Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук
(повна назва)
Кафедра Програмної інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Дослідження швидкодії, можливостей та вартості системи Data Lake в залежності від різних постачальників хмарних сервісів

(тема)

Виконав:
студент 2 курсу групи ІПЗм-20-4
Кузьменко О. В.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення

Тип програми Освітньо-наукова

Керівник проф. Смеляков. К.С.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ З.В. Дудар

2022 р.

Харківський національний університет радіоелектроніки

Факультет Комп’ютерних наук
Кафедра Програмної інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 - Інженерія програмного забезпечення
(код і повна назва)

Освітня програма Інженерія програмного забезпечення
(повна назва)

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

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Кузьменко Олексію Вікторовичу
(прізвище, ім’я, по батькові)

1. Тема роботи: «Дослідження швидкодії, можливостей та вартості системи Data Lake в залежності від різних постачальників хмарних сервісів»
затверджена наказом по університету від ___ _____2022 р. № _____
2. Термін подання студентом роботи до екзаменаційної комісії ___ _____2022 р.
3. Вихідні дані до роботи: *У програмного забезпеченні передбачено: авторизація, створення проекту, розгортання Озера Даних з використання обраного постачальника хмарних послуг, створення даних, пошук по метаданим.*
Використовувати: СКБД MongoDB, Ubuntu 20.04, мову python 3.10, Terraform, Kubernetes
4. Перелік питань, що потрібно опрацювати в роботі: *Вступ, аналітичний огляд, аналіз існуючих методів або алгоритмів , архітектура та проектування програмного забезпечення, опис прийнятих програмних рішень , висновки, перелік посилань.*

КАЛЕНДАРНИЙ ПЛАН

Номер	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналітичний огляд	5 березня 2022	виконано
2	Аналіз існуючих методів або алгоритмів	21 березня 2022	виконано
3	Розробка моделі	12 квітня 2022	виконано
4	Формування вимог ПЗ	15 квітня 2022	виконано
5	Проектування та розробка ПЗ	25 квітня 2022	виконано
6	Написання пояснювальної записка	1 травня 2022	виконано
7	Перевірка записка керівником та нормконтролером	5 травня 2022	виконано
8	Оцінка роботи стороннім рецензентом, отримання відгуку від керівника атестаційної роботи, попередній захист	7 травня 2022	виконано
9	Здача роботи у електронний архів, допуск роботи до захисту завідувачем кафедри та передача готової роботи секретарю ЕК	15 травня 2022	виконано
10	Здача готової роботи	25 травня 2022	виконано

Дата видачі завдання 17.01.2022 р.

Керівник проф. _____(Смеляков. К.С.)

Завдання прийняв до виконання_____ (Кузьменко О.В.)

РЕФЕРАТ / ABSTRACT

Звіт з практики містить : 54 с., 4 табл., 21 рис., 11 джер.

ОЗЕРО ДАНИХ, ВЕЛИКІ ДАНІ, МІКРОСЕРВІСИ, ХМАРНІ ОБЧИСЛЕННЯ, Python, Terraform, Kubernetes, AWS, GCP, AZURE.

Об’єктом аналізу та дослідження є система розгортання Озера Даних.

Метою роботи є створення системи автоматичного розгортання систем Озера Даних з використанням постачальника хмарних послуг AWS, AZURE та GCP.

Озеро Даних – це архітектура для зберігання даних будь-якого формату у необмежених кількостях, з можливістю пошуку цих даних по метаданим, а також подальшою їх обробкою.

У результаті роботи було створено таку систему з використанням кращих практик мікросервісної архітектури, використанням таких технологій, як Terraform, Kubernetes, Python, MongoDB, FastAPI, та зроблено висновки стосовно роботи цієї системи, та потенційного розвитку.

DATA LAKE, BIG DATA, MICROSERVICE, CLOUD CALCULATION, Python, Terraform, Kubernetes, AWS, GCP, AZURE.

The object of analysis and research is the Data Lake deployment system.

The aim of the work is to create a system of automatic deployment of Data Lake systems using the cloud service provider AWS, AZURE and GCP.

Data Lake is an architecture for storing data of any format in unlimited quantities, with the ability to search for this data on metadata, as well as their further processing.

As a result, the system was created using the best practices of microservice architecture, using technologies such as Terraform, Kubernetes, Python, MongoDB, FastAPI, and conclusions were drawn about the operation of this system and potential development.

Я, Кузьменко Олексій Вікторович, студент групи ІПЗм-20-4, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження швидкодії, можливостей та вартості системи Data Lake в залежності від різних

постачальників хмарних сервісів», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElAr KhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів

ЗМІСТ

ВСТУП

З розвитком систем штучного інтелекту, зокрема нейронних мереж, виникла потреба у великій кількості даних для навчання. Звичайні СУБД не підходять для таких завдань, оскільки для навчання часто необхідно зберігати сирі дані, щоб згодом над ними вже робити перетворення, досліджувати залежності та експериментувати. Стандартні СУБД сильно втрачають у швидкості пошуку та вставки даних при зростанні розміру БД. Тому була створена архітектура зберігання даних Озеро Даних [1][2], що не має таких недоліків. Оскільки для створення Озера Даних на власному сервері потрібні навички адміністрування систем, останнім часом все частіше вдаються до використання хмарних провайдерів.

Метою даної роботи є створення інформаційної системи з автоматичного розгортання Озера Даних на 3-х cloud провайдерах: GCP, AWS, Azure, та порівняння 4-х варіантів реалізації Озера Даних на цих провайдерах за такими параметрами за швидкістю додавання до них нової інформації.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Загальні поняття Озер Даних

Озеро Даних – це сховище, яке може зберігати велику кількість структурованих, напівструктурованих і неструктурованих даних. Це місце для зберігання всіх типів даних у рідному форматі без фіксованих обмежень на розмір облікового запису або файл. Він пропонує велику кількість даних для підвищення аналітичної продуктивності та інтеграції.

Озеро даних схоже на великий контейнер, який дуже схожий на справжні озеро та річки. Так само, як в озері є кілька приток, озеро даних має структуровані дані, неструктуровані дані, логи, що протікають в режимі реального часу.

Озеро даних є економічно ефективним способом зберігання всіх даних організації для подальшої обробки. Аналітик-дослідник може зосередитися на пошуку смислових моделей даних, а не самих даних.

На відміну від ієрархічного сховища даних, де дані зберігаються у файлах і папках, озеро даних має плоску архітектуру. Кожному елементу даних в озері даних надається унікальний ідентифікатор і позначається набір інформації метаданих.

Ось основні можливості та ключові відмінності Озера Даних:

- озеро даних забезпечує достатнє сховище даних для зберігання всіх даних підприємства чи організації;

- озеро даних може зберігати величезні обсяги даних усіх типів, включаючи структуровані, напівструктуровані та неструктуровані дані;

- дані, що зберігаються в озері даних, є необробленими даними або повною копією бізнес-даних. Дані зберігаються в озері даних, як і в бізнес-системі;

- озеро даних надає повні метадані для керування всіма типами елементів, пов’язаних із даними, включаючи джерела даних, формати даних, інформацію про з’єднання, схеми даних та можливості керування дозволами;

- озеро даних надає різноманітні аналітичні можливості, включаючи пакетну обробку, потокові обчислення, інтерактивну аналітику та машинне навчання, а також можливості планування роботи та керування;

- озеро даних підтримує комплексне управління життєвим циклом даних. На додаток до необроблених даних, озеро даних зберігає проміжні результати аналітики та обробки та веде повний запис про ці процеси. Це допоможе вам відстежити весь процес виробництва будь-якого запису даних;

- озеро даних надає комплексні можливості для пошуку та публікації даних. Озеро даних підтримує широкий спектр джерел даних. Він отримує повні та додаткові дані з джерел даних і зберігає отримані дані стандартним чином. Озеро даних передає результати аналізу та обробки даних у відповідні механізми зберігання даних, які підтримують доступ з різних програм;

- озеро даних надає можливості для великих даних, включаючи надвеликий простір для зберігання та масштабованість, необхідні для обробки даних у великому масштабі.

Оскільки озера даних є основою для аналітики та штучного інтелекту, підприємства в кожній галузі використовують їх для збільшення доходів, економії грошей та зниження ризиків.

Ось кілька прикладіва компаній, яким можне знадобитися Озеро Даних:

- компанія, яка пропонує потокову музику, радіо та подкасти, може збільшити дохід, покращивши свою систему рекомендацій, щоб користувачі більше споживали їхні послуги, дозволяючи компанії продавати більше реклами;

- багатонаціональна телекомунікаційна компанія може заощадити гроші, створюючи моделі схильності до відтоку, які зменшують відтік клієнтів;

- інвестиційна компанія може покладатися на озера даних для розвитку машинного навчання, тому вони можуть керувати портфельними ризиками, як тільки ринкові дані в режимі реального часу стануть доступними.

1.2 Рівні Озер Даних

Рівень необроблених даних – також називається шаром прийому/приземленням, оскільки він буквально є поглиначем нашого озера даних. Основна мета – зберегти дані в сирому вигляді якомога швидше та ефективно. Для цього дані повинні залишатися в рідному форматі. Ми не

допускаємо жодних перетворень на цьому етапі. За допомогою рівня необроблених даних ми можемо повернутися до певного моменту часу, оскільки архів підтримується. Не допускається перезаписування, що означає обробку дублікатів та різних версій одних і тих самих даних. Важливо зазначити, що кінцевим користувачам не слід надавати доступ до цього рівня. Дані тут не готові до використання, вони вимагають великих знань з точки зору відповідного та релевантного споживання.

Стандартизований рівень даних – може розглядатися як необов'язковий у більшості реалізацій. Якщо ми очікуємо, що наша архітектура озера даних буде швидко розвиватися, це правильний напрямок. Основна мета цього рівня – покращити продуктивність передачі даних з рівня необроблених даних рівня в рівні, де відбувається взаємодія з користувачами. Включаються як щоденні трансформації, так і обробка на вимогу. Якщо в рівні сирих даних дані зберігаються у рідному форматі, у Стандартизованому ми вибираємо формат, який найкраще підходить для очищення. Конструкція така ж, як і в попередньому шарі, але при необхідності її можна розділити, щоб зменшити зернистість.

Очищений рівень даних – також називається Curated Layer/Conformed Layer. Дані перетворюються в споживані набори даних і можуть зберігатися у файлах або таблицях. Призначення даних, а також їх структура на цьому етапі вже відомі. Перед цим шаром варто очікувати очищення та перетворення. Також поширеною є денормалізація та консолідація різних об'єктів. Завдяки всьому вищесказаному, це найскладніша частина всього рішення Озера Даних. Що стосується організації ваших даних, то структура досить проста і зрозуміла. Наприклад: Мета/Тип/Файли. Зазвичай кінцевим користувачам надається доступ лише до цього рівня.

Рівень даних програми – також званий довіреним рівнем/безпечним рівнем/виробничим рівнем, отриманий із Очищеного рівня та доповнений будь-якою необхідною бізнес-логікою. Це можуть бути сурогатні ключі, спільні між програмою, безпека на рівні рядків або щось інше, що характерно для програми, яка використовує цей рівень. Якщо будь-яка з ваших програм використовує

моделі машинного навчання, які розраховуються на вашому Озері Даних, ви також отримаєте їх звідси. Структура даних залишиться тією ж, що й у Очищеному рівні.

Рівень даних пісочниці – ще один рівень, який можна вважати необов’язковим, призначений для роботи досвідчених аналітиків і науковців з даних. Тут вони можуть проводити свої експерименти, шукаючи закономірності чи кореляції. Якщо у вас є ідея збагатити свої дані будь-яким джерелом з Інтернету, Рівень пісочниці – відповідне місце для цього.

2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ АБО АЛГОРИТМІВ

2.1 Аналіз існуючих рішень

Вже існують деякі рішення, що допомагають розгорнути Озеро Даних на хмарних сервісах AWS, але всі мають свої недоліки.

Одним з таких рішень є платформа Qubole, яка дозволяє створити Озеро Даних на декількох платформах, включаючи AWS.

Qubole — це відкрита платформа Озера Даних для машинного навчання, потокової передачі та спеціальної аналітики. Платформа надає комплексні послуги, які зменшують час, зусилля та витрати, необхідні для запуску конвеєрів даних, потокової аналітики та виконання робочих навантажень машинного навчання в будь-якій хмарі.

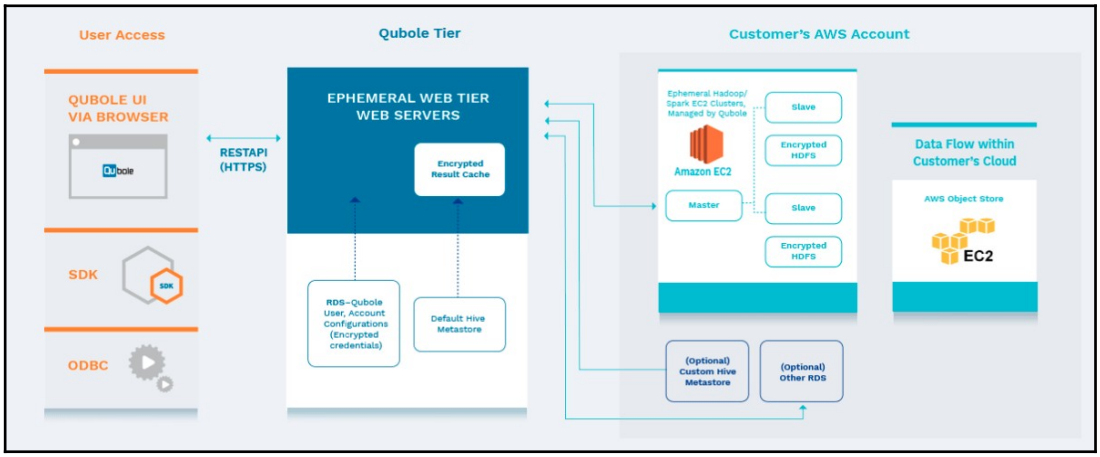

Рисунок 2.1 – Архітектура Озера Даних від Qubole

Недоліком цієї платформи є відсутність будь-яких рекомендацій щодо збільшення або зменшення використання обчислювальних ресурсів.

Ще одним варіантом є Озеро Даних від Azure. Озеро Даних від Azure включає всі можливості, необхідні для того, щоб розробникам, науковцям з даних і аналітиків було легко зберігати дані будь-якого розміру, форми та швидкості, а також виконувати всі типи обробки й аналітики на різних платформах і мовах. Це позбавляє від складнощів отримання та зберігання всіх ваших даних, а також пришвидшує роботу з пакетною, потоковою та інтерактивною аналітикою. Озеро

Даних від Azure працює з наявними інвестиціями в IT для ідентифікації, керування та безпеки для спрощеного керування даними та керування ними. Він також легко інтегрується з оперативними сховищами та сховищами даних, щоб ви могли розширювати поточні програми даних. Озеро Даних від Azure вирішує багато проблем продуктивності та масштабованості, які заважають вам максимізувати цінність ваших активів даних за допомогою служби, яка готова задовольнити ваші поточні та майбутні потреби бізнесу.

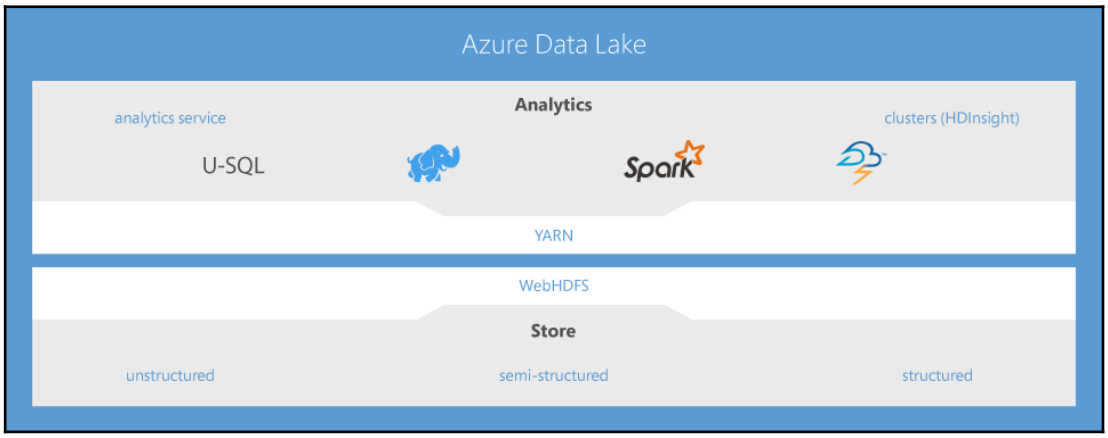

Рисунок 2.2 – Архітектура Озера Даних від Azure

Недоліком є висока вартість через використання багатьох сервісів.

Іншим рішенням є CloudFormation [3] маніфести від самого постачальника хмарних послуг AWS. Озеро даних на базі AWS автоматично налаштовує основні служби AWS, необхідні для легкого позначення, пошуку, спільного використання, трансформації, аналізу й керування певними підмножинами даних у компанії чи з іншими зовнішніми користувачами. Керівництво розгортає консоль, до якої користувачі можуть отримати доступ для пошуку та перегляду доступних наборів даних для своїх бізнес-потреб. Він також містить об’єднаний шаблон, який дозволяє запустити версію рішення, готову до інтеграції з Microsoft Active Directory.

Це рішення являється достатньо дорогим через використання великої кількості сервісів, а також потребує додаткових знань платформи AWS, щоб самому розгорнути Озеро Даних за допомогою CloudFormation маніфестів.

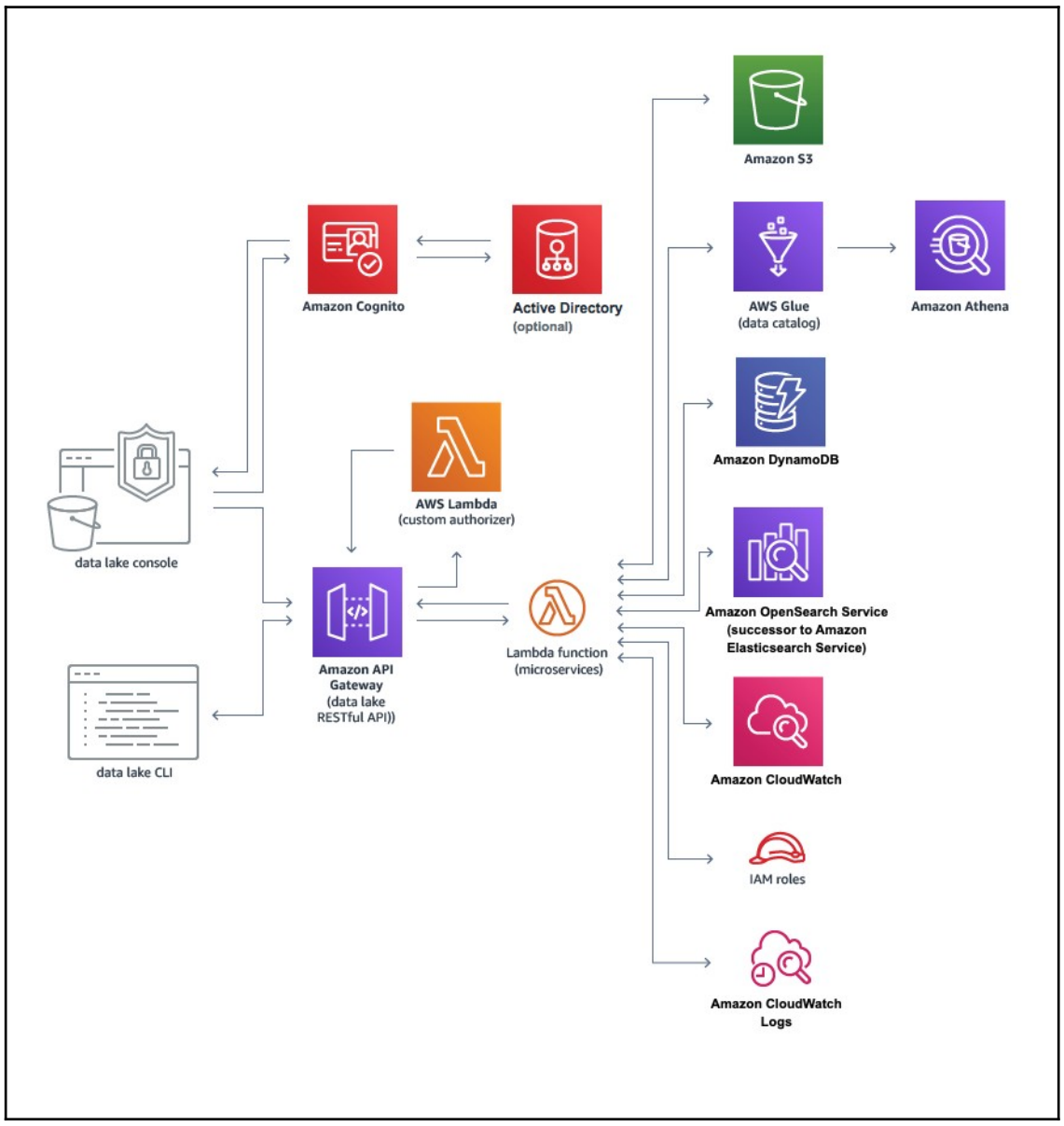

Рисунок 2.3 – Архітектура Озера Даних від AWS

Внаслідок аналізу стає зрозуміло, що хоч вже і є певні рішення в цій області, у тому числі від самої платформи AWS, та вони дуже не гнучкі, і не дають можливості адаптувати використання ресурсів під конкретний проект з потоком даних.

3 ПОСТАНОВКА ЗАДАЧІ

3.1 Опис вимог до рішення

Для початку необхідно визначити можливості, які повинна надавати система розгортання Озер Даних. Ці можливості можна розділити на декілька категорій:

- надавати можливість розгортати користувачами варіанти Озер Даних з доступного списку;
- перевіряти правильність усіх введених параметрів;
- збирати усі логи для подальшого аналізу;
- надавати однаковий інтерфейс для зберігання та пошуку даних, незалежно від обраного типу розгортання та хмарного постачальника.

Далі детальніше про ці пункти.

3.1.1 Надавати можливість розгортати користувачами варіанти Озер Даних з доступного списку

Для коректної роботи системи, необхідно розробити конкретні варіанти розгортання системи Озер Даних, що дозволять зберігати необмежену кількість даних, та виконувати пошук по метаданим. Для більш коректного порівняння, буде використовуватися однакова архітектура з використанням сервісів трьох постачальниках хмарних послуг: AWS, GCP та Azure. Під однаковою архітектурою мається на увазі використання двох сервісів, які будуть реалізувати зберігання даних та метаданих в Озері Даних, а саме NoSQL база даних для зберігання і пошуку даних за метаданими, та хмарне сховище даних.

У рамках цієї роботи було розроблено наступні варіанти розгортувань:

а Розгортання на основі GCP з використання BigTable та Cloud Storage. Дані будуть зберігатися у Cloud Storage, у той час як метадані, по яким буде робитися пошук, будуть знаходитися у BigTable. Для зручності будемо називати цей тип розгортання GCP_2.

б Розгортання на основі AWS з використанням S3 та OpenSearch. Дані будуть зберігатися у S3, у той час як метадані, по яким буде робитися пошук,

будуть знаходитися у OpenSearch. Для зручності будемо називати цей тип розгортання AWS_1.

в Розгортання на основі AZURE з використання Blob Storage та CosmosDB. Дані будуть зберігатися у Blob Storage, у той час як метадані, по яким буде робитися пошук, будуть знаходитися у CosmosDB. Для зручності будемо називати цей тип розгортання AZURE_1.

3.1.2 Перевіряти правильність усіх введених параметрів

Так як робота сервісу складається з взаємодії з різними постачальниками хмарних послуг, то для цієї взаємодії користувачам потрібно надати ключі та паролі, які будуть надавати необхідні права у конкретного постачальника хмарних послуг.

Після перевірки самих ключів та паролів, необхідно також перевірити параметри розгортання Озера Даних у цього постачальника, щоб коректно створити цей варіант розгортання.

3.1.3 Збирати усі логи для подальшого аналізу

У процесі розгортання та взаємодії з Озером Даних багато що може піти не так, тому вкрай необхідно збирати усі логи, та надавати їх користувачу та розробникам сервісу, щоб з’ясувати причину проблеми та виправити її.

3.1.4 Надавати однаковий інтерфейс для зберігання та пошуку даних, незалежно від обраного типу розгортання та хмарного постачальника

Важливою частиною розроблюваної системи є однаковий інтерфейс, незалежно від обраного типу розгортання.

Цей пункт дозволяє нам розбити взаємодію з Озером Даних на етапи, незалежно від типу розгортання, і проводити оцінювання швидкодії.

Можна виділити 3 етапи роботи з Озером Даних:

- Створити запис про майбутні дані, та додати користувацькі метадані, які потім будуть використовуватися для пошуку.

- Завантажити дані у сирому вигляді.
- Пошук по створеним даним і записам.

Розбиття на перший та другий пункт є критично важливим, так як це дозволяє розділити дані, що додає користувач, та оригінальний формат даних, який буде зберігатися.

4 ФОРМУВАННЯ ВИМОГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Серверна частина

Уся система складається з трьох мікросервісів, що в організують повноцінну систему для розгортання Озер Даних та взаємодії з ними. В кожного з сервісів є своя база даних, щоб зробити його максимально автономним та надати можливість для масштабування.

Діаграму з сервісами та їх взаємодією можна побачити на рисунку 4.1.

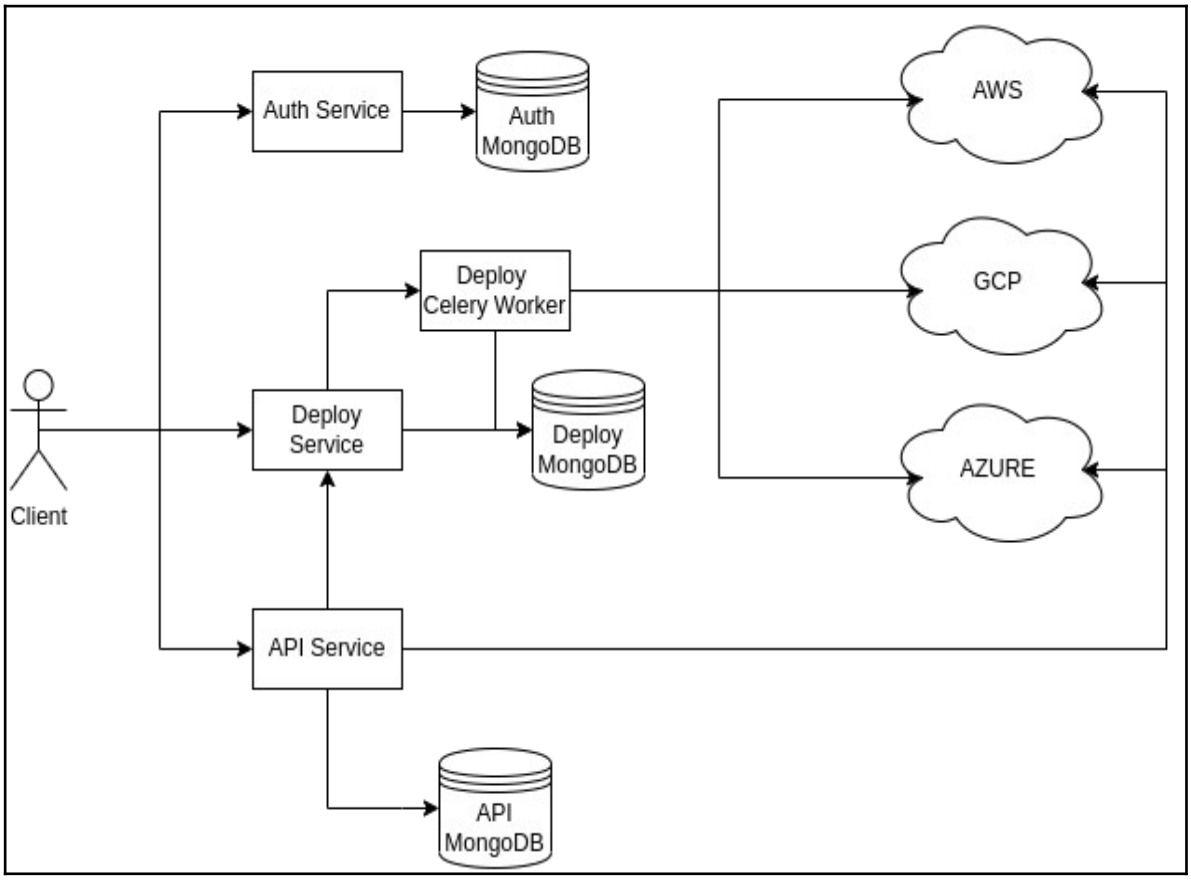

Рисунок 4.1 – Діаграма взаємодії сервісів

Кожен з сервісів надає API-інтерфейс RESTful, завдяки якому користувач і може взаємодіяти із системою.

Для розгортання як самої системи, так і користувацьких Озер Даних, використовуються технології Terraform [4] та Kubernetes [5]. Ці технології дозволяють тримати всю систему за принципом Інфраструктура як код, а також дасть можливість простого масштабування, і оновлення версій застосунку.

5 ПРОЕКТУВАННЯ ТА АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

5.1 Опис Auth сервісу

Auth сервіс призначений для реєстрації та авторизації користувачів. Сервіс написаний мовою Python [6] з використанням бази даних mongodb [7]. Для цих цілей була обрана бібліотека FastAPI [8], що використовує бібліотеку pydantic, та допоміжна бібліотека fastapi-users. Комбінація цих бібліотек дозволяє використовувати і валідувати дані у зручний для мови Python спосіб. Ті ж моделі, що використовуються для валідації даних в API можна використовувати для зберігання даних в базі даних.

Прикладом таких моделей може бути наступний код моделей користувача:

```
class BaseUser(CreateUpdateDictModel):
    """Base User model."""

    id: UUID4 = Field(default_factory=uuid.uuid4)
    email: EmailStr
    is_active: bool = True
    is_superuser: bool = False
    is_verified: bool = False

class BaseUserCreate(CreateUpdateDictModel):
    email: EmailStr
    password: str
    is_active: Optional[bool] = True
    is_superuser: Optional[bool] = False
    is_verified: Optional[bool] = False

class BaseUserUpdate(CreateUpdateDictModel):
    password: Optional[str]
    email: Optional[EmailStr]
    is_active: Optional[bool]
    is_superuser: Optional[bool]
    is_verified: Optional[bool]

class BaseUserDB(BaseUser):
    hashed_password: str

    class Config:
        orm_mode = True
```

З цього прикладу можна побачити, що написана один раз модель, може бути використана для інших моделей API, а також моделі бази даних. Це зменшує кількість коду, що повинен бути написаний, та зменшує кількість потенційних помилок.

У результаті отримуємо сервіс з кінцевими точкам, що наведені на рисунку 5.1. Їх список можна переглянути з використанням технології Swagger, що одночасно дозволяє провзаємодіяти з цими кінцевими точками, що у свою чергу прискорить розробку системи.

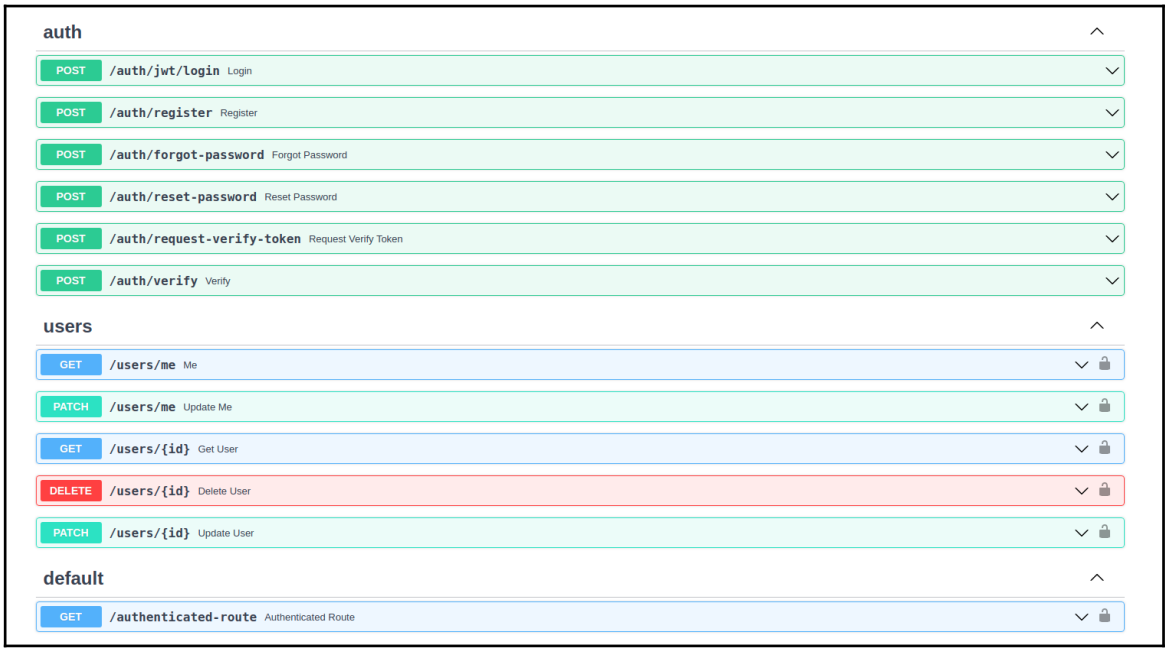

Рисунок 5.1 – Кінцеві точки сервісу авторизації і реєстрації

Основну функціональність дають нам 2 кінцеві точки: /auth/register та /auth/jwt/login. Вони дозволяють зареєструватися та авторизуватися відповідно. Після проведення цих операцій, ми отримаємо JWT токен, який необхідно буде використовувати у двох інших сервісах.

5.2 Опис Deploy сервісу

Сервіс Deploy складається з двох частин: серверної частини, через яку користувач може взаємодіяти з системою, та частини, що відповідає за асинхронне розгортування Озера Даних. Обидві частини написані мовою Python з

використанням бази даних mongodb. Серверна частина використовує бібліотеку FastAPI для взаємодії з користувачем. Частина, що виконує розгортування, слугує обгорткою над технологією Terraform, яка дозволяє розгортати різноманітні рішення на серверах постачальників хмарних послуг. Для роботи цього сервісу було розроблено 3 моделі даних: Project, ProjectCredentials та ProjectDeploy. Нижче наведений програмний код цих моделей.

```
     class ServiceProviderType(str, Enum):
        AWS = "AWS"
        AZURE = "AZURE"
        GCP = "GCP"


     class ProjectMixin(BaseModel):
        verified: typing.Optional[bool] = False
        name: str
        service_provider: constr(
            regex=rf"^({'|'.join([e.value for e in ServiceProviderType])})
$"  # noqa
        )

        def create_update_dict(self):
            return self.dict(
                exclude_unset=True,
                exclude={
                    "id",
                    "verified",
                },
            )

        def create_update_dict_superuser(self):
            return self.dict(exclude_unset=True, exclude={"id"})


     class GCPCredentials(BaseModel):
        type: constr(max_length=256)
```

```
        project_id: constr(max_length=256)
        private_key_id: constr(min_length=40, max_length=40)
        private_key: constr(max_length=2096)
        client_email: constr(max_length=256)
        client_id: constr(min_length=21, max_length=21, regex=r"^\d+$")  #
noqa
        auth_uri: stricturl(max_length=256)
        token_uri: stricturl(max_length=256)
        auth_provider_x509_cert_url: stricturl(max_length=256)
        client_x509_cert_url: stricturl(max_length=256)


    class AWSCredentials(BaseModel):
        access_key_id: constr(max_length=256)
        secret_access_key: constr(max_length=256)


    class AzureCredentials(BaseModel):
        tenant_id: constr(max_length=256)
        client_id: constr(max_length=100)
        client_secret: constr(max_length=100)


    class ProjectCredentialsMixin(BaseModel):
        project: typing.Optional[UUID4]
            credentials:  typing.Union[GCPCredentials,  AWSCredentials,
AzureCredentials]

        def create_update_dict(self):
            return self.dict(
                exclude_unset=True,
                exclude={
                    "id",
                    "project",
                },
            )

        def create_update_dict_superuser(self):
```

```
            return self.dict(exclude_unset=True, exclude={"id"})


    class AWSProjectDeployType(str, Enum):
        AWS_1 = "AWS_1"


    class GCPProjectDeployType(str, Enum):
        GCP_1 = "GCP_1"
        GCP_2 = "GCP_2"


    class AzureProjectDeployType(str, Enum):
        AZURE_1 = "AZURE_1"


    class ProjectDeployMixin(BaseModel):
        project: typing.Optional[UUID4]
        deploy_type: constr(
            regex=rf"^({'|'.join([e.value for e in (*AWSProjectDeployType,
*GCPProjectDeployType, *AzureProjectDeployType)])})$"  # noqa
        )
        project_structure: typing.Optional[dict]

        def create_update_dict(self):
            return self.dict(
                exclude_unset=True,
                exclude={
                    "id",
                    "project",
                },
            )

        def create_update_dict_superuser(self):
            return self.dict(exclude_unset=True, exclude={"id"})
```

Наступні кінцеві точки доступні користувачеві для взаємодії з даним сервісом:

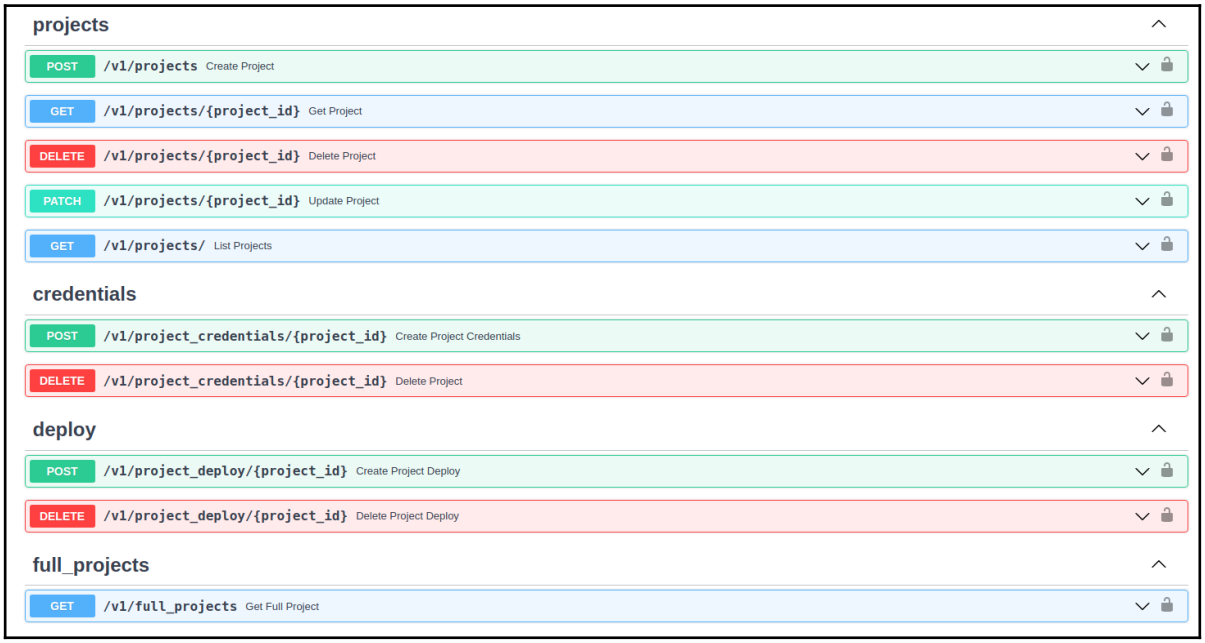

Рисунок 5.2 – Кінцеві точки сервісу розгортання

Користувач створює проект для певного постачальника хмарних послуг, додає необхідні ключі та паролі для взаємодії з ним, та потім обирає можливу реалізацію Озера Даних для обраного постачальника.

Після цього, за допомогою черги RabbitMQ [9] та бібліотеки Celery [10], відбувається взаємодія з частиною розгортання, там результати виконання розгортання будуть занесені в базу даних.

5.3 Опис API сервісу

Сервіс призначений для надання однакового інтерфейсу взаємодії з Озером Даних, незалежно від конкретної реалізації. Сервіс написаний мовою Python з використанням бази даних mongodb. Для цих цілей була обрана бібліотека FastAPI, що використовує бібліотеку pydantic.

Хоча в цьому сервісі немає конкретної взаємодії з базою даних, але в кінцевих точках також використовується певний формат даних. Через це було також створено наступні моделі, аби стандартизувати взаємодію з сервісом.

```
class Tag(BaseModel):
    name: str
    value: typing.Optional[str]


class BlobCreate(BaseModel):
    name: str
    content_type: typing.Optional[str] = "application/json"
    timestamp: typing.Optional[str] = Field(
        default_factory=lambda: datetime.datetime.now().isoformat()
    )
    source: typing.Optional[str] = ""
    user_tags: typing.Optional[list[Tag]] = []
    system_tags: typing.Optional[list[Tag]] = []
    size: typing.Optional[int] = 0  # in bytes


class Blob(BlobCreate):
    name: typing.Optional[str] = ""
    blob_id: str  # UUID?


class BlobDataJson(BaseModel):
    data: typing.Optional[dict] = None


class BlobDataBinary(BaseModel):
    data: typing.Optional[dict] = None
```

Було реалізовано 3 кінцеві точки, що дозволяють створювати об’єкти в Озері Даних, та шукати їх за метаданими.

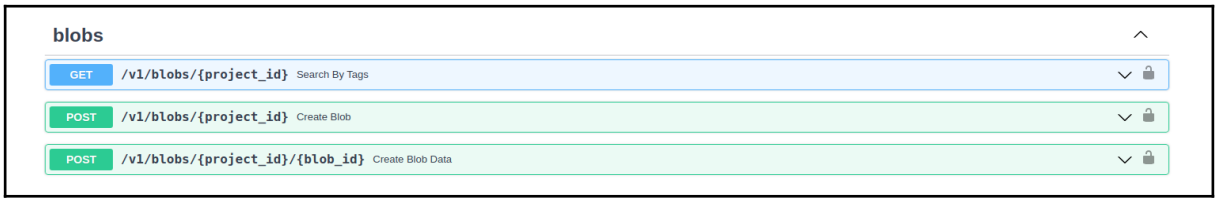

Рисунок 5.3 – Кінцеві точки сервісу API

Додавання даних відбувається в 2 етапи, щоб дозволити створення записів будь-якого формату. На першому етапі створюється запис у строгому форматі, і користувачеві видається ідентифікатор майбутніх даних. На другому етапі користувач може надіслати дані будь-якого формату, що вже будуть аналізуватися на предмет метаданих, там зберігатися у одному із обраних варіантів Озер Даних.

Через обмеження деяких варіантів розгортування, на першому етапі для тимчасового зберігання користувацьких даних використовується база даних MongoDB. Це дозволяє на другому етапі надіслати дані відразу в NoSQL та сховище даних, без оновлень.

6 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ ТА АНАЛІЗ ВАРІАНТІВ РОЗГОРТУВАННЯ

6.1 Проведення експериментів

Було проведено експерименти з замірами швидкодії системи та різних розгортувань Озер Даних на етапі створення даних, а також на етапі пошуку даних за метаданими. Для більш об’єктивної оцінки, були використані 2 варіанта даних: JSON об’єкти з інформацією з IoT датчиків, та зображення з датасету imagenet. Обидві колекції даних було взято з сервісу Kaggle. Кожен з тестів проводився від тисячі разів, і як результат проведення тесту береться середній час виконання.

6.1.1 Експерименти з оцінкою швидкості створення даних

У цьому експерименті оцінюється швидкість зберігання усіх даних на сервісах відповідного провайдера, та як результат середній час відповіді сервісу на запит створення даних.

6.1.1.1 Створення даних на базі розгортування AWS

Середній час створення даних на базі розгортування AWS з форматом JSON дорівнює 1.3 секунди, тоді як середній час створення зображення - 1.4 секунди. Результати вимірювань у вигляді графіків. Зростання часу зумовлюється обробкою цих зображень для отримання додаткових метаданих на стороні самого сервісу. Результати вимірювань можна побачити на рисунку 6.1.

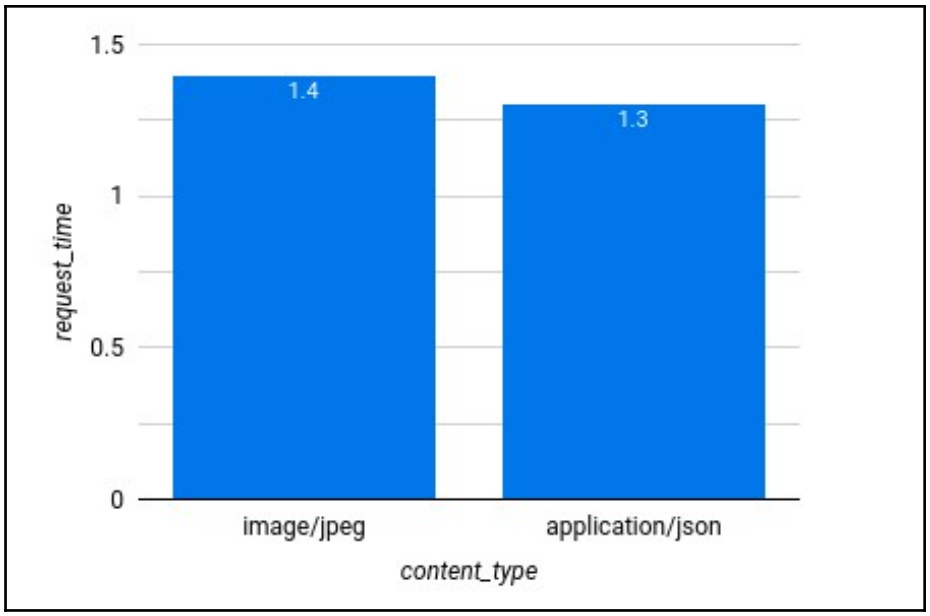

Рисунок 6.1 – Середній час збереження даних на базі розгортування AWS в залежності від типу даних

Зі збільшенням розміру зображень не виявлено різких змін у швидкості збереження даних. Результати вимірювань наведені на рисунку 6.2.

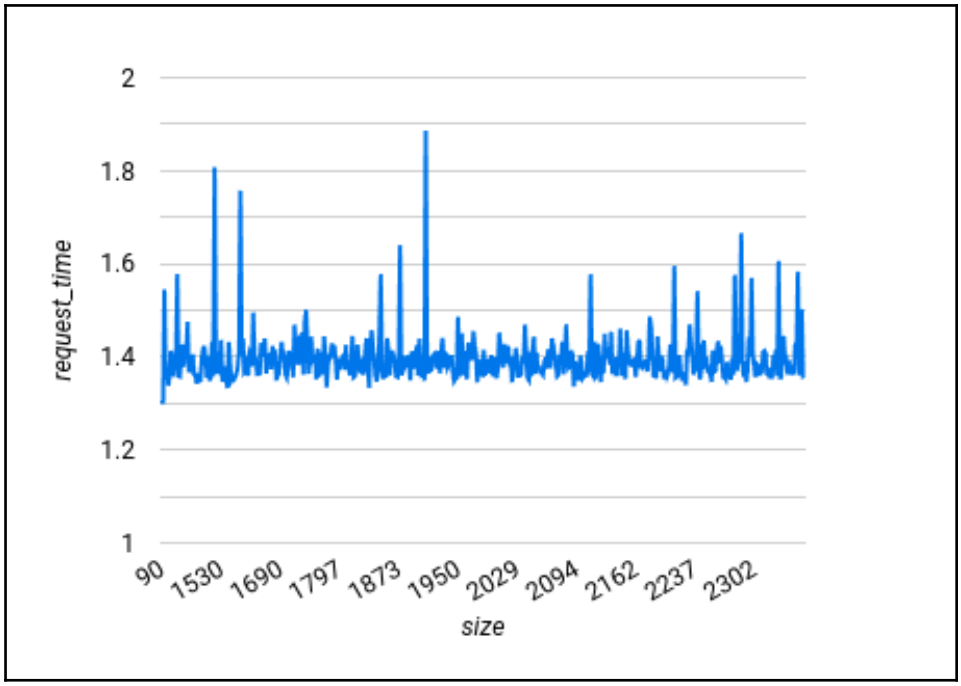

Рисунок 6.2 – Середній час збереження даних на базі розгортування AWS в залежності від розміру даних

6.1.1.2 Створення даних на базі розгортування GCP

Середній час створення даних на базі розгортування GCP з форматом JSON дорівнює 1.37 секунди, тоді як середній час створення зображення - 1.41 секунди. Результати вимірювань можна побачити на рисунку 6.3. Зростання часу зумовлюється обробкою цих зображень для отримання додаткових метаданих на стороні самого сервісу.

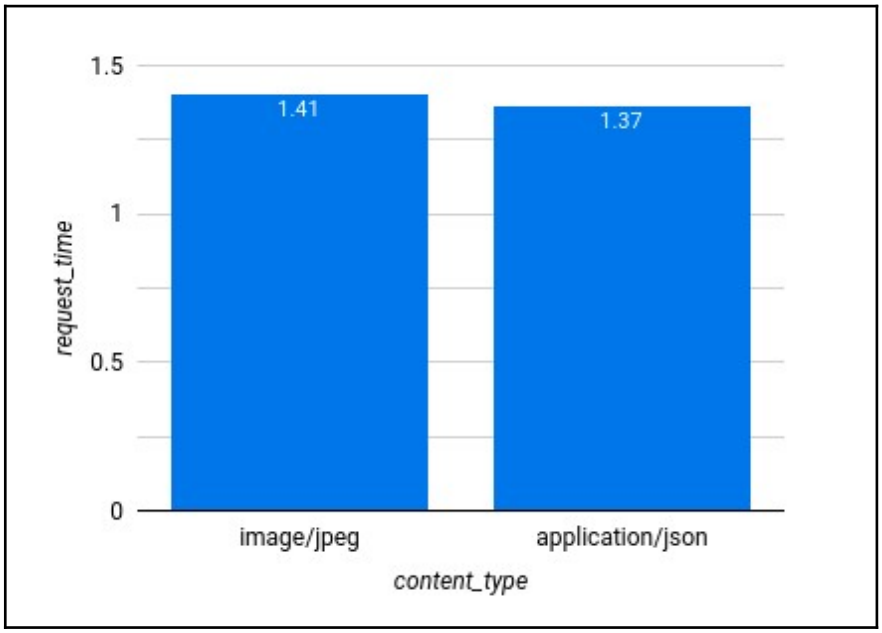

Рисунок 6.3 – Середній час збереження даних на базі розгортування GCP в залежності від типу даних

Зі збільшенням розміру зображень не виявлено різких змін у швидкості збереження даних. Результати вимірювань, створених за допомогою Data Studio [11], наведені на рисунку 6.4.

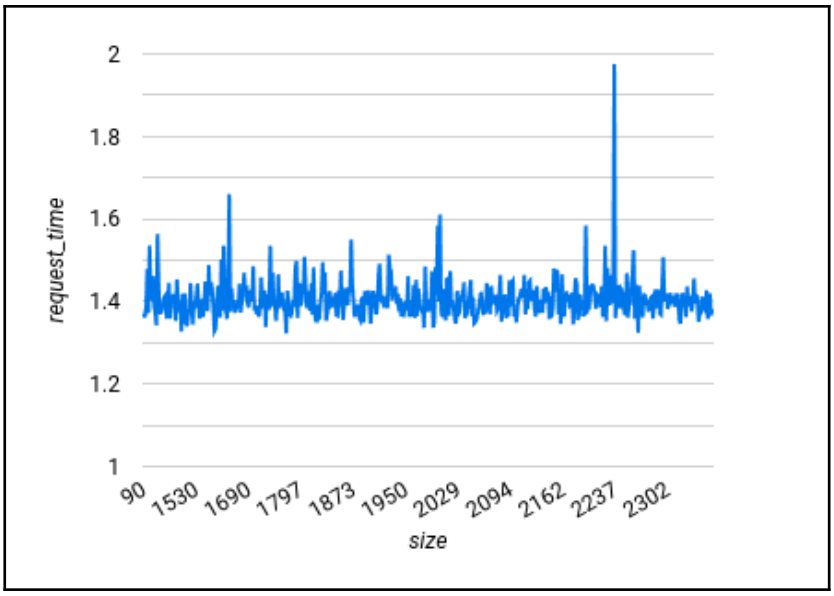

Рисунок 6.4 – Середній час збереження даних на базі розгортування GCP в залежності від розміру даних

6.1.1.3 Створення даних на базі розгортування Azure

Середній час створення даних на базі розгортування Azure з форматом JSON дорівнює 2.65 секунди, тоді як середній час створення зображення - 2.75 секунди. Результати вимірювань можна побачити на рисунку 6.5. Зростання часу зумовлюється обробкою цих зображень для отримання додаткових метаданих на стороні самого сервісу.

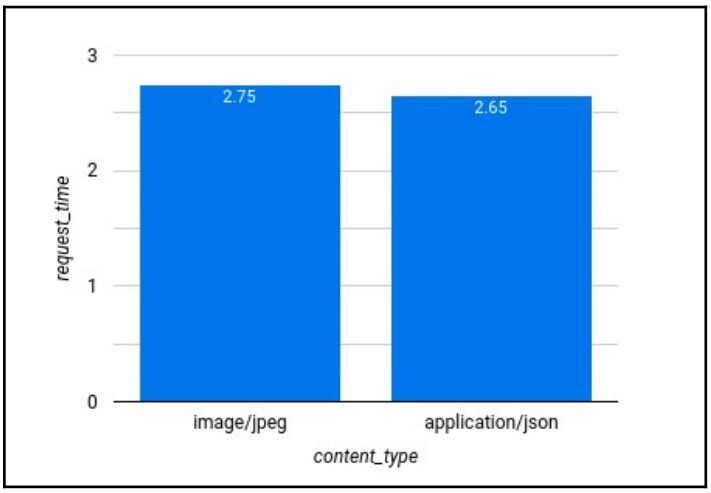

Рисунок 6.5 – Середній час збереження даних на базі розгортування Azure в залежності від типу даних

Зі збільшенням розміру зображень не виявлено різких змін у швидкості збереження даних. Результати вимірювань наведені на рисунку 6.4.

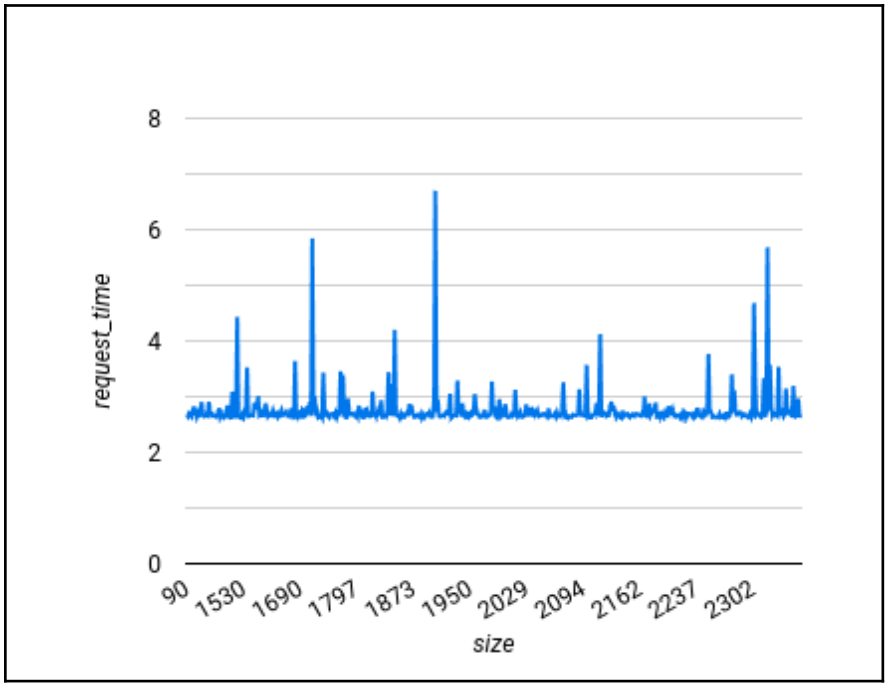

Рисунок 6.4 – Середній час збереження даних на базі розгортування Azure в залежності від розміру даних

6.1.1.4 Порівняння результатів замірів часу для всіх варіантів розгортування

Після проведення експериментів по додаванню даних до Озер Даних на базі різних хмарних провайдерів, можна сказати, що результати часу створення даних майже не відрізняються для AWS та GCP. У той же час, створення даних у Озері Даних на базі Azure займає вдвічі більше часу, що може бути критично при роботі з великою кількістю та частотою створення даних. Візуально це можна побачити на рисунку 6.5.

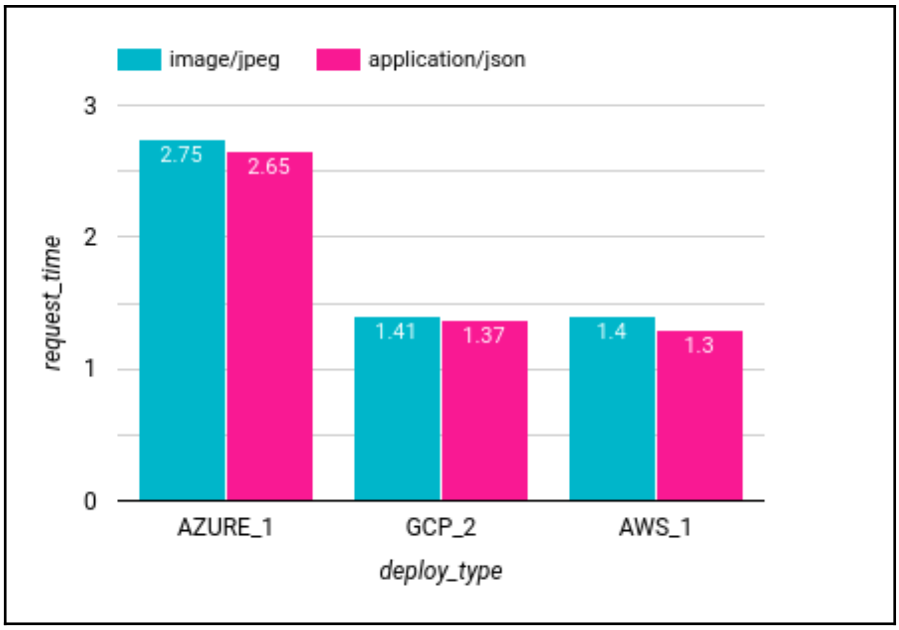

Рисунок 6.5 – Середній час збереження даних в залежності від типу даних та варіанту розгортування

Для всіх варіантів розгортання суттєвої різниці в часу створенні даних при збільшенні розміру даних не виявлено. Візуально це можна побачити нижче, на рисунку 6.6.

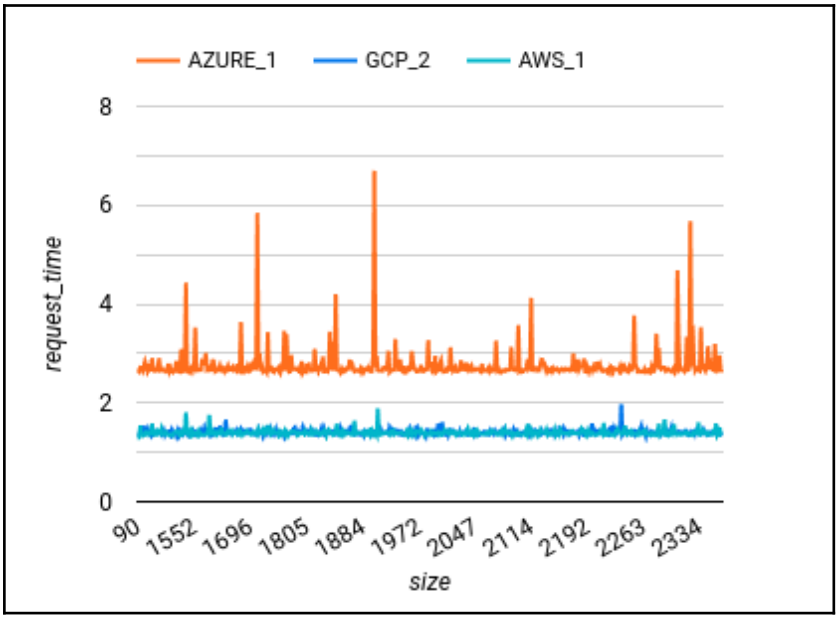

Рисунок 6.6 – Середній час збереження даних в залежності від розміру даних та варіанту розгортування

6.1.2 Експерименти з оцінкою швидкості пошуку даних
6.1.2.1 Пошук даних на базі розгортування AWS

Неможливо однозначно визначити, як точно впливає кожен з параметрів на швидкість пошуку, але можна сказати, що зі збільшенням кількості результатів росте й час роботи операції пошуку. Так, найшвидшими запитами є запити з одним результатом і одним параметром пошуку, а також з трьома результатами і двома параметрами пошуку, які виконуються у середньому за 560 мілісекунд. У той же час, найдовше виконувався запит, що повертав 982 результати, і фільтрувався тільки по одному параметру: він виконувався за 4 секунди та 250 мілісекунд. Вплив кількості параметрів дослідити важко, бо в різних випадках при збільшенні кількості параметрів швидкість роботи як зростає, так і спадає, у кожному конкретному запиті. Результати таких запитів можна побачити нижче, на рисунку 6.7.

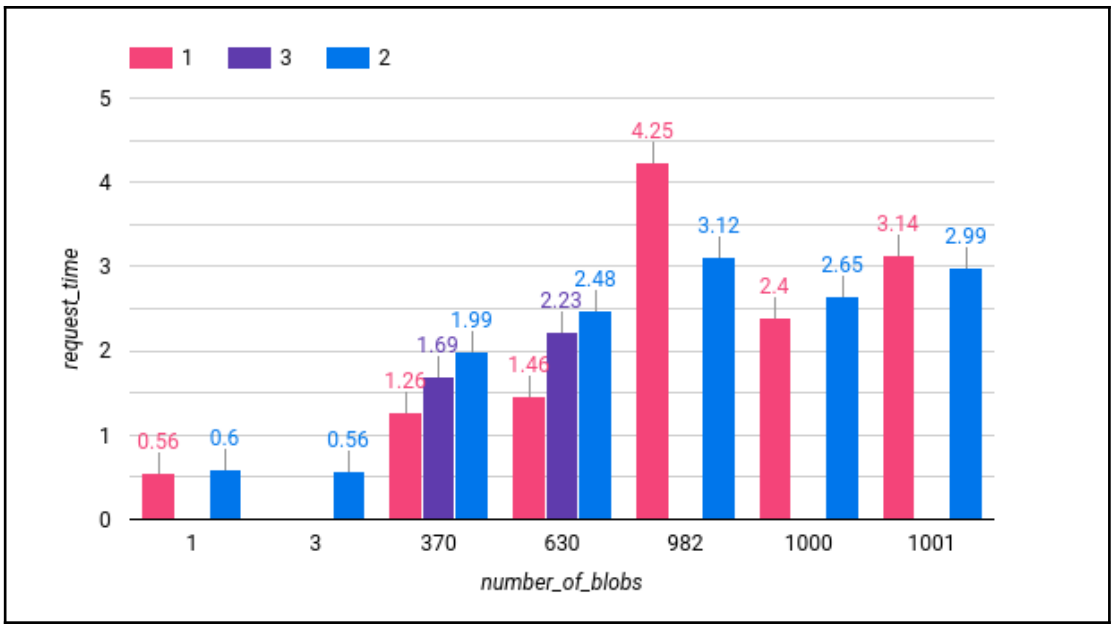

Рисунок 6.7 – Середній час пошуку даних в залежності від кількості результатів та кількості параметрів для пошуку на базі розгортування AWS

Хоча все теж не дуже однозначно, але загалом ясно, що при збільшенні кількості результатів, середній час роботи пошуку зростає, що і можна побачити на рисунку 6.8.

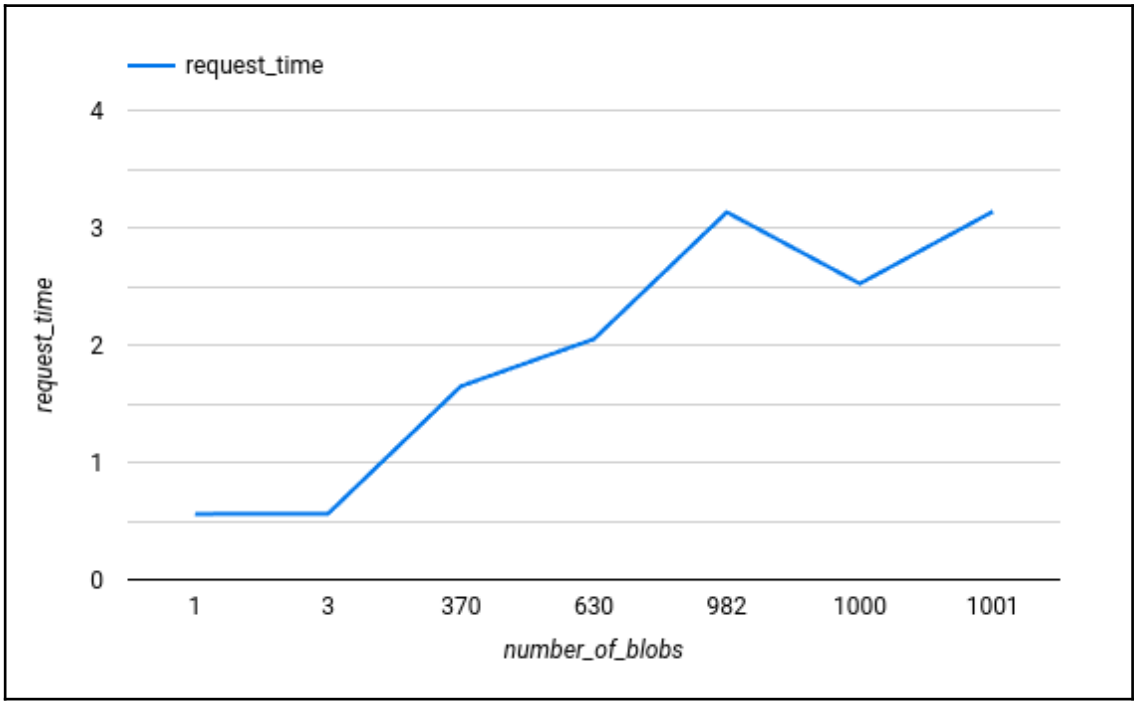

Рисунок 6.8 – Середній час пошуку даних в залежності від кількості результатів на базі розгортування AWS

6.1.2.2 Пошук даних на базі розгортування GCP

Пошук на базі GCP має дуже великий недолік, що унеможливлює одночасний пошук по декільком параметрам з одного типу метаданих (користувацьких або системних), тому кількість варіантів тестуваннння для цієї системи також обмежена. З одного експерименту видно, що кількість параметрів для фільтрування впливає не суттєво на час виконання запиту, але він не є показовим, тому не будемо розглядати вплив кількості параметрів на швидкість пошуку.

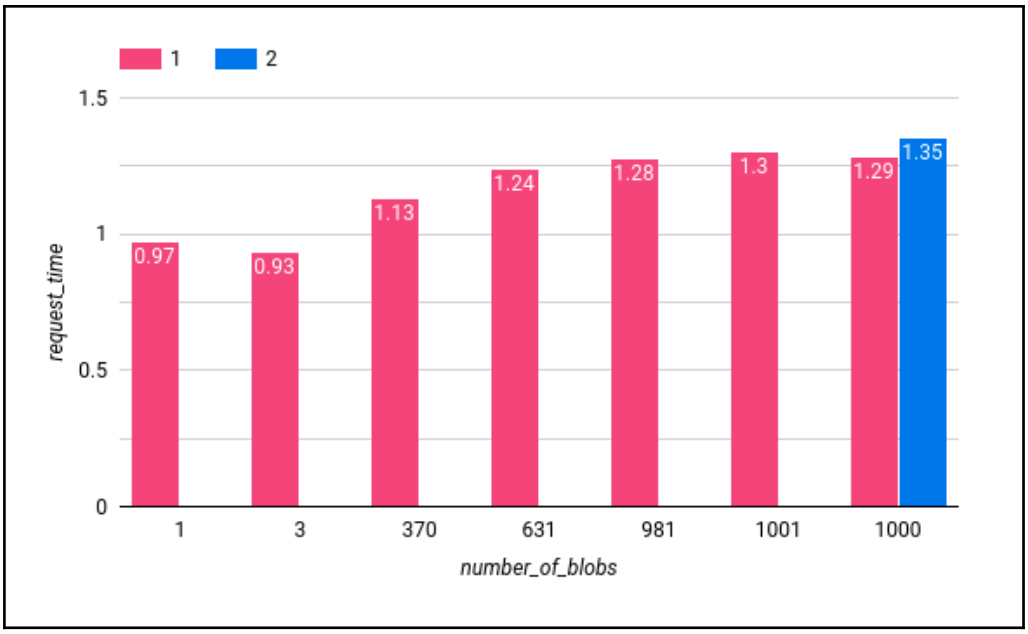

Рисунок 6.9 – Середній час пошуку даних в залежності від кількості результатів та кількості параметрів для пошуку на базі розгортування GCP

В той же час, можемо побачити, що кількість результатів впливає на швидкість, але не дуже суттєво. Для результату з одним об’єктом пошук виконується за 970 мілісекунд, в той час як для результату пошуку з 1000 об’єктами пошук займає 1 секунду 300 мілісекунд.

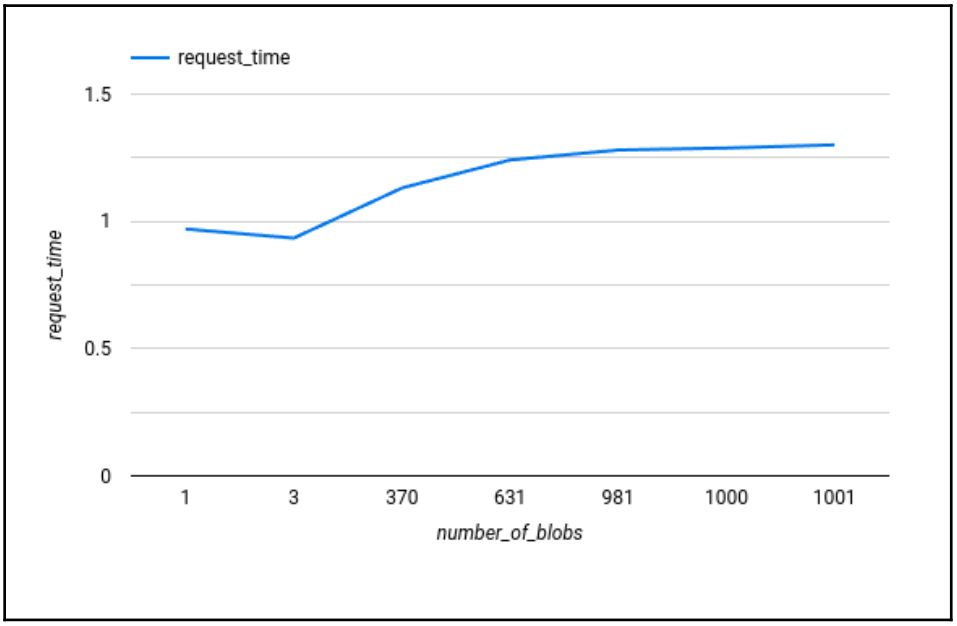

Рисунок 6.10 – Середній час пошуку даних в залежності від кількості результатів на базі розгортування GCP

6.1.2.3 Пошук даних на базі розгортування Azure

З експериментів на базі Azure стає зрозуміло, що основним критерієм у цьому випадку є саме кількість результатів. Для тестів з однаковою кількістю результатів, кількість параметрів пошуку ніяк не впливала, або впливала не суттєво. Це можна добре побачити на рисунку 6.11.

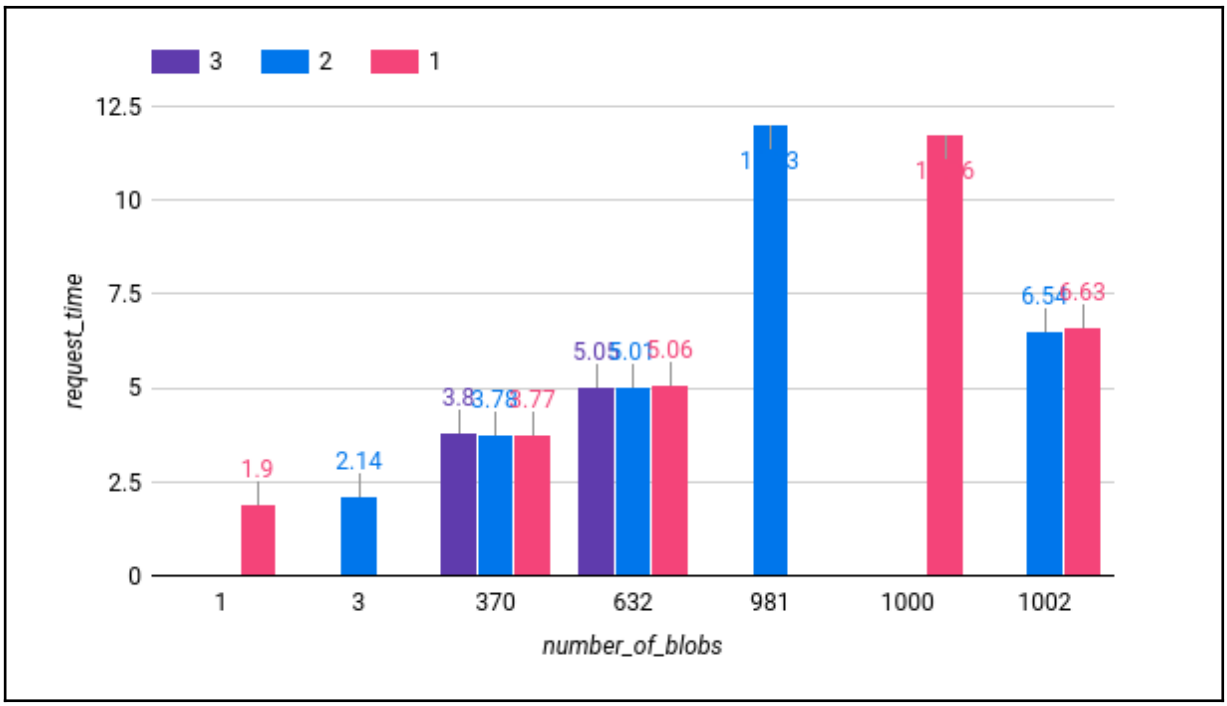

Рисунок 6.11 – Середній час пошуку даних в залежності від кількості результатів та кількості параметрів для пошуку на базі розгортування Azure

Кількість же результатів впливала дуже суттєво на швидкодію роботи Озера Даних на базі Azure, збільшуючи швидкість роботи зі збільшенням кількості результуючих даних, з дивним викидом на відмітці 1002 об’єкта. Скоріше за все цей викид зумовлений поверненням усіх даних з внутрішніх індексів, що зменшило потребу в фільтрації, і пришвидшило пошук. В інших же випадках, швидкість збільшувалася пропорційно, починаючи з 1 секунди 900 мілісекунд при результаті в один об’єкт і більше одинадцяти секунд при тисячі об’єктах. Результати замірів можна побачити на рисунку 6.12.

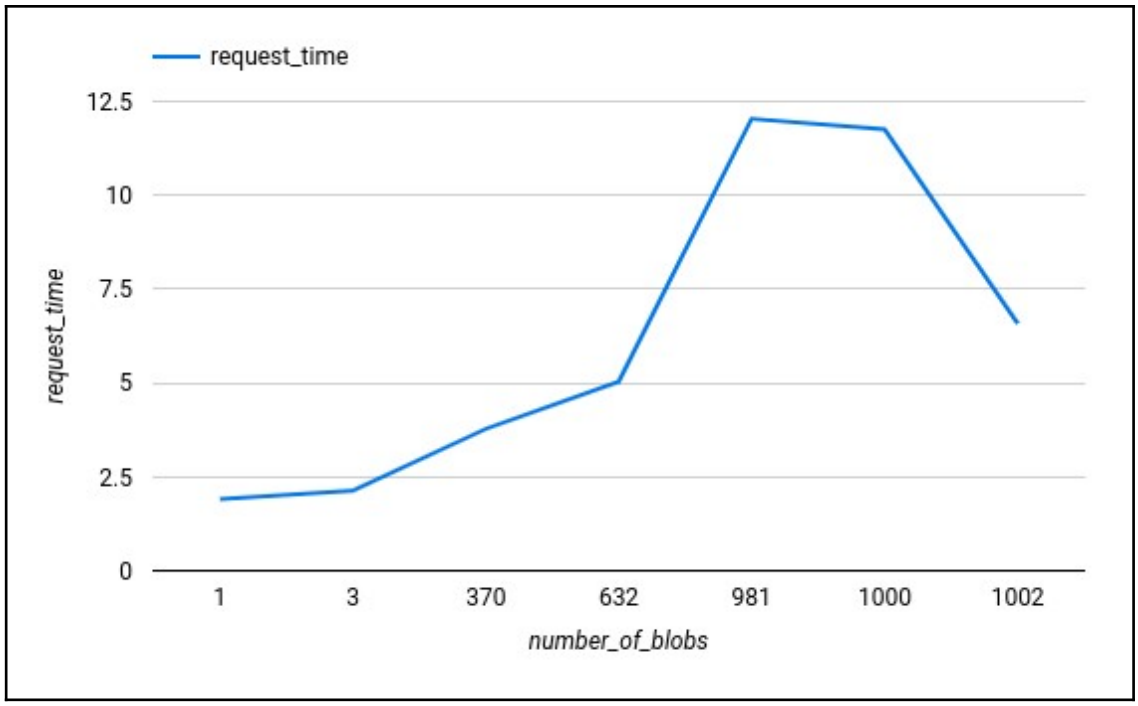

Рисунок 6.12 – Середній час пошуку даних в залежності від кількості результатів на базі розгортування Azure

6.1.1.4 Порівняння результатів замірів часу для всіх варіантів розгортування

З результатів, отриманих після всіх експериментів, можна зробити висновок, що хоча пошук у Озері Даних на основі GCP й працює найбільш стабільно, але так як в ньому фактично неможливо шукати по багатьом параметрам, насправді стабільні показники дає нам Озеро Даних на основі AWS. У такому варіанті розгортання ми отримуємо найшвидший запит, гнучкий пошук по всім можливим параметрам, а також більш-менш прогнозований ріст часу пошуку в залежності від росту даних.

В той же час, розгортування на базі Azure працює повільніше обох інших варіантів, і дає нам найшвидший пошук за майже 2 секунди, що майже у чотири рази повільніше для аналогічного запиту для AWS та два рази за такий же запит для GCP. Також для цього варіанту розгортування ми отримуємо найповільніші запити, що досягають 12 секунд. Незважаючи на повільність, такий вид розгортання дозволяє нам виконувати усі види запитів, що робить цей варіант достатньо гнучким для роботи.

Усі заміри часу в залежності від кількості параметрів пошуку, кількості результатів, а також варіанту розгортання можна побачити у таблиці 1, наведеній нижче.

Таблиця 1 – Швидкість запитів пошуку, в залежності від кількості параметрів пошуку, кількості результатів та варіантів розгортання

Кількість параметрів для пошуку	Кількість результатів	Середня тривалість запиту (в секундах)		
		AWS	GCP	AZURE
1	1	0.56	0.97	1.9
1	370	1.26	1.13	3.78
1	630	1.46	1.24	5.06
1	1000	2.4	1.3	11.76
2	3	0.56	0.93	2.14
2	370	1.99	–	3.78
2	630	2.48	–	5.01
2	982	3.12	1.28	12.03
2	1000	2.99	–	6.54
3	370	1.69	–	3.8
3	630	2.23	–	5.05

6.2 Аналіз можливостей розгортування

Різні варіанти розгортувань дають різні можливості при конфігурації відповідних сервісів, що може бути важливим для рішень з точки зору бізнесу. Нижче в таблиці 2 наведені різні параметри та варіанти розгортань для сховищ, в яких будуть зберігатися самі дані.

Таблиця 2 – Параметри порівняння для сервісів зберігання даних

Функціонал	AWS	GCP	AZURE
Регіони	Північна Америка (Північна Вірджинія, Північна Каліфорнія, Орегон, Огайо) Азія (Мумбаї, Осака, Сеул, Сінгапур, Сідней, Токіо) Канада Європа (Франкфурт, Лондон, Париж, Стокгольм) Південна Америка (Сео Пауло)	Канада (Торонто, Монреаль) Південна Америка (Сео Пауло, Сантьяго) Північна Америка (Айова, Південна Кароліна, Північна Вірджинія, Орегон, Лос Анджелес, Солт Лейк Сіті, Лас Вегас) Європа (Варшава, Фінляндія, Лондон, Франкфурт, Нідерланди, Цюріх, Мілан, Париж) Азія (Тайвань, Гонконг, Токіо, Осака, Сеул, Мумбай, Дельгі, Сінгапур, Джакарта, Сідней, Мельбурн)	Африка Азія (Австралія, Індія, Японія, Корея) Канада Європа (Франція, Німеччина, Норвегія, Швеція, Великобританія, Бразилія, Швейцарія) Південна Америка Північна Америка
Можливість мульти регіонального зберігання	Відсутня	Присутня	Присутня
Версіонування	Присутнє	Присутнє, можливість визначення кількість версій можливість додати захист на видалення об’єкту протягом деякого часу	Присутнє, можливість відновлення видалених файлів протягом деякого часу
Шифрування даних	Присутнє	Присутнє	Присутнє

У таблиці 3 наведені схожі параметри, але вже для сервісів зберігання метаданих.

Таблиця 3 – Параметри порівняння для сервісів зберігання метаданих

Функціонал	AWS	GCP	AZURE
Доступні регіони	Північна Америка (Північна Вірджинія, Північна Каліфорнія, Орегон, Огайо) Африка (Кейптаун) Азія (Гонконг, Джакарта, Мумбаї, Осака, Сеул, Сінгапур, Сідней, Токіо) Канада Європа (Франкфурт, Лондон, Париж, Стокгольм) Південна Америка (Сео Пауло)	Канада (Торонто, Монреаль) Південна Америка (Сео Пауло, Сантьяго) Північна Америка (Айова, Південна Кароліна, Північна Вірджинія, Орегон, Лос Анджелес, Солт Лейк Сіті, Лас Вегас) Європа (Варшава, Лондон, Франкфурт, Нідерланди, Цюріх, Мілан, Париж) Азія (Тайвань, Гонконг, Токіо, Осака, Сеул, Мумбай, Дельгі, Сінгапур, Джакарта, Сідней, Мельбурн)	Південна Америка Північна Америка Європа (Швеція, Швейцарія, Великобританія, Франція, Норвегія) Канада Азія (Австралія, Японія)
Максимальний об’єм даних	1024 Гб * 80 нод = 81920 Гб даних	Необмежений	Необмежений
Шифрування даних	Відсутнє	Присутнє	Присутнє
Можливість мульти регіону	Відсутня, але можливе розгортування 2 або 3 зони всередині одного регіону	Присутня	Присутня

6.3 Оцінка вартості

Для оцінки вартості розглянемо 3 різні ситуації, з якими мого б використовуватися Озеро Даних. Для кожної ситуації припустимо, що будуть використовуватися дані одного й того ж формату і розміру, з однаковою кількістю метаданих, а саме 100 Кб розмір самого файлу, та 10 Кб метаданих для кожного файлу.

Припустимо, що в нас починаючий стартап зі збором даних, і ми тільки починаємо починаємо збирати дані, тільки іноді переглядаючи, що зберігається в нашому Озері Даних. За перший місяць збору даних в нас вийшло 100 000 запитів на запис, на 1000 на зчитування. Загалом ми отримаємо 10Гб самих даних, та 1 Гб метаданих для пошуку.

Далі нам потрібно проаналізувати отримані дані, тому наступний місяць інженери з даних зробили 100 000 запитів на пошук метаданих. Під кінець місяця ми отримали необхідні результати, та видали старі дані.

На третій місяць роботи з даними, проаналізувавши попередні результати і виправивши помилки, ми запустили новий, більш масштабний, збір даних. За третій місяць було виконано 100 000 000 запитів на запис, та 10 000 пошук по метаданим. Загалом в нас вийшло 10 000 Гб даних і 1000 Гб метаданих відповідно.

6.3.1 Оцінка вартості при розгортанні на базі AWS

Для коректної оцінки використаємо конфігурацію розгортування у Північній Вірджинії, з трьома машинами у кластері OpenSearch розміром t3.medium.search.

При таких конфігураціях ціна на запис 1000 об’єктів в S3 буде 0.005 USD, зберігання в місяць 1 Гб на S3 буде 0.023 USD та робота OpenSearch 0.036 USD в годину * 24 * 31 * 3 = 80.352 USD на місяць.

За перший місяць на запис в S3 піде 0.005 USD * 100 = 0.5 USD, за зберігання в S3 10 Гб * 0.023 USD = 0.23 USD, та робота OpenSearch 80.352 USD. Загалом за перший місяць отримаємо 81.082 USD.

За другий місяць запис в S3 0.005 USD * 0 = 0 USD, зберігання в S3 10 Гб * 0.023 USD = 0.23 USD, робота OpenSearch 80.352 USD. Загалом за другий місяць отримаємо 80.582 USD.

За третій місяць запис в S3 0.005 USD * 100 000 = 500 USD, зберігання в S3 10 000 Гб * 0.023 USD = 230 USD, робота OpenSearch 80.352 USD. Загалом за третій місяць отримаємо 810.352 USD.

6.3.2 Оцінка вартості при розгортанні на базі GCP

Для коректної оцінки використаємо конфігурацію розгортування у Північній Вірджинії.

При такій конфігурації ціна на запис 10 000 об’єктів в Cloud Storage буде 0.05 USD, зберігання 1 Гб в Cloud Storage буде 0.02 USD, та робота BigTable 822.56 USD за місяць.

За перший місяць запис в Cloud Storage обійдеться в 0.005 USD * 10 = 0.05 USD, зберігання в Cloud Storage 10 Гб * 0.02 USD = 0.2 USD, робота з BigTable 822.56. Загалом за другий місяць отримаємо 822.81 USD.

За другий місяць на запис в Cloud Storage доведеться заплатити 0.005 USD * 0 = 0 USD, за зберігання в Cloud Storage: 10 Гб * 0.02 USD = 0.2 USD, робота з BigTable 822.56. Загалом за другий місяць отримаємо 822.76 USD.

В третій місяць за запис в Cloud Storage доведеться заплатити 0.005 USD * 10 000 = 50 USD, за зберігання в Cloud Storage: 10 000 Гб * 0.02 USD = 200 USD, робота з BigTable 822.56. Загалом за третій місяць доведеться заплатити 1072.56 USD.

6.3.3 Оцінка вартості при розгортанні на базі Azure

Для коректності порівняння будемо використовувати розгортування у Північній Америці. Ціна на запис 10 000 об’єктів в Cloud Storage буде становити 0.065 USD, зберігання 1 Гб в Blob Storage обійдеться 0.018 USD та іна роботи Cosmos DB за 1 місяць з пропускною здатністю 100 операцій в секунду буде коштувати 46.72 USD.

Таким чином, за перший місяць на запис в Blob Storage піде 0.065 USD * 10 = 0.65 USD, на зберігання в Blob Storage 10 Гб * 0.018 USD = 0.18 USD та на роботу Cosmos DB 46.72 USD. Загалом за перший місяць вийде 47.55 USD.

На другий місяць запис в Blob Storage обійдеться в 0.065 USD * 0 = 0 USD, зберігання в Blob Storage 10 Гб * 0.018 USD = 0.18 USD, та на роботу Cosmos DB піде 46.72 USD. Загалом за другий місяць отримаємо рахунок на 46.9 USD.

За останній місяць стартап витратить на запис в Blob Storage 0.065 USD * 10 000 = 650 USD, на зберігання в Blob Storage 10 000 Гб * 0.018 USD = 180 USD та на роботу з Cosmos DB 46.72 USD. За третій місяць робота з сервісами обійдеться в 876.72 USD.

6.3.4 Порівняння вартості всіх варіантів розгортань

Після проведення аналізу, було підраховано ціну кожного з розгортань в кожному з трьох місяців. Можна помітити, що найбільш дорогим варіантом розгортання є GCP, у перші місяці набагато випереджаючі інші варіанти в вартості.

Так як за перші два місяці майже не витрачалися гроші на запис та зберігання самих даних, то ми не бачимо суттєвої різниці, і основні кошти йдуть саме на оплату ресурсів сервісів для метаданих.

За третій місяць кількість даних дуже зросла, тому значно виросли й кошти. Зі збільшенням даних найдешевшим варіантом виявився AWS.

Нижче наведено таблицю 4, в якій і зібрані всі ці результати.

Таблиця 4 – Результати аналізу ціни кожного з варіантів розгортань

Ціна в USD	AWS	GCP	AZURE
За перший місяць	81.082	822.81	47.55
За другий місяць	80.582	822.76	46.9
За третій місяць	810.352	1072.56	876.72

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи, було розроблено систему розгортання та роботи з Озерами Даних на трьох різних постачальниках хмарних послуг: AWS, GCP та AZURE. Було використано архітектуру з використанням двох сервісів: для зберігання даних та метаданих відповідно.

Для кожної з цих архітектур було написано адаптер, що дозволило використовувати один і той самий інтерфейс взаємодії незалежно від варіанту розгортання.

Такий спосіб роботи Озерами Даних дозволив оцінити та порівняти різні варіанти розгортань за багатьма напрямками, провести аналіз та зробити висновки про доцільність використання в тих чи інших умовах.

Так, варіант розгортання Azure виявився найбільш повільним, як під час створення даних, так і під час пошуку. Сервіси Azure дозволили в повній мірі реалізувати весь функціонал, і при використанні невеликої кількості даних їх вартість менша, ніж аналоги від інших постачальників хмарних послуг, але при збільшенні кількості даних ціна зростає, а робота з цими сервісами стає ще більш повільною, тому не рекомендується використовувати це рішення.

Реалізація Озера Даних з використанням сервісів GCP виявилася досить швидкою, і найменш вразливою до кількості даних. Головними недоліками цього варіанту розгортання є завелика стартова ціна, навіть при малих об’ємах даних, та неможливість працювати з пошуком по кільком параметрам метаданих, що є великим недоліком при аналізі даних. В цілому, це рішення можливо використовувати, якщо модифікувати роботу з метаданими, та якщо у проекта закладений великий бюджет для Озера Даних.

Останнім же варіантом, що аналізувався у цій роботі, було розгортання на базі сервісів AWS. Це рішення підтримувало повний набір необхідних функцій, а також вийшло досить швидким та не дуже дорогим, у порівнянні з іншими рішеннями. Саме це рішення рекомендується використовувати у загальному варіанті, якщо не накладається додаткових умов або обмежень. Головним

недоліком є менша кількість базових функцій у сервісів, на базі яких реалізується Озеро Даних.

У подальшому, можливо досліджувати різні Озера Даних як зі схожою архітектурою, використовуючи інші постачальники, як наприклад Digital Cloud та Alibaba Cloud, так і з різною, використовуючи сервіси для зберігання данних з іншою природою (Filestore, HDFS), або метаданих (SQL бази, пошукові системи).

Також можливо дослідити всі ці архітектури, використовуючи більше даних, з іншим форматом, а також у більшій кількості.

ПЕРЕЛІК ПОСИЛАНЬ

1 A. Gorelic, The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science / Alex Gorelik – London, 2019. – 224 с.

2 Smelyakov, K., Chupryna, A., Sandrkin, D., Kolisnyk, M.Search by Image Engine for Big Data Warehouse - 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences, eStream 2020 - Proceedings, 2020

3 K. Tovmasyan, Mastering AWS CloudFormation: Plan, develop, and deploy your cloud infrastructure effectively using AWS CloudFormation / Karen Tovmasyan – Birmingham, 2020. – 300 с.

4 Y. Brikman, Terraform: Up & Running: Writing Infrastructure as Code 2nd Edition / Yevgeniy Brikman – California, 2019. – 368 с.

5 N. Poulton, P. Joglekar, The Kubernetes Book / Nigelauthor Poulton, Pushkar Joglekar – Birmingham, 2019. – 228 с.

6 E. Matthes, Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming 2nd Edition / Eric Matthes – San Francisco, 2019. – 544 с.

7 K. Chodorow, E. Brazil, MongoDB: The Definitive Guide: Powerful and Scalable Data Storage 3rd Edition / Kristina Chodorow, Eoin Brazil – Sebastopol, 2010. – 216 с.

8 F. Voron, Building Data Science Applications with FastAPI: Develop, manage, and deploy efficient machine learning applications with Python / Voron François – Birmingham, 2021. – 426 с.

9 A. Videla, Jason J.W. Williams, RabbitMQ in Action: Distributed Messaging for Everyone / Alvaro Videla, Jason J.W. Williams – Manning, 2012 – 253 с.

10 F. Pierfederici, Distributed Computing with Python: Harness the power of multiple computers using Python through this fast-paced informative guide / Francesco Pierfederici – Birmingham, 2016. – 170 с.

11 H. Lee, Hands On With Google Data Studio: A Data Citizen's Survival Guide / Lee Hurst – Hoboken, 2020. – 432 с.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

2. Smelyakov, K., Chupryna, A., Sandrkin, D., Kolisnyk, M.Search by Image Engine for Big Data Warehouse - 2020 IEEE Open Conference of Electrical, Electronic and Information Sciences, eStream 2020 - Proceedings, 2020