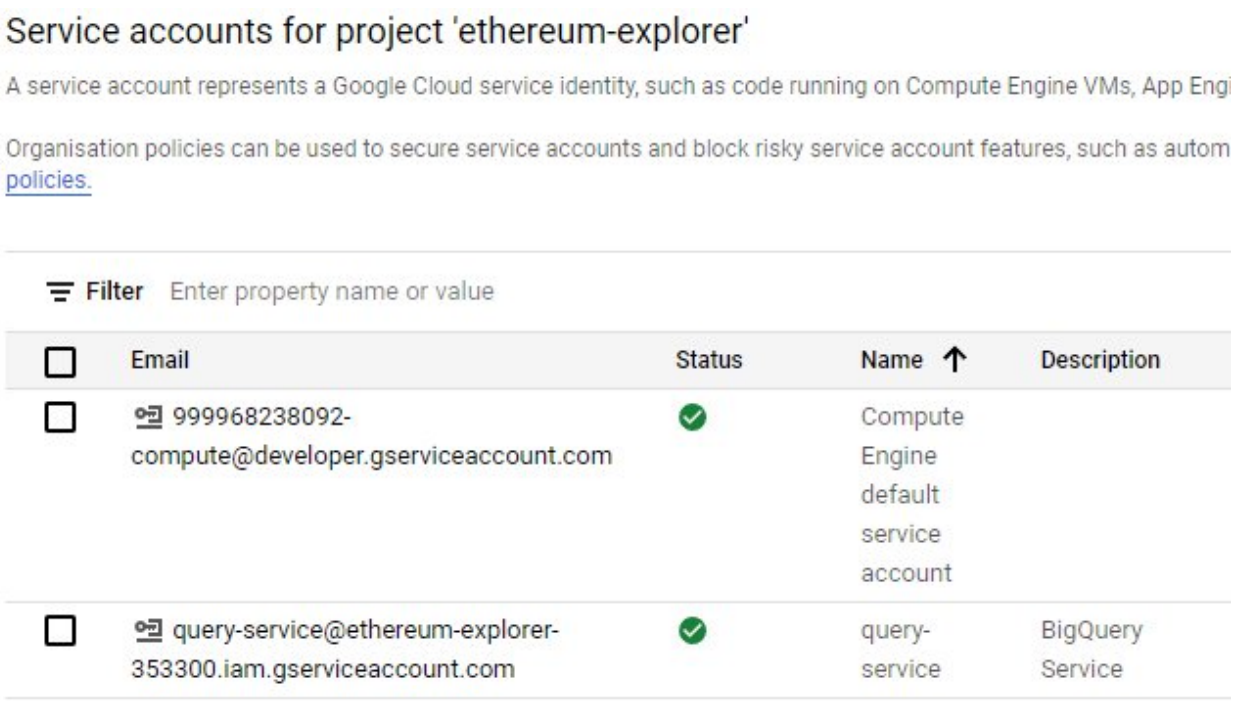


Рисунок 5.11., 5.12. Створення сервіс акаунту

Далі необхідно згенерувати приватний ключ сервісу та додати його в сформовану директорію ключів.

Створимо API контролер на сервері що виконує запит напряму до датасету у BigQuery:

```
You, 1 hour ago | 1 author (You)
import json
import os

from flask import Flask, jsonify
from flask_cors import CORS

import pandas as pd

from google.cloud import bigquery

app = Flask(__name__)
CORS(app)

@app.route('/get-latest-blocks')
def get_latest_blocks():
    bq_path = os.getcwd() + '\keys\keyBQ.json'
    os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = bq_path
    query = "SELECT * FROM `ethereum-explorer-353300.ethereum.blocks` ORDER BY number DESC LIMIT 50"
    bq_client = bigquery.Client()
    query_job = bq_client.query(query)
    df = query_job.to_dataframe()

    result = df.to_json()

    return json.dumps({'result': 200, 'data': result})      You, 1 hour ago • + latest blocks and tr
```

Рисунок 5.13. Контролер, що виконує REST запит

Контролер отримує з директорії, згенерований сервіс акаунтом BigQuery, ключ та додає його в системну змінну. Формується SQL запит га отримання блоків за найбільшим номером з лімітом в 50 записів. Ініціалізується BigQuery клієнт який робить запит до сервіс акаунту в GCP на отримання даних, дані опрацьовуються та перетворюються в JSON формат та віддаються на фронтенд. Перевага такого підходу над запитом до локальної бази даних або бази даних на сервері у тому, що не потрібно масштабувати апаратне забезпечення зі стрімким ростом даних, а також у тому, що BigQuery працює на роцподілених кластерах та запускає SQL query у паралельному режимі, що значно пришвидшує знаходження необхідних даних на великих датасетах.

Реалізація останніх транзакцій оформлена аналогічним чином, за виключенням того, що SQL запит формується за найбільшим показником часової мітки транзакції.

З клієнтської сторони запити надсилаються одночасно та виконуються через http запит до API контролера:

```
async ngOnInit() {
  this.dateNow = Date.now();

  await Promise.all([
    await this.getLatestBlocks(),
    await this.getLatestTxs(),
  ]);

  this.dateUpdateInterval = setInterval(() => {
    this.dateNow = Date.now();
  }, 1000);

  this.loading = false;
}
```

```
private async getLatestBlocks() {
  const blocks = JSON.parse(
    (
      await this.http
        .get(`${environment.apiEndpoint}/get-latest-blocks`)
        .toPromise()
    )['data']
  );
```

Рисунок 5.14. Запит до контролера

Результат роботи API, що використовує сервіс Big Query:

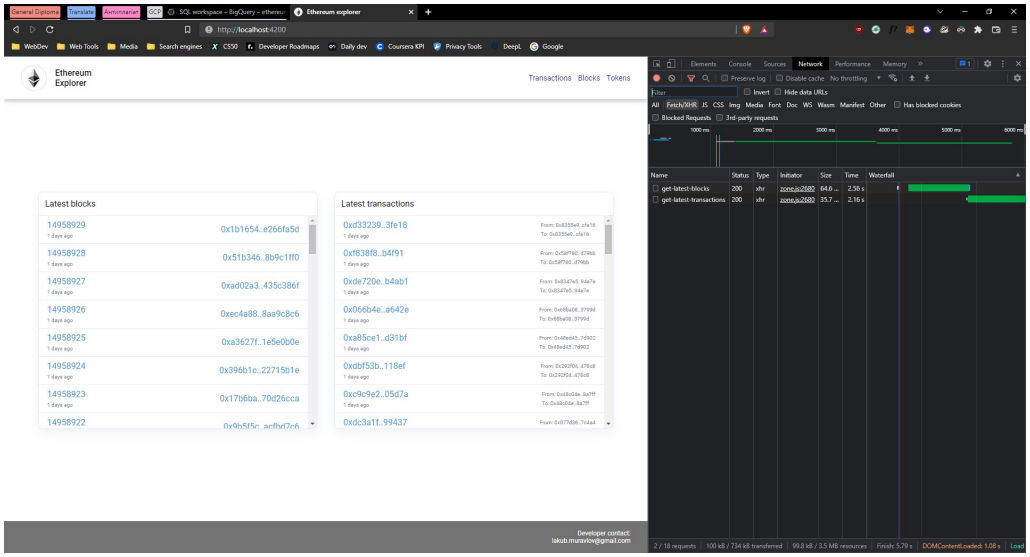


Рисунок 5.15. Результат роботи запиту

Бачимо, що обробка запитів займає в середньому 2 секунди.

Сторінки Transactions та Blocks будуть містити перелік всіх транзакцій та блоків, які надаються пакетами по 50 одиниць.

Сторінка токенів має такий же функціонал, однак якщо подивитись на таблицю токенів в датасеті можна побачити, що вона дуже неоднорідна:

tokens QUERY SHARE COPY SNAPSHOT DELETE EXPORT

SCHEMA DETAILS PREVIEW

Row	address	symbol	name	decimals	total_supply	block_timestamp	block_number
1	0xfe6151364713f3a8c5f1406df3f17fbfcb6b5deb	null	null	null	null	2019-11-06 02:25:21 UTC	8881215
2	0x547f05402071b30466deb598e75e908b49e06d0b	null	null	null	null	2019-11-11 06:39:51 UTC	8912936
3	0xd8d53eb00399ecb782233c7ba707b639fa14af2c	null	null	null	null	2019-11-03 12:30:28 UTC	8865134
4	0x88c8cf3a212c0369698d13fe98fcb76620389841	null	null	null	null	2020-04-02 08:36:23 UTC	9791519
5	0x867265021494b8f6e00411ba016f4c0621c79dfe	null	null	null	null	2019-11-14 06:09:29 UTC	8930859
6	0x03046a4875bf6a8c16e089678fd3491edf5dcf2e	null	null	null	null	2019-10-30 07:00:32 UTC	8838866
7	0x6e90b9de939e7d34542bc26c5ca32a488a3986d0	null	null	null	null	2019-11-01 16:12:55 UTC	8853621
8	0x56b498737c28e0c98c600545fdcb96949ffc91ea	null	null	null	null	2019-11-10 08:28:47 UTC	8907462
9	0xaaaacedf439e3d75c37b4e05a3024afd8bfafec4	null	null	null	null	2019-11-06 16:23:48 UTC	8884852
10	0xea5a478a5ec2a643075bbb3a8af0b9ad29a63065	null	null	null	null	2019-10-31 09:08:40 UTC	8845607
11	0x2ed0a347000adfee9e687c2a543090817deec23d	null	null	null	null	2019-11-15 06:49:13 UTC	8937004
12	0x3017e3d1d6ac30991a9aab04e07b81592c8a12cc	null	null	null	null	2019-11-09 06:40:22 UTC	8900875
13	0x230de5c8e8d2f3dbbadeba1ea4c1be19d782d174	CTR	CryptoCars	null	null	2019-08-03 03:32:11 UTC	8275309
14	0x1f2634e19785d08649baa6667fb945aa55916752	null	null	null	null	2019-11-01 10:17:36 UTC	8852087
15	0xc128585fa0e8c1b40e5915cc2dcdfe1b9bc13804	null	null	null	null	2019-11-01 08:01:57 UTC	8851528
16	0x5b20e5676c26696c8fc75aa3f1e6bdce2def8c6f	null	null	null	null	2017-09-25 10:29:03 UTC	4310125
17	0xda58c06b2acbb32c647b245ae85621efcc626dce	null	null	null	null	2018-04-06 05:23:23 UTC	5389338
18	0xfce43246ff755d91d0bca5dc73a6f9a59c421d07	null	null	null	null	2019-10-30 10:15:46 UTC	8839709
19	0xce52d7727fb77ca7b084f50bbbf335997fcb0dd3	null	null	null	null	2019-10-30 00:47:40 UTC	8837211
20	0x415f38ce740d496ba6cfa5e66163e9d85df24afd	null	null	null	null	2018-03-20 15:49:33 UTC	5289966
21	0x35e361914936257 61a16c9aba6484699e6e9931e	null	null	null	null	2019-11-12 18:39:34 UTC	8921963
22	0x128a559d714fd74630eaf28c5adc275e07e487ff	null	null	null	null	2017-03-15 16:26:31 UTC	3357127
23	0x51697de9c5b92aee9e2aafa6fb348cfa5545bc5f	null	null	null	null	2019-11-04 00:23:00 UTC	8868266

Рисунок 5.16. Попередній перегляд таблиці токенів

Токени без символу (скороченого позначення), кількості знаків після коми та загальної пропозиції не зможуть навіть сформувати транзакцію обміну, відправки або отримання, тому таблицю токенів необхідно опрацювати. Реалізуємо це за допомогою імплементації фреймворку MapReduce в GCP. Цей же функціонал буде використовувати строка пошуку, яка буде шукати транзакції, токени та блоки за адресою яку користувач буде вводити.

Для цього треба використовувати оркестрування сервісів, GCP підтримує декілька варіантів реалізації оркестрування сервісів, однак усі вони мають приблизно однаковий механізм — є тригер фукнкції (Cloud Storage Trigger, Pub/Sub), до цього тригера прив'язується функція яка в залежності від запиту виконує певний сервіс (цей функціонал реалізується за допомогою Cloud Workflow або Cloud Functions).

Архітектура оркестрування сервісами має настпуний вигляд:

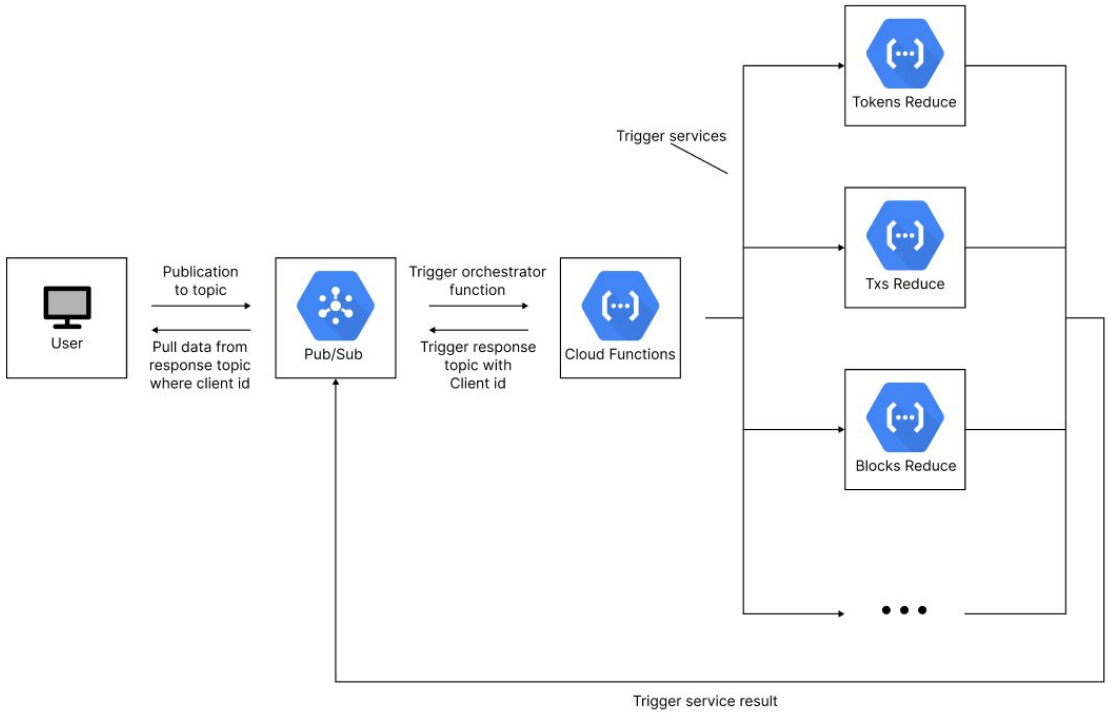


Рисунок 5.17. СОА з оркестрацією сервісами

Почнемо з реалізації обробки таблиці токенів до відображення. Для цього спочатку необхідно створити сервісний акаунт для сервісної шини (Pub/Sub) та також створити сервісний ключ на подобі того як це було зроблено для BigQuery:

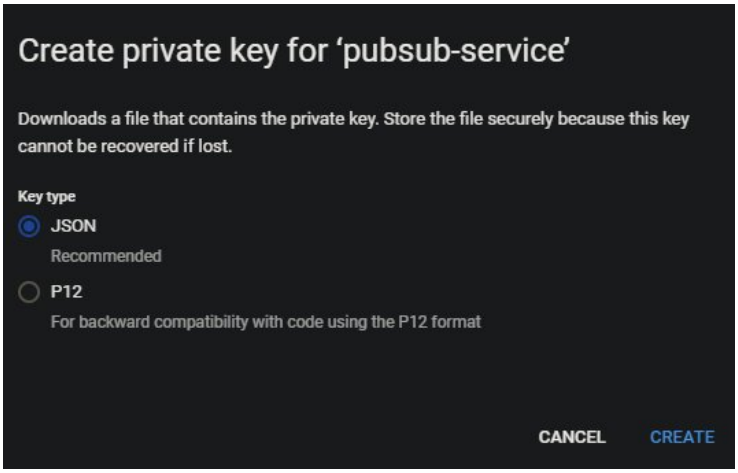


Рисунок 5.18. Створення ключа для сервісу шини

Також необхідно створити topic, ресурс на який надходять повідомлення від паблішера:

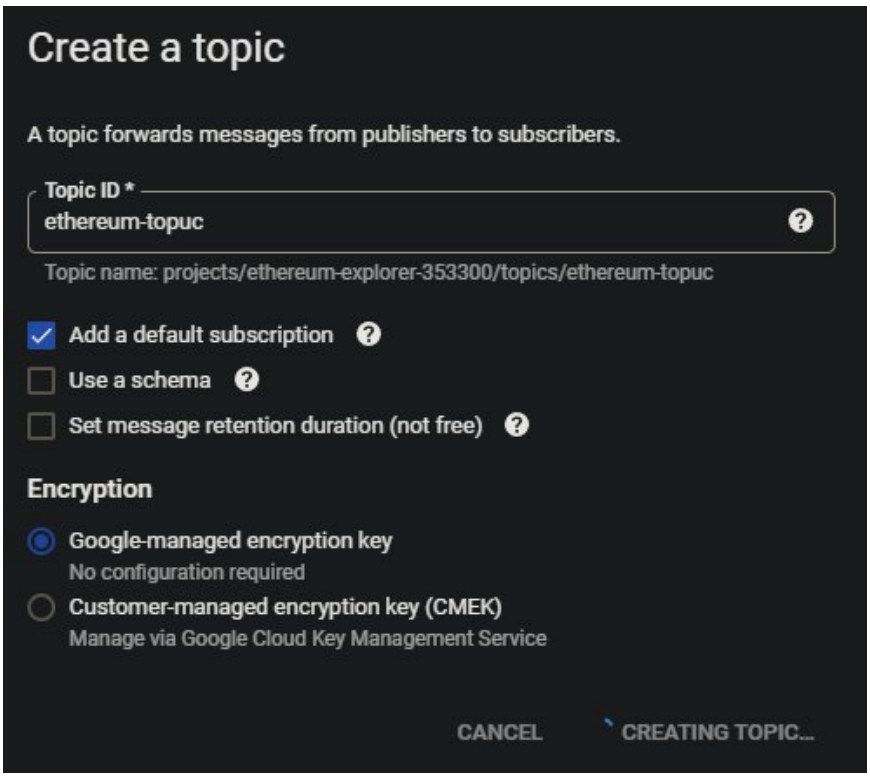


Рисунок 5.19. Створення топіку

Далі прив’яжемо функцію до цього топіку, це можна зробити перейшовши до топіку та нажавши «Trigger Cloud Function»:

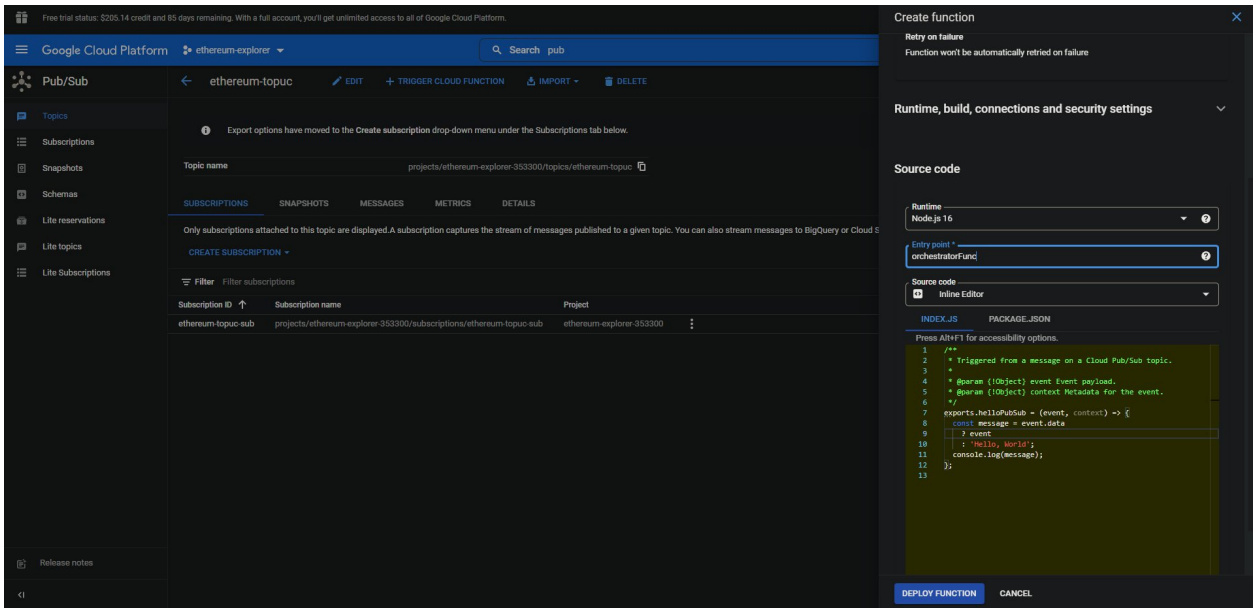


Рисунок 5.20. Прив’язка функції до топіку

Первіримо роботу сервісу, створимо контролер для отримання токенів, який буде звертатись до топіку:

```python
@app.route('/get-tokens', methods=['GET', 'POST'])
def get_tokens():       You, 17 minutes ago • Uncommitted changes
    requester = f'requester-{randint(0,10000)}-{randint(0,10000)}-{randint(0,10000)}-{randint(0,10000)}'
    ps_key = os.getcwd() + '\keys\keyPS.json'
    os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = ps_key
    publisher = pubsub_v1.PublisherClient()

    topic_path = 'projects/ethereum-explorer-353300/topics/ethereum-topuc'

    message = 'reduce-tokens'
    message = message.encode('utf-8')

    request = {
        'from': requester,
        'destination': 'token-service',
        'operation': 'reduce-tokens'
    }

    future = publisher.publish(topic_path, message, **request)

    return json.dumps({'result': 200, 'data': request})
```

Рисунок 5.21. Контролер отримання токенів.

Рисунок 5.22. Результат виконання хмарної функції.

Далі, на основі даних які оркестратор отримує, він викликає функції, які беспосередньо оброблюють дані, в залежності від поля operation:

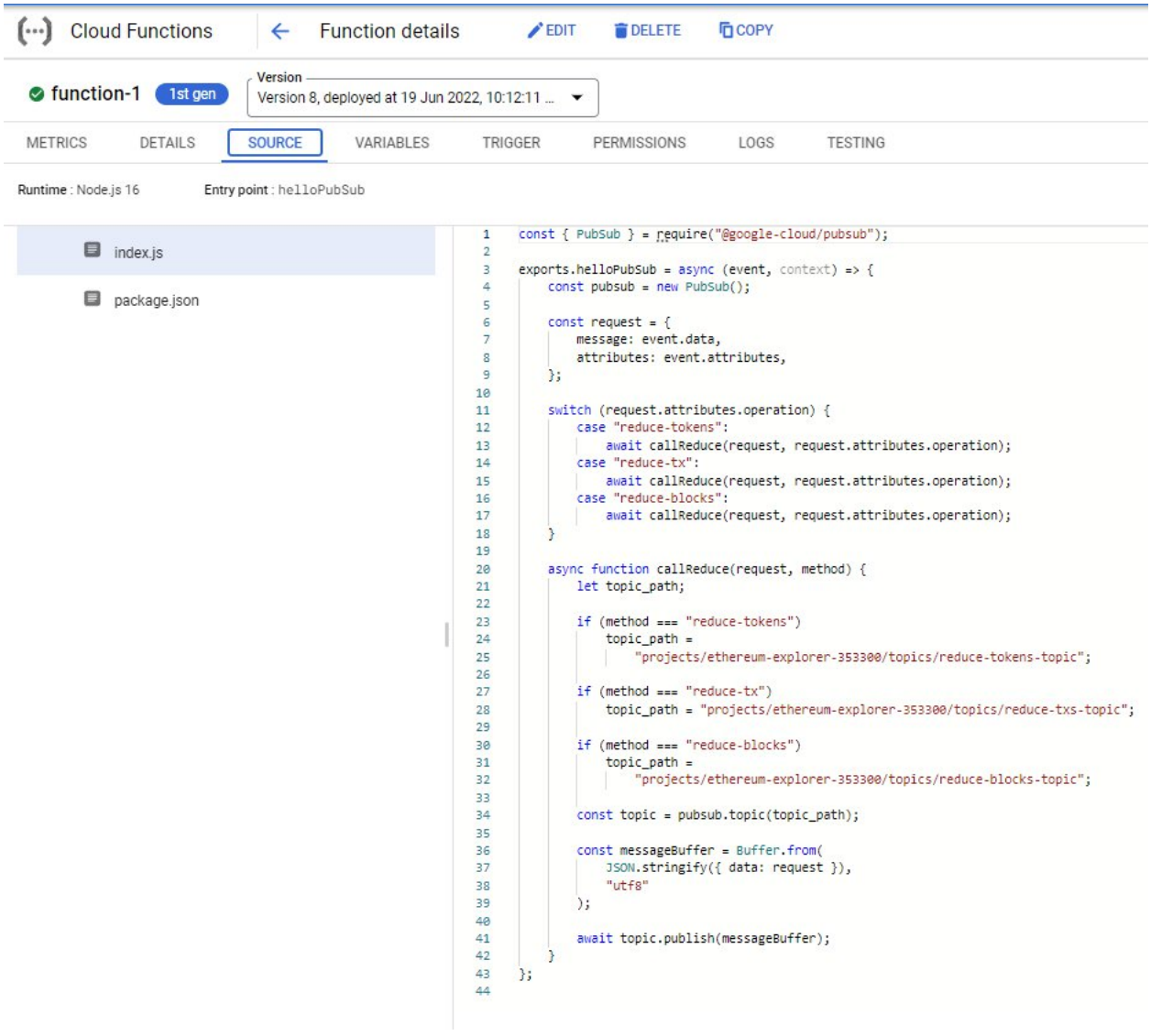


Рисунок 5.23. Структура оркестратора

В залежності від того, які методи та операції знаходяться в полях викликаються топіки, що прив’язані до відповідних сервісів. Сервіс для обробки токенів приймає в себе дані з оркестратора та в залежності від поля method, використовує MapReduce() щоб прибрати токени, значення яких не задовольняють вимоги (символ, кількість знаків після коми та загальна пропозиція не нульові), або щоб знайти співпадіння по підрядку адреси чи символу та надсилає повідомлення з ID користувача, що зробив запит, та методом на який було створено запит:

```
def reduce_tokens(event, context):
    req = json.loads(base64.b64decode(event['data']).decode('utf-8'))
    query = "SELECT * FROM `ethereum-explorer-353300.ethereum.tokens`"
    bq_client = bigquery.Client()
    query_job = bq_client.query(query)
    df = query_job.to_dataframe()

    tokens = processDf(df)

    result = None

    topic_path = 'projects/ethereum-explorer-353300/topics/response-topic'

    message = 'reduce-tokens'
    message = message.encode('utf-8')

    if(req['method'] == 'remove-null'):
        with beam.Pipeline() as pipeline:
            (pipeline | 'Tokens' >> beam.Create(tokens) | 'Filter null' >> beam.Filter(is_null) | beam.Map(mapToken))
        request = {
            'to': requester,
            'operation': 'reduce-tokens',
            'method': 'remove-null',
            'data': result
        }
    if(req['method'] == 'search'):
        substring = req['substring']
        with beam.Pipeline() as pipeline:
            (pipeline | 'Tokens' >> beam.Create(tokens) | 'Filter null' >> beam.Filter(is_null) | beam.Map(mapToken))
        request = {
            'to': requester,
            'operation': 'reduce-tokens',
            'method': 'remove-null',
            'data': result
        }

    publisher = pubsub_v1.PublisherClient()

    publisher.publish(topic_path, message, **request)
```

Рисунок 5.24. Сервіс обробки токенів

Сервіси обробки блоків працюють аналогічно, тільки приймають у себе лише метод пошуку. Створено наступні топіки:

Topic ID	Encryption key	Topic name
ethereum-topuc	Google-managed	projects/ethereum-explorer-353300/topics/ethereum-topuc
reduce-blocks-topic	Google-managed	projects/ethereum-explorer-353300/topics/reduce-blocks-topic
reduce-tokens-topic	Google-managed	projects/ethereum-explorer-353300/topics/reduce-tokens-topic
reduce-txs	Google-managed	projects/ethereum-explorer-353300/topics/reduce-txs
response-topic	Google-managed	projects/ethereum-explorer-353300/topics/response-topic

Рисунок 5.25. Топіки, створені відповідно архітектури

Останній топік надсилає респонси з сервісів. Також відповідно топіків створено функції-сервіси (FaaS) які приймають в себе дані від оркестратора:

		Environment	Name ↑	Region	Trigger
☐	✓	1st gen	function-1	us-central1	Topic: ethereum-topuc
☐	✓	1st gen	function-2	us-central1	Topic: reduce-tokens-topic
☐	✓	1st gen	function-3	us-central1	Topic: reduce-txs
☐	✓	1st gen	function-4	us-central1	Topic: reduce-blocks-topic

Рисунок 5.26. Сервіси, що використовуються в додатку

Додамо до ендпойнтів підписку на респонси з сервісної шини:

```python
@app.route('/get-tokens', methods=['GET', 'POST'])
def get_tokens():
    req = request.get_json()
    page = req['page']
    requester = f'requester-{randint(0,10000)}-{randint(0,10000)}-{randint(0,10000)}-{randint(0,10000)}'
    ps_key = os.getcwd() + '\keys\keyPS.json'
    os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = ps_key
    publisher = pubsub_v1.PublisherClient()

    topic_path = 'projects/ethereum-explorer-353300/topics/ethereum-topuc'

    message = 'reduce-tokens'
    message = message.encode('utf-8')

    request_tokens = {
        'from': requester,
        'destination': 'token-service',
        'operation': 'reduce-tokens',
        'method': 'remove-null',
        'offset_index': f'{page}'
    }

    pub_future = publisher.publish(topic_path, message, **request_tokens)

    def callback(message):
        if message['to'] == requester and message['operation'] == 'reduce-tokens':
            return json.dumps({'result': 200, 'data': message['tokens']})

    with pubsub_v1.SubscriberClient() as subscriber:
        future = subscriber.subscribe(sub_name, callback)
```

Рисунок 5.27. Підписка на топік респонсів

Висновки до розділу 5

У даному розділі було описано основні потоки роботи з даними для кожного з хмарних провайдерів сервісів. Розглянуто переваги і недоліки кожного з них. Обрано провайдера для реалізації додатку — Google Cloud Platform.

Знайдено відкритий датасет у маркетплейсі GCP з даними блокчейну Ethereum, який оновлюється кожного дня. На основі датасету сформовано потреби до додатку який необхідно реалізувати.

З вмісту датасету випливає, що можна створити блокчейн експлорер, який буде дозволяти шукати транзакції, токени та блоки, та інформацію про них. Розглянуто підходи:

- Використання REST при взаємодії з сервісом
- Використання сервісної шини та оркестрування сервісів
- Використання фреймворку MapReduce() при роботі з великими даними

Користувацький інтерфейс реалізовано за допомогою фреймворку Angular, серверсайд та логіку реалізовано на мові Python. Додаток протестовано за допомогою метода приймального тестування.

РОЗДІЛ 6. Функціонально-вартісний розрахунок продукту

У цьому розділі оцінено основні функціональні та вартісні характеристики програмного продукту, розробленого для вирішення проблеми фільтрації образливих висловлювань у соціальних мережах.

Програмне забезпечення призначено для використання на ПК під керуванням будь-якої операційної системи.

Далі проаналізовано різні варіанти реалізації модуля з метою вибору оптимальної стратегії побудови програмного продукту з урахуванням економічних факторів та характеристик продукту, що впливають на його продуктивність та сумісність з апаратним забезпеченням. Для цього використано інструмент аналізу функціональної цінності.

Функціонально-вартісний аналіз - це спосіб оцінки реальної вартості товару чи послуги, який залежить від організаційної структури підприємства. Прямі та непрямі витрати розподіляються між продуктами та послугами відповідно до обсягу ресурсів, що є на кожному етапі виробництва. Дії, які виконуються цих етапах, у методі ФВА називаються функціями.

Мета ФВА - забезпечити оптимальне розподілення ресурсів, виділених на виробництво продукту або послуги, на прямі та непрямі витрати. У цьому випадку йдеться про аналіз функцій програмного продукту та визначення всіх витрат, пов'язаних з реалізацією цих функцій.

Підхід ФВА починається з визначення послідовності функцій, необхідних при виробництві продукту. Спочатку розглядаються всі можливі функції та поділяються на дві групи: ті, що впливають на вартість продукту, та ті, що не впливають. На цьому етапі оптимізується і сама послідовність дій шляхом скорочення кількості етапів, що не впливають на вартість.

Для кожної операції визначається загальна вартість та кількість робочих годин на рік. За підсумками цих оцінок визначаються кількісні характеристики

джерел витрат. Після визначення джерел витрат розраховується кінцева вартість виробленого товару.

6.1. Постановка задачі проектування

Підхід ФВА був застосований для аналізу придатності використання сервіс-орієнтованої архітектури при роботі з великими даними. Основні проектні рішення відносяться до всієї системи, і ці рішення мають бути враховані у кожній підсистемі. Таким чином, фактичний аналіз був функціональним аналізом програмного продукту для обробки та фільтрації даних.

Технічні вимоги до цього продукту:

1. Продукт повинен працювати на комп'ютері зі стандартною конфігурацією веб-браузера.
2. Висока швидкість обробки та реакція користувача в режимі реального часу.
3. Простота використання та взаємодії.
4. Простота конфігурування, розробки та обслуговування.
5. Низькі витрати на встановлення програмного забезпечення.

6.1.1. Обґрунтування функцій програмного продукту

Головною функцією F0 є розробка програмного продукту. Виходячи з поставленого завдання можна виділити такі основні функції:

F1 – вибір провайдерів хмарних сервісів;

F2 - вибір мови програмування на бекенд;

F3 – вибір середовища розробки.

Будь-яка з основних функцій може мати кілька реалізацій.

Функція F1:

a) Microsoft Azure
b) Google Cloud Platform
c) AWS

Функція F2:

a) Python
b) Node.js
c) Java

Функція F3:

a) Microsoft VS Code
b) Pycharm
c) IntelliJ

6.1.2. Варіанти реалізації основних функцій

Варіанти реалізації основних функцій показані на морфологічній діаграмі системи у рисунку 6.1. На морфологічній діаграмі показані всі можливі комбінації функціональних варіантів, що становлять повний варіант ПП:

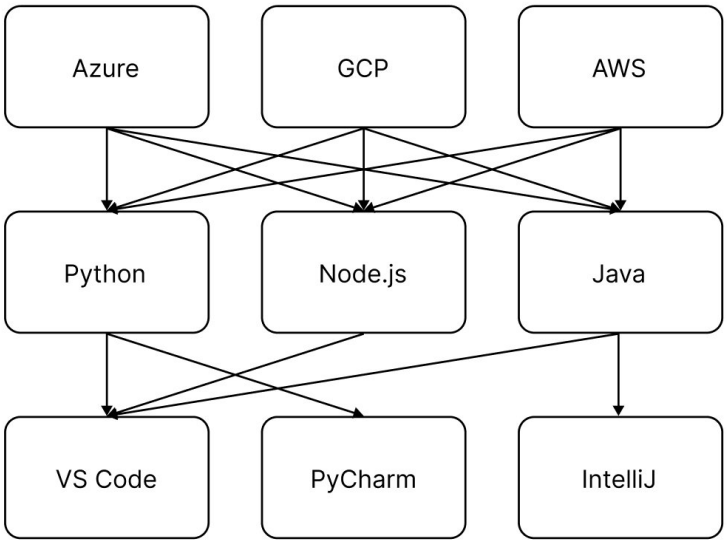


Рисунок 6.1. Морфологічна діаграма

Функція	Варіант реалізації	Переваги	Недоліки
F_1	a	Великий спектр інструментів, детальна документація, інтеграція з іншими продуктами Microsoft, гарна підтримка serverless рішень	Деякі сервіси надають повний функціонал лише при більш дорогому плані оплати, погана підтримка для продуктів та ПЗ, що не належать Microsoft
	b	Великий спектр інструментів, тісна інтеграція з продуктами Google, гарна підтримка сервісів для роботи з аналітикою даних	Погана документація у порівнянні з конкурентами
	c	Великий спектр існтрументів та сервісів, велика розповсюдженість та комьюніті, детально описана документація	Прив’язка до платформи при використанні serverless сервісів
F_2	a	Кросплатформена, інтерпретована, популярна і має широку підтримку багатьох сервісів, тісно пов’язана з аналізом даних	Відносно повільна
	b	Кросплатформена, інтерпретована, має багато пакетів у репозиторії пакетів, популярна	Відносно невелика підтримка серед провайдерів хмарних сервісів, одно-поточна
	c	Кросплатформена, має велику підтримку в хмарних сервісах, багатопоточна	Відносно повільна, виконується в ВМ, instant legacy синтаксис
F_3	a	Легковісний, розширяється плагінами, гнучкі налаштування, зручний інтерфейс, обширна інтеграція з продуктами Microsoft	З коробки не має функціоналу повноцінного середовища розробки
	b	Обширна підтримка Python, зручний інтерфейс, інтеграція з іншими продуктами JetBrains	Повний функціонал доступний лише у платній версії
	c	Обширна підтримка Java, зручні інструменти налаштування Java проекту, інтеграція з іншими продуктами JetBrains	Повний функціонал доступний лише у платній версії

Таблиця 6.1. Позитивно-негативна матриця варіантів основних функцій

З аналізу позитивно-негативної матриці можна дійти невтішного висновку, що у процесі розробки програмного продукту деякі функціональні

реалізації доводиться відкидати, оскільки де вони відповідають цілям програмного продукту. Ці альтернативи зазначено на карті морфологічного аналізу.

Функція F_1:

Оскільки програмне рішення потребує роботи з даними, необхідно обрати інструмент з найкращою підтримкою Big Data, тому надаємо перевагу варіанту В.

Функція F_2:

Варіант В має низьку підтримку на провайдерах сервісів, та майже не прописаний в документаціях їх сервісів та рішень, тому цей варіант не доцільно використовувати. Варіанти А та С вважаємо рівнозначними.

Функція F_3:

Варіант А підтримує усі мови програмування, що розглядаються для використання, дозволяє не використовувати різні програмні рішення на фронтенд та бекенд, тому надаємо перевагу йому.

Таким чином будемо розглядати наступні варіанти реалізації програмного продукту:

— F_1(В) — F_2(А) — F_3(А)

— F_1(В) — F_2(С) — F_3(А)

6.2. Обґрунтування системи параметрів програмного продукту

З основних функцій і вимог до програмного продукту, визначаються ключові параметри продукту, на основі яких розраховується коефіцієнт технічного рівня.

Для характеристики програмного продукту будемо використовувати такі параметри:

— X1 - швидкодія обраної мови програмування

— X2 - кількість сервісів для обробки та збереження даних

— X3 - час, необхідний на реалізацію продукту

— X4 - потенційний об’єм програмного коду

Найгірше, середнє і краще значення параметрів вибираються з вимог замовника і умов, що характеризують працездатність програмного продукту (див. таблицю 6.2.).

Опис параметру	Умовні позначення	Одиниці виміру	Значення параметру		
			Найгірше	Середнє	Краще
Швидкодія мови програмування	X1	мс/оп	30000	16000	2000
Кількість сервісів для роботи з даними	X2	одиниць	2	5	10
Час, необхідний на реалізацію продукту	X3	годин	300	240	200
Потенційний об’єм програмного коду	X4	строк	8000	5000	2000

Таблиця 6.2. Основні параметри програмного продукту

За даними таблиці будуються графічні характеристики параметрів:

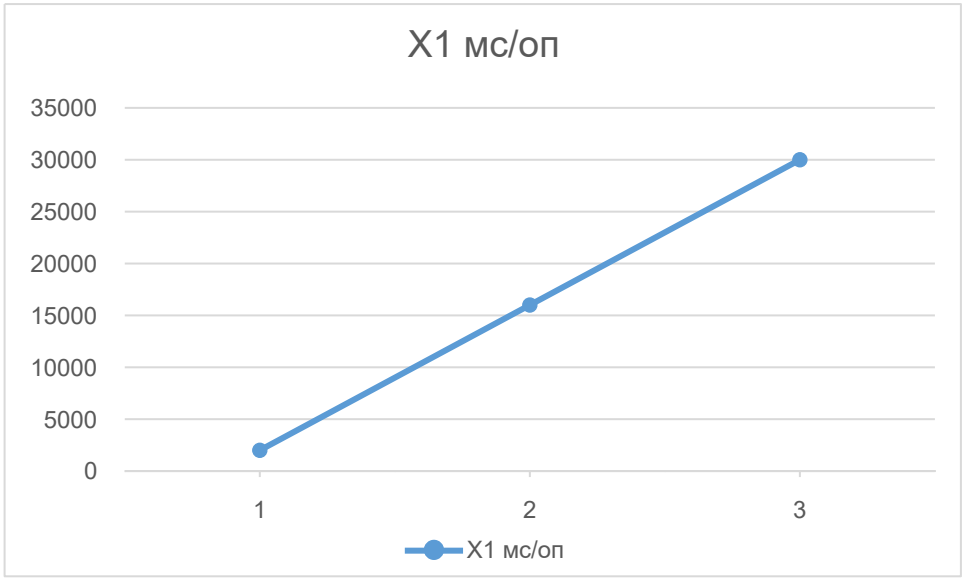


Рисунок 6.2. Швидкодія мови програмування

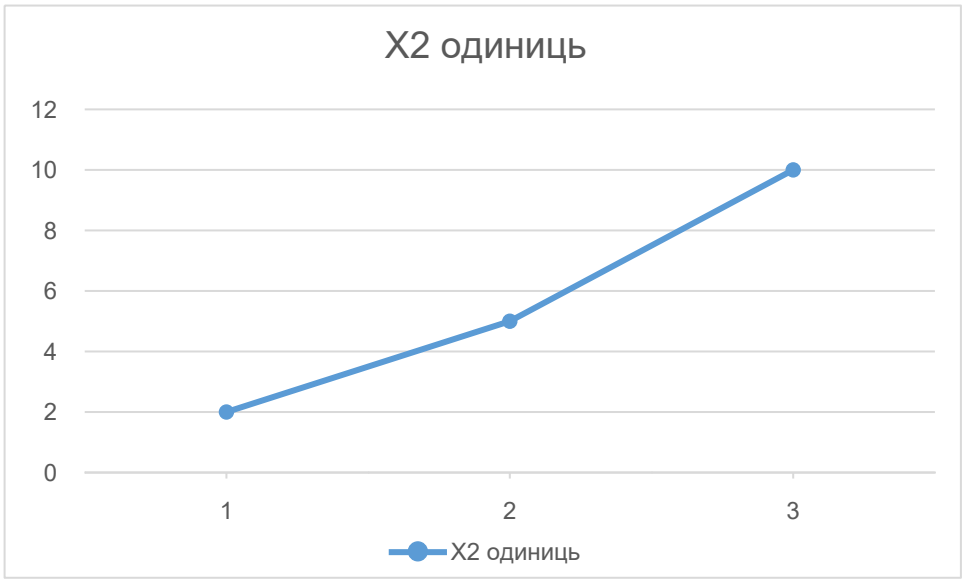


Рисунок 6.3. Кількість сервісів для роботи з даними

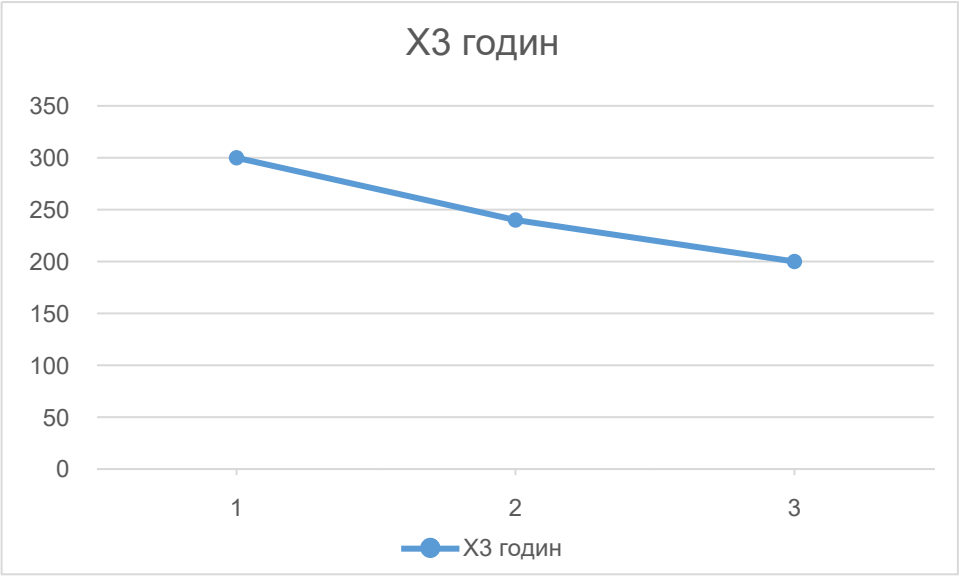


Рисунок 6.4. Час, необхідний на реалізацію продукту

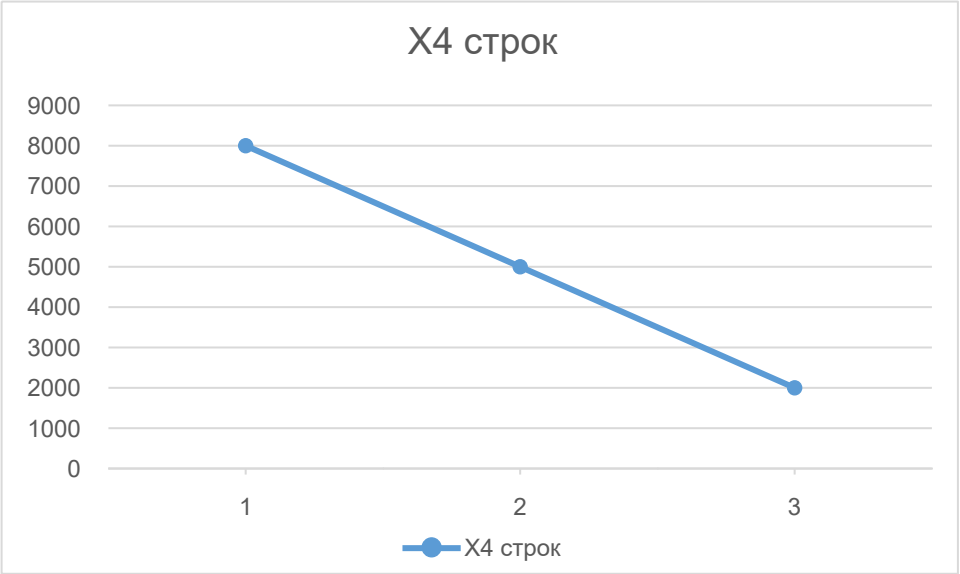


Рисунок 6.5. Потенційний об’єм програмного коду

6.3. Аналіз експертного оцінювання параметрів

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 5 людей. Визначення коефіцієнтів значимості передбачає:

1) визначення рівня значимості параметра шляхом присвоєння різних рангів;

2) перевірку придатності експертних оцінок для подальшого використання;

3) визначення оцінки попарного пріоритету параметрів;

4) обробку результатів та визначення коефіцієнту значимості.

Параметр	**Ранг параметру за оцінкою експерта**					**Сума рангів**	**Відхилення, Δ_i**	**Δ_i^2**
	1	**2**	**3**	**4**	**5**			
X1	1	1	2	2	1	7	-5.5	30.25
X2	2	3	1	1	2	9	-3.5	12.25
X3	4	2	4	3	3	16	3.5	12.25
X4	3	4	3	4	4	18	5.5	30.25
Разом	10	10	10	10	10	50	0	85

Таблиця 6.3. Результати ранжування показників

Для перевірки ступеню достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} * r_{ij} = 50, \quad (1)$$

де r_{ij} – ранг *i*-го параметра, визначений *j*-м експертом;

N – число експертів.

б) середня сума рангів *T*:

$$T = \frac{1}{n} R_i = 12{,}5. \quad (2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T. \quad (3)$$

г) загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{n} * \Delta_i^2 = 85. \quad (4)$$

д) коефіцієнт узгодженості (конкордації):

$$W = \frac{12S}{N^2\left(n^3 - n\right)} = \frac{12 \cdot 85}{5^2\left(4^3 - 4\right)} = 0{,}68 > W_k = 0{,}67. \quad (5)$$

Ранжирування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67. Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 6.3. Числове значення, що визначає ступінь переваги *i*-го параметра над *j*-тим, a_{ij} визначається за формулою:

$$a_{ij} = \{1{,}5 \;\; x_i > x_j;\; 1{,}0 \;\; x_i = x_j;\; 0{,}5 \;\; x_i < x_j. \quad (6)$$

Параметр	**Експерти**					**Підсумкова оцінка**	**Числове значення коефіцієнтів переваги**
	1	**2**	**3**	**4**	**5**		
X1, X2	<	<	>	>	<	<	0.5
X1, X3	<	<	<	<	<	<	0.5
X1, X4	<	<	<	<	<	<	0.5
X2, X3	<	>	<	<	<	<	0.5
X2, X4	<	<	<	<	<	<	0.5
X3, X4	>	<	>	<	<	<	0.5

Таблиця 6.3. Результати ранжування параметрів

З отриманих числових оцінок переваги складемо матрицю $A = \|a_{ij}\|$. Для кожного параметра розрахунок вагомості K_{B_i} проводиться за наступною формулою:

$$K_{B_i} = \frac{b_i}{\sum_{i=1}^{n} * b_i}, \quad (7)$$

де $b_i = \sum_{j=1}^{N} a_{ij}$ – вагомість *i*-го параметра за результатами оцінок всіх експертів;

a_{ij} –коефіцієнт переваги *i*-го на *j*-тим параметром.

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%). На другому і наступних кроках відносні оцінки розраховуються за наступною формулою:

$$K_{B_i} = \frac{b_i'}{\sum_{i=1}^{n} * b_i'}, \quad (8)$$

де $b_i' = \sum_{j=1}^{N} a_{ij} b_j$.

Як видно з таблиці 6.4., різниця значень коефіцієнтів вагомості після другої ітерації не перевищує 2%, тому додаткові ітерації не потрібні.

i	*j*				Перша ітерація		Друга ітерація	
	X1	X2	X3	X4	B_i	K_{B_i}	B_i^1	$K_{B_i}^1$
X1	1,0	0,5	0,5	0,5	2,5	0,156	9,25	0,157
X2	1,5	1,0	0,5	0,5	3,5	0,219	12.25	0,208
X3	1,5	1,5	1,0	0,5	4,5	0,281	16.25	0,275
X4	1,5	1,5	1,5	1,0	5,5	0,344	21.25	0,36
Разом					16,0	1,0	59	1,0

Таблиця 6.4. Розрахунок вагомості параметрів

6.4. Аналіз рівня якості варіантів реалізації функцій

Рівень якості кожного варіанту виконання основних функцій визначається окремо.

Абсолютні значення параметрів X1(швидкодія мови програмування) та X2 (об’єм пам’яті для збереження даних) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3 (час, необхідний на реалізацію ПП) обрано не найгіршим, тобто це значення відповідає варіанту b) 240 год. або c) 200 год.

Абсолютне значення параметра X4 (потенційний об’єм коду) обрано не найгіршим, тобто це значення відповідає варіанту b) 5000 або c) 2000 строк коду.

Коефіцієнт технічного рівня якості для кожного варіанта реалізації ПП розраховується за формулою:

$$K_{TP} = \sum_{i=1}^{n} K_{B_i} B_i, \qquad (9)$$

де n – кількість параметрів;

K_{B_i} – коефіцієнт вагомості і–го параметра;

B_i – оцінка i–го параметра в балах.

Розрахунок показників рівня якості представлено відповідно в таблиці 6.5.

Основні функції	Варіант реалізації	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F_1(X1)	B	16000	6	0,157	0.942
F_2(X3)	A	200	10	0,275	2.75
	C	240	5	0,275	1.375
F_2(X4)	A	2000	10	0,36	3.6
	C	5000	5	0,36	1.8
F_3(X2)	A	5	5	0,208	1.04

Таблиця 6.5. Розрахунок показників якості

За цими даними визначаємо рівень якості кожного з варіантів:

— F_1(B) — F_2(A) — F_3(A) = 0.942 + 2.75 + 3.6 + 1.8 = 9.092

— F_1(B) — F_2(C) — F_3(A) = 0.942 + 1.375 + 1.8 + 1.04 = 5.157

Отже найкращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

6.5. Економічний аналіз варіантів розробки програмного продукту

Щоб визначити вартість розробки програмного продукту, спочатку розрахуйте витрати.

Усі варіанти передбачають виконання двох різних завдань:

1. проектування архітектури служби обробки даних;

2. розробка програмного забезпечення відповідно до архітектури.

Завдання 1 віднесено до групи А за новизною, а завдання 2 - до групи Б за новизною. Алгоритм, що використовується в завданні 1, знаходиться за складністю у групі 1, а алгоритм, що використовується у завданні 2, знаходиться у групі 3.

Для реалізації завдання 1 використовує інформацію у вигляді даних, а завдання 2 використовує стандартні методи збору даних. Розрахуємо вартість та час розробки для кожного завдання. Загальні трудові витрати розраховуються за такою формулою (10).

$$\mathrm{Т_О} = \mathrm{Т_Р}\cdot\mathrm{К_П}\cdot\mathrm{К_{СК}}\cdot\mathrm{К_М}\cdot\mathrm{К_{СТ}}\cdot\mathrm{К_{СТ.М}} \qquad (10)$$

де $\mathrm{Т_Р}$ – трудомісткість розробки програмного продукту;

$\mathrm{К_П}$ – поправочний коефіцієнт;

$\mathrm{К_{СК}}$ – коефіцієнт на складність вхідної інформації;

$\mathrm{К_М}$ – коефіцієнт рівня мови програмування;

$\mathrm{К_{СТ}}$ – коефіцієнт використання стандартних модулів і прикладних

програм;

$К_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: $Т_Р = 72$ людино-днів. Поправочний коефіцієнт, який враховує вид вхідної інформації для першого завдання: $К_П = 1,7$. Поправочний коефіцієнт, який враховує складність контролю вхідної та вихідної інформації рівний $К_{СК} = 1$. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ} = 0,85$. Тоді загальна трудомісткість розробки першого завдання дорівнює:

$$Т_1 = 72 \cdot 1.7 \cdot 0.85 = 104.05 \text{ людино-днів} \quad (11)$$

Проведемо аналогічні розрахунки для другого завдання, в якому використовується алгоритм третьої групи складності зі ступенем новизни Б, тобто $Т_Р = 29$ людино-днів, $К_П = 0,9$, $К_{СК} = 1$, $К_{СТ} = 0,8$:

$$Т_2 = 29 \cdot 0.9 \cdot 0.8 = 20.88 \text{ людино-днів.} \quad (12)$$

Оскільки загальна трудомісткість усіх варіантів реалізації збігаються, їх можна об’єднати в одну групу.

Загальна трудомісткість складає:

$$Т_О = (104,05 + 20,88) \cdot 8 = 999,44 \text{ людино-годин;} \quad (13)$$

В розробці беруть участь програміст з окладом 26500 грн., один архітектор з окладом 29000 грн, та один інженер даних з окладом 19500 грн.

Визначимо середню зарплату за годину за формулою:

$$С_Ч = \frac{M}{T_m \cdot t} \text{грн.,} \quad (14)$$

де M – місячний оклад працівників;

T_m – кількість робочих днів на місяць;

t – кількість робочих годин в день.

$$С_Ч = \frac{26500 + 29000 + 19500}{3 \cdot 20 \cdot 8} = 217{,}25 \text{ грн.} \quad (15)$$

Тоді, розрахуємо заробітну плату за формулою:

$$С_{ЗП} = С_Ч \cdot T_i \cdot К_Д, \quad (16)$$

де $С_Ч$ – величина погодинної оплати праці програміста;

T_i – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

$$С_{ЗП} = 217{,}25 \cdot 999{,}44 \cdot 1{,}2 = 260\,554 \text{ грн.} \quad (17)$$

Відрахування на соціальний внесок становить 22%:

$$С_{СВ} = С_{ЗП} \cdot 0{,}22 = 260\,554 \cdot 0{,}22 = 57\,321{,}88 \text{ грн.} \quad (18)$$

Тепер визначимо витрати на оплату однієї машино-години. Оскільки одна ЕОМ обслуговується одним інженером апаратного забезпечення з окладом 12000 грн. та коефіцієнтом зайнятості $К_З = 0{,}2$ то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 12\,000 \cdot 0{,}2 = 28\,800 \text{ грн.} \quad (19)$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 28\,800 \cdot (1 + 0{,}2) = 34\,560 \text{ грн.} \quad (20)$$

Відрахування на соціальний внесок становить 22%:

$$С_{СВ} = С_{ЗП} \cdot 0{,}22 = 34\,560 \cdot 0{,}22 = 7\,603.2 \text{ грн.} \quad (21)$$

Амортизаційні відрахування розраховуємо за формулою при амортизації 25% та вартості ЕОМ – 33 500 грн.:

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1{,}15 \cdot 0{,}25 \cdot 33500 = 9\,631.25 \text{ грн.} \quad (22)$$

де $К_{ТМ}$ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

$К_А$ – річна норма амортизації;

$Ц_{ПР}$ – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо за формулою:

$$С_Р = К_{ТМ} \cdot К_Р \cdot Ц_{ПР} = 1{,}15 \cdot 0{,}05 \cdot 33\,500 = 1\,926.25 \text{ грн.} \quad (23)$$

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t \cdot К_В, Т_{ЕФ} = (365 - 104 - 12 - 16) \cdot 8 \cdot 0{,}9 = 1\,667{,}6 \text{ год.} \quad (24)$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$,$Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_C \cdot К_З \cdot Ц_{ЕЛ} = 1\,677{,}6 \cdot 0{,}65 \cdot 0{,}35 \cdot 3{,}6352 = 1387{,}39 \text{ грн.} \quad (25)$$

де N_C – середньо-споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕЛ}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$$С_Н = Ц_{ПР} \cdot 0{,}67 = 33\,500 \cdot 0{,}67 = 22\,445 \text{ грн.} \quad (26)$$

Тоді, річні експлуатаційні витрати будуть складати:

$$C_{ЕК} = C_{ЗП} + C_{СВ} + C_{А} + C_{Р} + C_{ЕЛ} + C_{Н}$$
$$= 34\,560 + 7\,603.2 + 9\,631.25 + 1\,926.25 + 1387.39 + 22\,445$$
$$= 77553.09 \text{ грн.} \quad (27)$$

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$C_{МГ} = \frac{C_{ЕК}}{T_{ЕФ}} = \frac{77553.09}{1\,667,6} = 46,51 \text{ грн/год.} \quad (28)$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу складають:

$$C_{М} = C_{МГ} \cdot T = 46,51 \cdot 999,44 = 46\,483,95 \text{ грн.} \quad (29)$$

Накладні витрати складають 67% від заробітної плати:

$$C_{Н} = C_{ЗП} \cdot 0,67 = 260\,554 \cdot 0,67 = 174\,571.18 \text{ грн.} \quad (30)$$

Отже, вартість розробки програмного продукту за варіантами становить:

$$C_{ПП} = C_{ЗП} + C_{СВ} + C_{М} + C_{Н}$$
$$= 260\,554 + 57\,321,88 + 46\,483,95 + 174\,571.18$$
$$= 538931.01 \text{ грн.} \quad (31)$$

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕР} = \frac{К_{К}}{C_{ПП}} = \frac{9.092}{538931.01} = 1.69 \cdot 10^{-5} \quad (32)$$

Висновки до розділу 6

Економічна частина систематизує та поглиблює теоретичні знання з галузі економіки та управління виробництвом та застосовує їх у техніко-економічному обґрунтуванні з використанням функціонально-вартісного аналізу.

На основі даних про зміст основних функцій, які має виконувати програмне забезпечення, було визначено чотири найбільш перспективний

варіант впровадження продукту, з найбільшим значенням фактора техніко-економічного рівня, вартість $С_{ПП}$ = 538931.01 грн.

Цей варіант включає у себе

- Використання провайдера хмарних сервісів Google Cloud Platform.
- Використання мови програмування Python для бекенд розробки.
- Середовище розробки Microsoft VS Code.

ВИСНОВКИ

Великі дані є відносно молодою сферою, але сферою, яка вже на сьогоднішній день користується неймовірним попитом та має велкий простір для застосування — від полегшення бізнес аналітики до автоматизації інтернету речей, логістики, медицини, тощо. Однак переваги великих даних водночас є великими технічними випробовуваннями і традиційні підходи розширення датацентрів та масштабування програмного забезпечення просто не справляються зі швидкістю з якою дані надходять та їх об’ємом. Сервіс-орієнтована архітектура дозволяє вирішити велику кількість недоліків великих даних — надаючи можливість максимально абстрагувати та інкапсулювати їх обробку та заощадити на апаратному забезпеченні та розширені ПЗ, завдяки реюзабельності сервісів. Саме тому у цій роботі було прийнято рішеня розглянути парадигму сервіс-орієнтованої архітектури та її застосування у великих даних.

Було проаналізовано основні поняття та інструменти СОА, розглянуто механізми та рішення які надають провайдери хмарних сервісів для роботи з даними, серед яких GCP, AWS та Azure. При виборі хмарного сервіса для реалізації приділено увагу тому, що сервіси для роботи з даними від GCP зараз користуються надзвичайним попитом як у роботодавців так і в розробників та дата аналітиків.

Було дослідженно відкриті датасети великих даних, серед яких було обрано датасет з інформацією про on-chain транзакції, аккаунти, токени, блоки, контракти та логи блокчейну Ethereum. Криптовалюта є одним з найкращих прикладів великих даних, щосекунди генерується неймовірна кількість даних які необхідно опрацювати і цей датасет дозволяє зробити саме це. Виходячи з його властивостей було реалізовано додаток блокчейн експлорера.

При виборі фреймворку для написання користувацького інтерфейсу блуо обрано Angular, оскільки він дозволяє чітко розбити бізнес-логіку та компонентні рішення, а також підтримується та розробляється компанією Google, тому добре підтримує будь яке рішення яке необхідне для взаємодії з GCP. При виборі стеку для серверсайд процедур було обрано мову програмування Python через прекрасну підтримку в GCP та Apache Spark, читабельність коду та широкий функціонал для роботи з даними, що тільки розширюється такими бібліотеками, як pandas та matplotlib. В якості фреймворка для взаємодії з клієнтською стороною обрано Flask.

Окремо було розглянуто реалізацію механізмів взаємодії з сервісами описаних в роботі, а саме:

- REST запити напряму до сервісу
- Оркестрування сервісів за допомогою сервісної шини та Dataproc Workflows
- Реалізація рішення MapReduce() від GCP

Швидкодія додатку вражає, в тому числі тим, що при використанні традиційних рішень для обробки даних операції які використовуються займали б значно довше часу, а на досягнення такої ж швидкодії необхідно витратити набагато більше ресурсів на масштабування та розширення систем.

ПЕРЕЛІК ПОСИЛАНЬ

1. How Much Data Do We Create Every Day? The Mind-Blowing Stats Everyone Should Read [Електронний ресурс] - Режим доступу до ресурсу: https://www.forbes.com/sites/bernardmarr/2018/05/21/how-much-data-do-we-create-every-day-the-mind-blowing-stats-everyone-should-read/?sh=5cd457ef60ba
   Дата доступу - 11.05.2021
2. Paper AGW. ‘Big Data:’ big challenge, big opportunity; 1–6
3. O’Reilly Media I. Big Data Now: 2012 Edition. O’Reilly Media: Sebastopol, CA, 2012.
4. Taube BB, Solutions SG, Corporation V. Leveraging Big Data and real-time analytics to achieve situational awareness for smart grids; 1–20.
5. Casado R. The three generations of Big Data processing. In Big Data Spain, 2013.
6. Neumeyer L, Robbins B, Nair A, Kesari A. S4: distributed stream computing platform. In 2010 IEEE International Conference on Data Mining Workshops, 2010; c. 170–177.
7. The three V’s of big data [Електронний ресурс] - Режим доступу до ресурсу: https://subscription.packtpub.com/book/data/9781839219535/23/ch23lvl1sec19/the-three-v-s-of-big-data
   Дата доступу - 11.05.2021
8. Marz N, Warren J. Big Data Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co.: Shelter Island, NY, 2014; c. 425.

9. Breur, Tom (July 2016). "Statistical Power Analysis and the contemporary "crisis" in social sciences". Journal of Marketing Analytics. London, England: Palgrave Macmillan. c. 4 (2–3): 61–65.

10. What is Big data analytics Service Oriented Architecture [Електронний ресурс] - Режим доступу до ресурсу: https://contenteratechspace.com/blogs/what-is-big-data-analytics-service-oriented-architecture/#:~:text=Big%20data%20analytics%20as%20a,standardize%20everything%20across%20the%20enterprise.
Дата доступу - 13.05.2021

11. Watson Health Perspectives. 17 September 2016.

12. Crawford, Kate (21 September 2011). "Six Provocations for Big Data"

13. "Community cleverness required". Nature. 455 (7209): 1. September 2008.

14. Web service orchestration [Електронний ресурс] - Режим доступу до ресурсу: https://www.techtarget.com/searchapparchitecture/definition/web-service-orchestration
Дата доступу - 14.05.2021

15. SOA - Enterprise Service Bus [Електронний ресурс] - Режим доступу до ресурсу: https://www.tutorialspoint.com/soa/soa_enterprise_service_bus.htm
Дата доступу - 14.05.2021

16. Big Data – Buzz Words: What is MapReduce [Електронний ресурс] - Режим доступу до ресурсу: https://blog.sqlauthority.com/2013/10/09/big-data-buzz-words-what-is-mapreduce-day-7-of-21/
Дата доступу - 21.05.2021

17. Hirsch, Frederick; Kemp, John; Ilkka, Jani (2007-01-11). Mobile Web Services: Architecture and Implementation. John Wiley & Sons (published 2007). p. 27.

18. "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)". www.w3.org.

19. Ethereum Cryptocurrency [Електронний ресурс] - Режим доступу до ресурсу: https://console.cloud.google.com/marketplace/product/ethereum/crypto-ethereum-blockchain
Дата доступу - 29.05.2021

20. І.М. Пістунов, Антонюк О.П. ЕЛЕКТРОННА ЕКОНОМІКА Том. 1. Криптовалюта. Big Data. Дніпро: ДВНЗ «НГУ», 2017. 100 с.

21. What is MapReduce?: https://ua.talend.com/resources/what-is-mapreduce/
Дата доступу - 16.05.2021