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 впливу архітектури на можливості, вартість та швидкодію системи Data Lake
(тема)

Виконав: студент	2 курсу групи ІПЗм-20-4
	Хованський О.О.
	(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

студента Хованському Олександру Олександровичу
(прізвище, ім’я, по батькові)

1. Тема роботи: «Дослідження впливу архітектури на можливості, вартість та швидкодію системи Data Lake»
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, менеджмент проектів, п’ять варіантів архітектури Озера Даних, арі для створення та пошуку даних*.
Використовувати: MongoDB, Celery, Manjaro linux, python, Terraform
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_____ (Хованський О.О.)

РЕФЕРАТ / ABSTRACT

Звіт з практики містить: 61 с., 72 рис., 3 таб., 11 джер.

ВЕЛИКІ ДАНІ, ОЗЕРО ДАНИХ, AWS, GCP, GOOGLE DATA STUDIO, PYTHON, FASTAPI, GRPC, API, REST, МІКРОСЕРВІСНА АРХІТЕКТУРА

Об’єктом дослідження є залежність можливостей та вартості озера даних в залежності від постачальника хмарних послуг та обраної архітектури.

Метою практики є порівняння швидкості та вартості різних архітектур озера даних на різних провайдерах.

Результатом практики є програмна система з мікросервісною архітектурою що дозволяє автоматично розгортати озеро даних, та використовувати його через rest api.

BIG DATA, DATA LAKE, AWS, GCP, AZURE, GOOGLE DATA STUDIO, PYTHON, FASTAPI, GRPC, API, REST, MICROSERVICE ARCHITECTURE

The object of analysis and research is the dependence of capabilities and cost of data lakes with different architectures on different cloud providers.

The aim of the practice is to compare the speed and cost of different data lake architectures on different cloud providers.

The result of the practice is a software system with microservice architecture, which allows user to automatically deploy the data lake and use it through rest api.

Я, Хованський Олександр Олександрович, студент групи ІПЗм-20-4, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження впливу архітектури на можливості, вартість та швидкодію системи Data Lake», 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[1], 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 експерементувати. Стандартні СУБД [2] сильно втрачають у швидкості пошуку та вставки даних при зростанні розміру БД. Тому була створена архітектура зберігання даних Data Lake, що не має таких недоліків. Оскільки для створення Data Lake на власному сервері потрібні навички адміністрування систем, останнім часом все частіше вдаються до використання cloud-провайдерів.

Метою даної роботи є створення інформаційної системи з автоматичного розгортання Data Lake [3] на двох cloud провайдерах, з можливістю використання п’яти різних архітектур.

Метою цієї статті є порівняння п’ятьох варіантів реалізації Data Lake на цих провайдерах за такими параметрами: швидкість, вартість, розташування серверів, можливості сервісів, що беруть участь у деплої.

1 АНАЛІТИЧНИЙ ОГЛЯД

Озеро даних — це система або сховище даних, що зберігаються в природному/необробленому форматі, зазвичай у вигляді файлів. Озеро даних зазвичай є єдиним сховищем даних для оброблених та необроблених даних, включаючи необроблені копії вихідних системних даних, дані датчиків, соціальні дані тощо, а також перетворені дані, які використовуються для таких завдань, як звітування, візуалізація, розширена аналітика та машинне навчання. Озеро даних може включати структуровані дані з реляційних баз даних (рядки і стовпці), напівструктуровані дані (CSV, журнали, XML, JSON), неструктуровані дані (електронна пошта, документи, PDF-файли) і двійкові дані (зображення, аудіо, відео). Погано керовані озера даних жартівливо називають болотами даних.

Багато компаній використовують хмарні служби зберігання даних, такі як Google Cloud Storage і Amazon S3, або розподілену файлову систему, таку як розподілена файлова система Apache Hadoop (ADFS) [4]. Існує поступовий академічний інтерес до концепції озер даних. Наприклад, Personal DataLake в Університеті Кардіффа — це новий тип озера даних, метою якого є управління великими даними окремих користувачів шляхом надання єдиної точки збору, організації та обміну особистими даними.

Попереднє озеро даних (Hadoop 1.0) мало обмежені можливості з його пакетно-орієнтованою обробкою (Map Reduce) і було єдиною парадигмою обробки, пов’язаною з нею. Взаємодія з озером даних означала, що потрібно мати досвід роботи з Java з map reduce і інструментами вищого рівня, такими як Apache Pig, Apache Spark [5] і Apache Hive [6] (які самі по собі спочатку були орієнтовані на пакетну роботу).

2 АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ І АЛГОРИТМІВ

Існує невелика кількість рішень data lake as a service. Наприклад: bigstep, dataleyk, jet.su, qubole. Всі вони працюють з корпоративними клієнтами та не доступні для звичайного користувача. Рішення для автоматичного розгортання data lake ще немає на ринку.

Деякі постачальники хмарних рішень надають документацію з розгортання data lake на їхніх потужностях із переліком необхідних сервісів. Розглянемо архітектуру data lake, яку пропонує AWS. Нижче наведено діаграму архітектури.

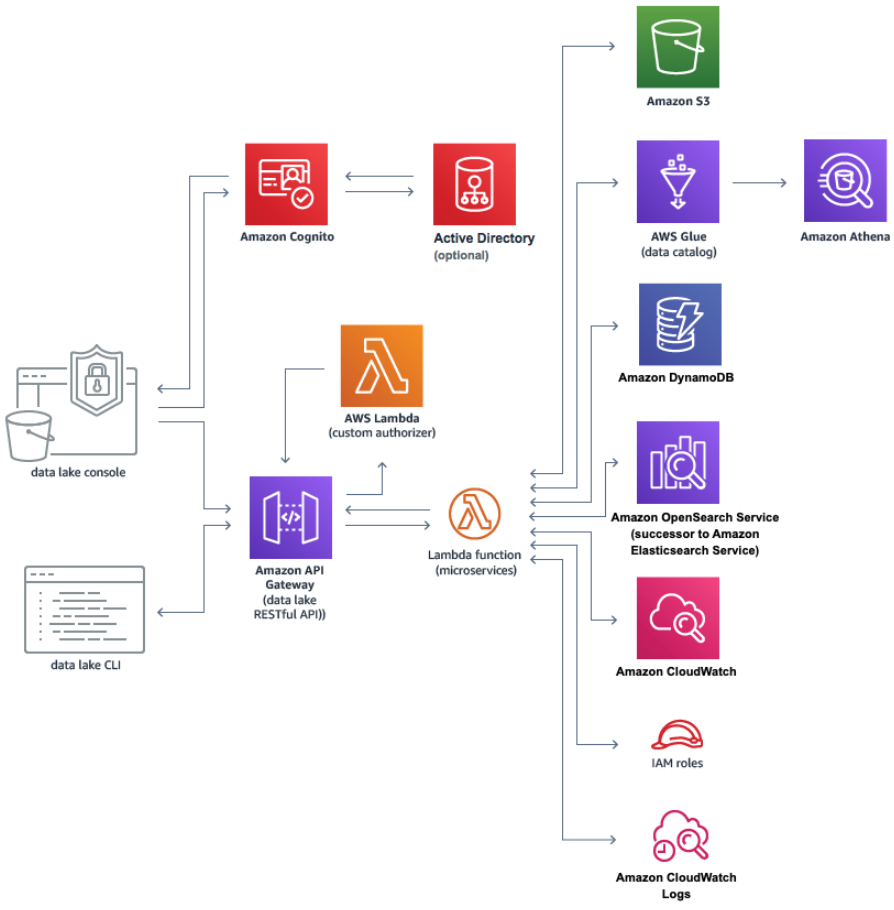

Рисунок 1 – Архітектура Озера Даних від AWS

Для деплою використовується AWS CloudFormation, який кофігурує основні сервіси, що включають набір мікросервісів (функцій) AWS Lambda, Amazon

OpenSearch Service (наступник Amazon Elasticsearch Service) для надійного пошуку, Amazon Cognito для автентифікації користувачів, AWS Glue для перетворення даних та Amazon Athena для аналізу , AWS S3 для зберігання даних, AWS DynamoDB для метаданих. Через велику кількість сервісів рішення виходить дорогим, але досить повним. Не всім користувачам потрібні всі ці послуги. Також необхідно знання AWS CloudFormation для використання шаблону від AWS.

Тепер розглянемо архітектуру від Azure. Нижче наведено діаграму архітектури. Для обробки даних використовується HDInsight - хмарна служба Apache Spark і Hadoop, BlobStorage як сховища. Сервіс HDInsight вимагає сервери, що постійно працюють, що призводить до великих витрат.

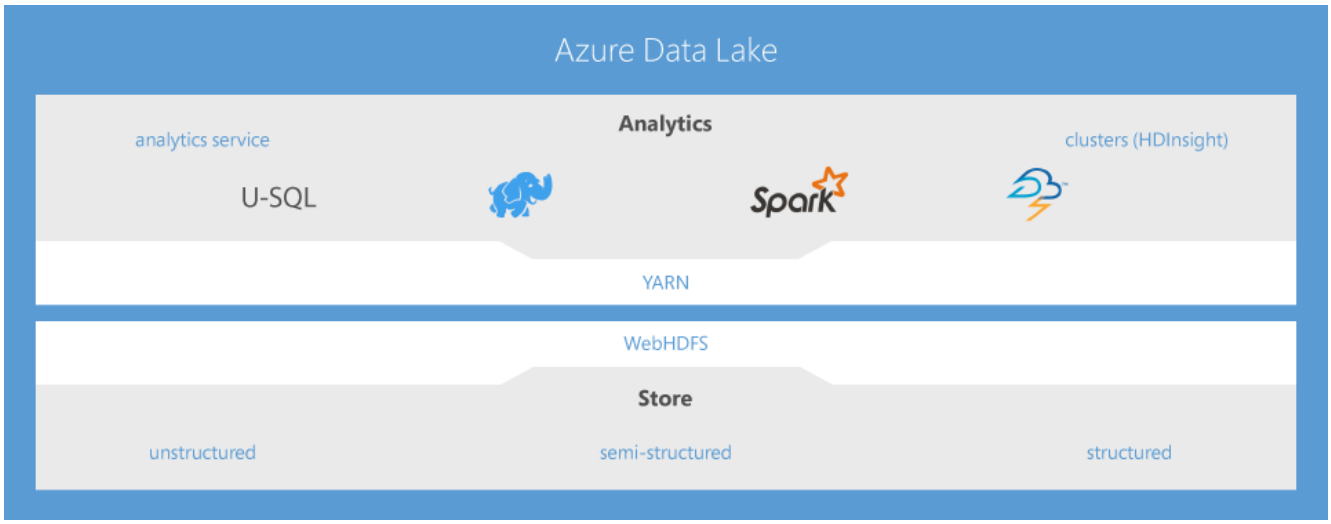

Рисунок 2 – Архітектура Озера Даних від Azure

Рішення від постачальників хмарних рішень підходять для великих компаній із великими вимогами до data lake та достатньою кількістю грошей. Щоб почати використовувати data lake необов'язково створювати такі складні та дорогі архітектури. Рішення створюване у межах даної практики має вирішити цю проблему.

3 ПОСТАНОВКА ЗАДАЧІ

Можливості, які має надавати сервіс:

а) авторизація;

б) управління проектами;

в) підтримка двох основних хмарних провайдерів (GCP, AWS);

г) верифікація сервісних ключів;

ґ) можливість деплою 5-х різних архітектур (3 на GCP, 2 на AWS).

Розберемо за пунктами.

Передбачається, що сервісом користуватиметься велика кількість людей, тому в системі необхідна авторизація.

Один користувач може мати безліч проектів, тому потрібно буде передбачити можливість створення, читання, редагування, видалення проектів.

GCP, AWS є двома найбільшими постачальниками хмарних послуг зі своїми потужностями, тому їх ціни, а значить і ціни розгортаэмих data lake, будуть найнижчими з можливих.

Для розгортання data lake необхідні ключі від сервісного облікового запису постачальника. Необхідно передбачити вірифікацію цих ключів.

У рамках практики було вирішено розробити 4 варіанти архітектури. 3 варіанти будуть використовувати подібні послуги на різних постачальниках. Варіант буде використовувати інші послуги для порівняння впливу архітектури на вартість і швидкість в рамках одного постачальника.

Для порівняння архітектури будуть використовуватися два набори даних. Перший набір буде складатися з даних температурного датчика iot пристрою. Усі дані у цьому наборі мають однаковий розмір. Другий набір містить у собі картинки різного розміру та змісту у форматі jpeg. В результаті ми отримаємо два графіки: середня швидкість завантаження в залежності від типу файлу, середня швидкість завантаження в залежності від розміру файлу. Графіки будуть створені за допомогою Google BigQuery [7] та Google DataStudio [8].

4 РЕАЛІЗАЦІЯ

4.1 Загальний опис системи

Система складається з трьох сервісів. Auth service, Deploy service, API service. Рішення про мікросервісну архітектуру було прийнято через функціонал API service. На нього передбачається високе навантаження, так як через нього будуть проходити всі файли користувача які завантажуються в data lake. В той час, як auth і deploy сервіс будуть відчувати набагато менше навантаження. Спілкування між сервісами відбувається за протоколом gRPC [9]. Як основний фреймфорок для всіх сервісів був обраний FastAPI [10], так як він є найшвидшим веб-реймворком на Python на даний момент. Це дозволило швидко розробити систему, яка задовольняє нашим запитам швидкодії. Діаграма взаємодії сервісів наведена нижче.

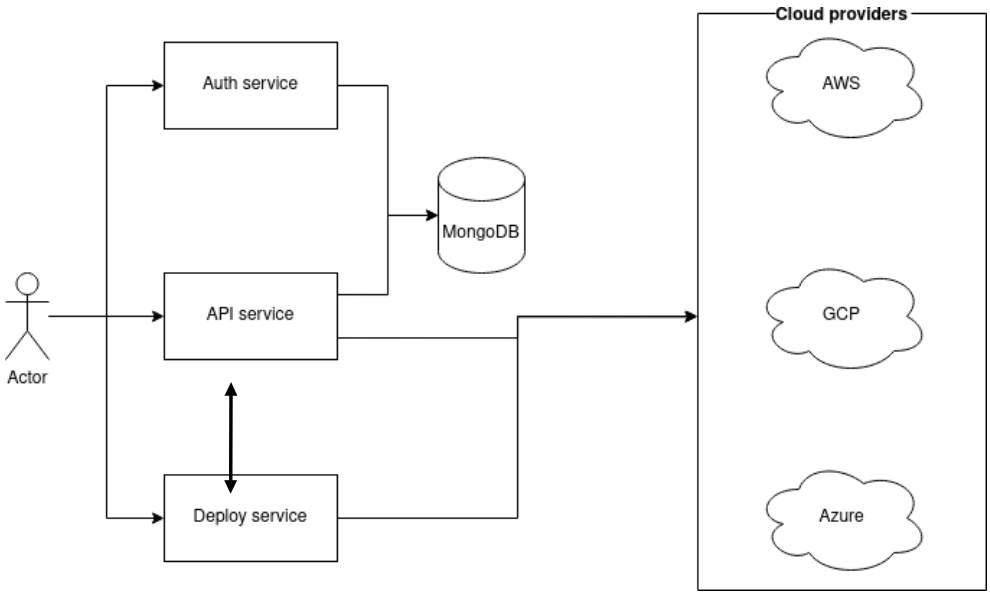

Рисунок 3 – Діаграма взаємодії сервісів

4.2 Опис кожного сервісу

4.2.1 Auth service

Для авторизації була використана технологія jwt токенів. Для цього було використано бібліотеку fastapi-users [11]. Вона дозволяє вибрати одну з декількох

технологій авторизацій, у тому числі jwt токени. Для зберігання даних користувачів використовується база даних MongoDB NoSQL. Структура бібліотеки наведена нижче.

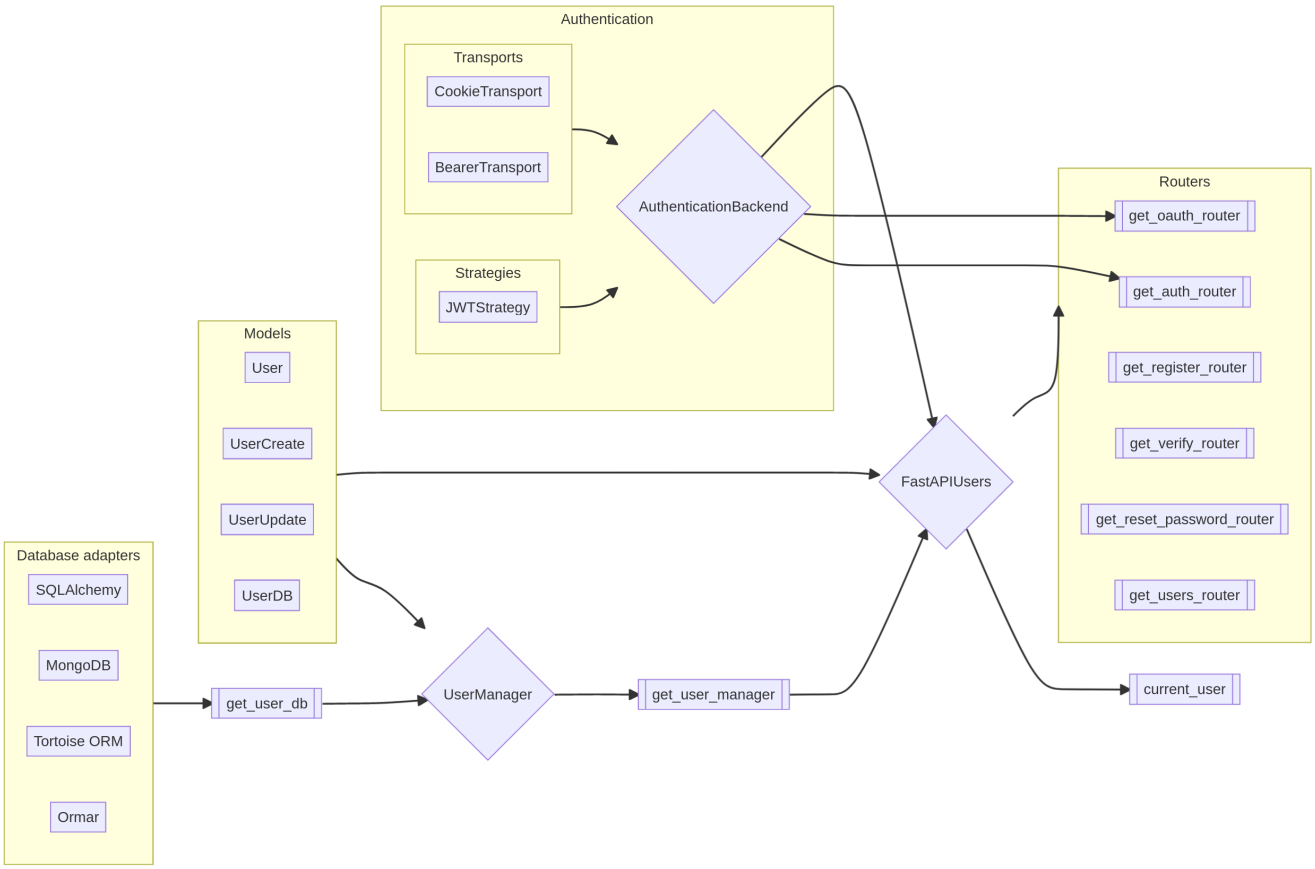

Рисунок 4 – Структура бібліотеки fastapi-users

FastAPI має вбудовану підтримку OpenAPI, що дозволяє автоматично генерувати документацію та використовувати точки доступу через swagger. Це дозволяє прискорити розробку, оскільки розробник не витрачає час на налаштування інструменту. Інтерфейс наведено на малюнку нижче.

Користувачу доступні дві основних точки доступу: /auth/regiseter, /auth/jwt/login. Перша дозволяє реєструватися у сервісі, друга – авторизуватися. При авторизації буде отримано jwt токен, який необхідно використовувати у заголовках запиту двох інших сервісів.

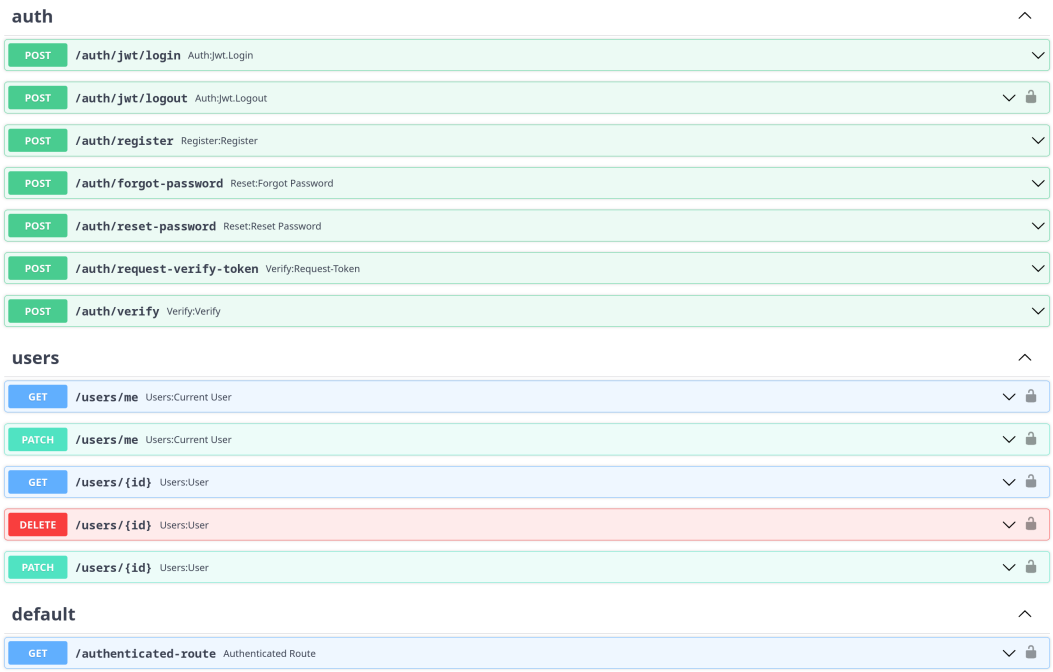

Рисунок 5 – Auth service swagger

4.2.2 Deploy service

Deploy сервіс складається із сервера на FastAPI та MongoDB. Усього було реалізовано п’ять варіанів деплою на двох провайдерах. Рогортання відбувається за допомогою celery worker, що використовує Terraform.

Terraform — програмне забезбечення з відкритим вихідним кодом розробленний компанією HashiCorp. Перший варіант реалізації deploy service використовував бібліотеки надані постачальником для розгортання кожного сервісу використовуванного в архітектурі. Це призводило до необхідності обробляти всі можливі помилки самостійно. Terraform дозволяє перейти від імперативного методу опису інфраструктури до декларативного. Він використовує конфігураційні файли для обису бажанного кінцевого стану. Використання terraform пришвидшило розробку нових архітектур, оскільки відпала необхідність в обробці помилок бібліотек. Також terraform має гарну документацію та підтримує необхідних нам провайдерів.

Доступ до проекту користувача на сервисі провайдера ми отримуємо через сервіс акаунти, реквізити для входу нам надає користувач через endpoint /v1/project_credentials/{project_id}. Розглянемо кожен варіант деплою.

GCP_1. У цій архітектурі як сховище даних і метаданих використовується Google BigQuery. BigQuery - хмарне сховище даних, що підтримує SQL запити та безліч агрегаційних функцій. Воно спеціально оптимізоване під потокову вставку даних та може витримувати великі навантаження. Але ця оптимізація накладає обмеження на можливість модифікації щойно доданих даних. Модифікувати файли можна лише за кілька годин (час може відрізнятися). Це обмеження поширюється лише на потокову вставку, на пакетну (csv, jsonl, avro файли) таких обмежень немає.

GCP_2. У цій архітектурі використовуються Google BigTable та Google Cloud Storage. BigTable – високопродуктивна NoSQL база даних. Вона використовується для метаданих. Cloud Storage – об'єктне сховище. У ньому зберігатимуться дані, які будуть представлені у вигляді файлів (навіть якщо це текстові дані). Ця архітектура не має істотних обмежень і є аналогом AWS_2.
Нажаль у GCP немає окремого сервісу ElasticSearch, вони пропонують використовувати рішення від компанії Elastic. Через це неможливо автоматично розгорнути ElasticSearch на GCP, тому він був замінений на BigTable, як подібне рішення.

GCP_3. У цій архітекутрі використовується Google BigQuery та Google Cloud Storage. Ця архітектура є подєнанням перших двох архітектур. Метадані зберігаються у Bigquery, а файли у Cloud Storage.

AWS_1. У цій архітектурі використовується AWS S3 як сховище даних та AWS OpenSearch (ElasticSearch) як сховище метаданих. S3 є аналогом Cloud Storage – об'єктним сховищем. Дані в ньому будуть представлені у вигляді файлів. OpenSearch (ElasticSearch) – пошуковий двигун з NoSQL сховищем, що забезпечує відмінну швидкість пошуку

AWS_2. цій архітектурі використовується AWS S3 як сховище даних та AWS DynamoDB як сховище метаданих. DynamoDB – NoSQL база даних, аналог BigTable. Цю архітектуру можна вважати аналогом GCP_2.

Наступний код містить клас відповідальний за розгортування архітектури GCP_1.

```
class BigQueryResource(BaseModel):
    project: str
    dataset: str
    table: str


class GCPDeployedResources1(BaseModel):
    bigquery: BigQueryResource


class GCPBlobHandler1(BaseBlobHandler):
    async def update_blob_data(
        self,
        full_project_structure: FullProjectStructure,
        processed_data: ProcessedData,
        blob_id: str,
    ):

        blob_d = await self.get_blob_from_first_step(blob_id)
        deployed_resources = GCPDeployedResources1(
            **full_project_structure.deploy.project_structure
        )
        gcp_credentials_path = get_credentials_tmp_path(
            full_project_structure.credentials
        )
        credentials =
service_account.Credentials.from_service_account_file(
            filename=gcp_credentials_path,
            scopes=["https://www.googleapis.com/auth/cloud-platform"],
        )
        client = bigquery.Client(credentials=credentials)
        table_id = (
            f"{deployed_resources.bigquery.project}."
            f"{deployed_resources.bigquery.dataset}."
            f"{deployed_resources.bigquery.table}"
        )

        blob_d["file"] = _bytes_to_json(processed_data.data)
        blob_d["size"] = len(processed_data.data)
        blob_d["system_tags"] = [tag.dict() for tag in
processed_data.system_tags]
        errors = client.insert_rows_json(table_id, [blob_d])
        if len(errors) == 0:
            await self.delete_blob_from_first_step(blob_id)

        for error in errors:
            LOGGER.error(f"Insert error: {error}")
```

```
    async def search_by_tags(
        self, full_project_structure: FullProjectStructure, tags: list[Tag]
    ):
        deployed_resources = GCPDeployedResources1(
            **full_project_structure.deploy.project_structure
        )
        gcp_credentials_path = get_credentials_tmp_path(
            full_project_structure.credentials
        )
        credentials =
service_account.Credentials.from_service_account_file(
            filename=gcp_credentials_path,
            scopes=["https://www.googleapis.com/auth/cloud-platform"],
        )
        client = bigquery.Client(credentials=credentials)
        table_id = (
            f"{deployed_resources.bigquery.project}."
            f"{deployed_resources.bigquery.dataset}."
            f"{deployed_resources.bigquery.table}"
        )

        sql = f"""
            SELECT DISTINCT
                id,
                name,
                type,
                size,
                timestamp,
                source,
                TO_JSON_STRING(user_tags) AS user_tags,
                TO_JSON_STRING(system_tags) AS system_tags
            FROM
              `{table_id}`
        """
        query_parameters = []

        for i, tag in enumerate(tags):
            if i == 0:
                sql += "\nWHERE\n"
            else:
                sql += "\nAND\n"

            sql += f"""
            (
                STRUCT(@tag_name_{i} AS name, @tag_value_{i} AS value) IN
UNNEST(user_tags)
                OR
                STRUCT(@tag_name_{i} AS name, @tag_value_{i} AS value) IN
UNNEST(system_tags)
            )
            """
            query_parameters.extend(
                (
                    bigquery.ScalarQueryParameter(f"tag_name_{i}",
"STRING", tag.name),
                    bigquery.ScalarQueryParameter(
```

```
                        f"tag_value_{i}", "STRING", tag.value
                    ),
                )
            )

        job_config =
bigquery.QueryJobConfig(query_parameters=query_parameters)
        res = client.query(sql, job_config=job_config)

        response = []
        for row in res:
            response.append(
                Blob(
                    blob_id=row.id,
                    name=row.name,
                    content_type=row.type,
                    timestamp=row.timestamp.isoformat(),
                    source=row.source,
                    user_tags=json.loads(row.user_tags),
                    system_tags=json.loads(row.system_tags),
                    size=row.size,
                )
            )

        print(len(response))

        return response
```

Оскільки сервер написано на FastAPI нам доступний Swagger, як і в Auth service. На наступному рисунку наведені всі кінцеві точки, з якими взаємодіє користувач.

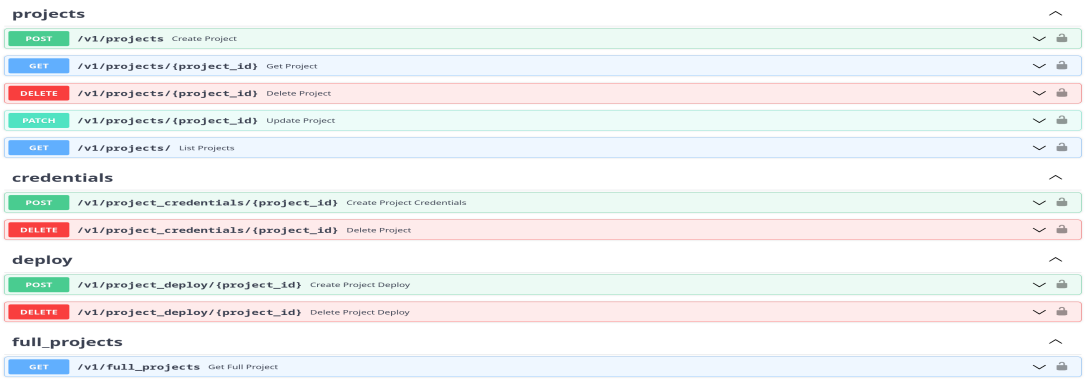

Рисунок 6 – Deploy service swagger

4.2.3 API service

Цей сервіс складається із сервера написаного на FastAPI та MongoDB. Через цей сервіс проходять файли та запити користувача. При завантаженні файлу крім

метаданих, які додав користувач, додаються системні метадані, які можна отримати проаналізувавши файл. Наприклад, при завантаженні зображення буде використано інструмент exiftool, який дозволяє працювати з метаданими зображень (розмір, тип, геолокація, колірна схема, тощо). Код роботи з метаданими зображень наведено нижче.

```
class ImagesJpegProcessor(BaseDataProcessor):
    async def process_request(self, request: Request) -> ProcessedData:
        image_data = await request.body()

        # exiftool necessarily needs the path to the file to work
        with tempfile.NamedTemporaryFile() as tf, exiftool.ExifTool() as
et:
            tf.write(image_data)
            tf.seek(0)
            metadata = et.get_metadata(tf.name)

        tags = convert_metadata_to_tags_list(metadata)
        return ProcessedData(data=await request.body(), system_tags=tags)
```

Усього для роботи доступно три endpoint. Два для створення файлу та один для пошуку за метаданими. Також є Swagger, скріншот наведено на малюнку нижче.

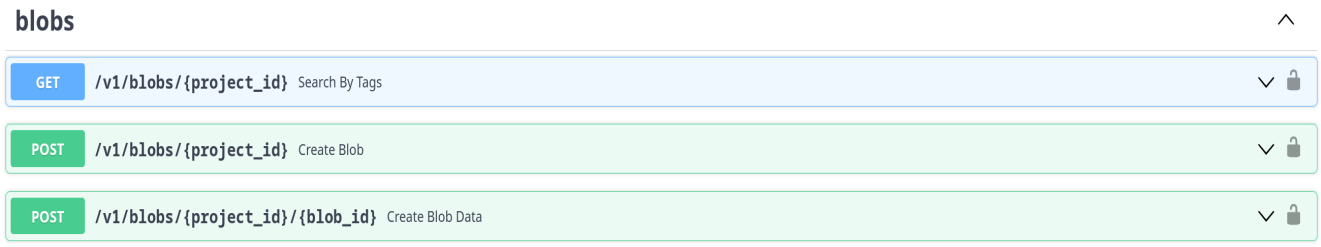

Рисунок 7 – API service swagger

Створення файлу відбувається в два етапи. На першому етапі відбувається додавання користувацьких метаданих. Зберігання користувацьких метаданих відбувається у тимчасовій таблиці MongoDB. На другому етапі користувачем завантажується сам файл, після чого відбувається вилучення системних

метаданих. Далі сервер комбінує файл та його системні метадані за метаданими з MongoDB та відправляє у потрібні сервіси.

4.3 Доступність використовуваних сервісів за регіонами

Одна з переваг хмарних сервіс провайдерів це можливість обирати у якому регіоні буде рогорнуто сервіс. Хмарні провайдери мають безліч data центрів по всьому світу і використання регіону територіально наближенного до кінцевого користувача дозволить пришвидшити дію додатку (на думку користувача).

4.3.1 Доступність регіонів у GCP

Для розгортання Data Lake на GCP ми використовуємо 3 сервіси: Cloud Storage, BigQuery, BigTable. На рисунку 8 зображенно мапу доступних регіонів для Cloud Storage.

Рисунок 8 – Доступні регіони для Cloud Storage

На мапі можна побачити гарне покриття західної Европи, північної Америки та східної та північної Азії. Є лише один сервер у східній Америці та два

в Австралії. Взагалі нема серверів у Африці, центральній Азії та на Близькому Сході. Також є можливість обрати один з трьох мультирегіонів (eu, ua, asia).

На рисунку 9 зображенно мапу доступних регіонів для BigQuery.

Рисунок 9 – Доступні регіони для BigQuery

Bigquery має схоже з Cloud Storage покриття. Різниця є у двох регіонах: Париж, Мадрид. Вони є тільки у Cloud Storage. Також немає можливості обрати asia мультирегіон. Проте є цікава можливисть використати регіони інших хмарних провайдерів (aws-us-east-1 та azure-eastus2).

На рисунку 10 зображенно мапу доступних регіонів для BigTable.

BigTable має такеж покриття як і у Cloud Storage, проте через те що BigTable є постійно працюючою віртуальною машиною в нього нема можливості обрати мультирегіон.

GCP також мають Low CO2 регіони. Це регіони, які мають CFE% (безвуглецевої енергії) щонайменше 75%. Ці регіони зображенно на рисунку 11.

Рисунок 10 – Доступні регіони для BigTable

Рисунок 11 – Доступні Low CO2 регіони

4.3.2 Доступність регіонів у AWS

Для розгортання Data Lake на AWS ми також використовуємо 3 сервіси: S3, OpenSearch, DynamoDB. На рисунку 12 зображенно мапу доступних регіонів для S3.

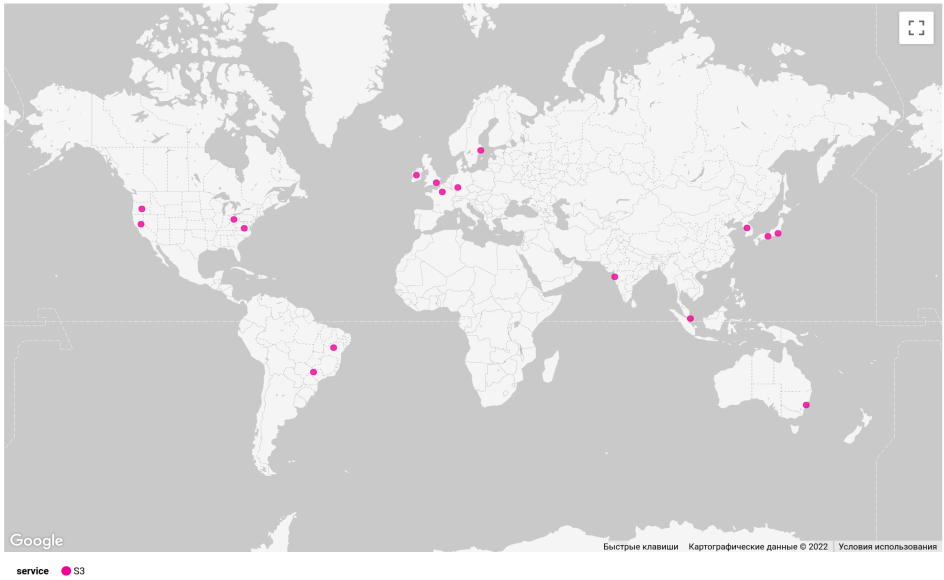

Рисунок 12 – Доступні регіони для S3

На данний момент доступно 17 регіони для розгортання S3. На мапі можна побачити помірне покриття Европи, Північної Америки та Східної Азії. Від одного до двох серверів є в Південній Америці, Австралії та Центральній Азії. Також є вимкненні сервери. На рисунку 13 зображенно мапу відімкненних регіонів S3.

На мапі можна побачити 5 відключенних серверів. На ній можна побачити що більше неможливо створити S3 в Африці та Близькому Сході. Інші регіони мають альтернативи недалеко від них.

Взагалом можна відзначити меншу варіативність порівнянно з Cloud Storage у GCP. Проте теріторіально сервери знаходяться поруч. Скоріш за все це відбувається через різну політику збільшення потужностей. AWS модернізує існуючи датацентри в той час коли Google створює нові. Можливий також інший варінат. AWS може групувати близько розташовані датацентри для спрощення сприяння користувачем.

На рисунку 14 зображенно доступні регіони для сервісу OpenSearch.

Можна побачити тіж регіони що й у S3, разом з вимкненними регіонами. З цього можна зробити висновок, що AWS вимикає регіон для деяких сервісів коли в датацентрі не вистачає обладння.

Усього доступно 22 регіони для розгортання OpenSearch. На мапі можна побачити помірне покриття Європи, Північної Америки та Східної Азії. Від одного до двох серверів є в Південній Америці, Австралії, Центральній Азії та Близькому Сході.

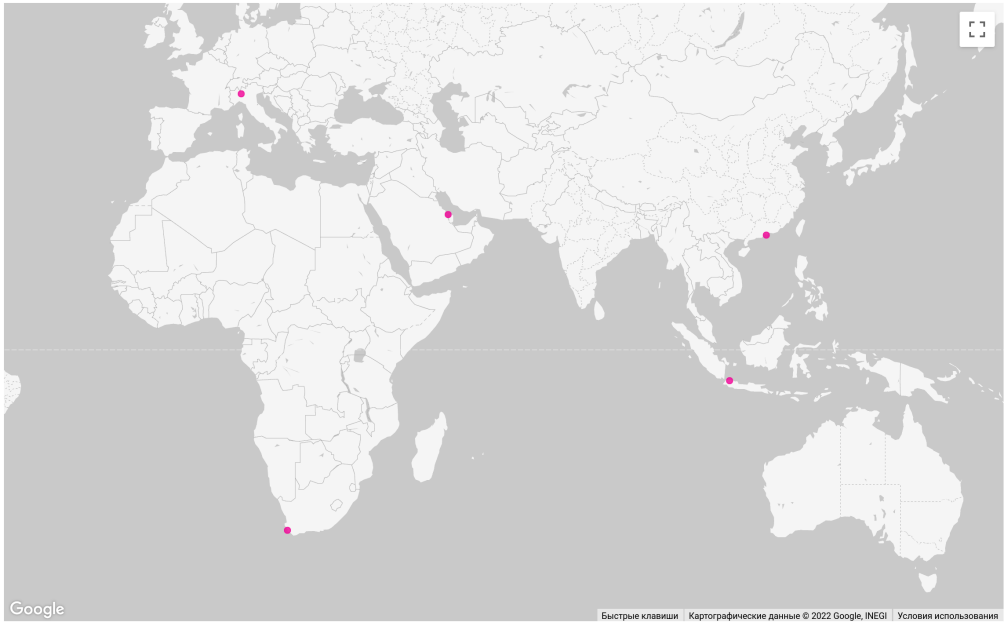

Рисунок 13 – Вимкненні регіони S3

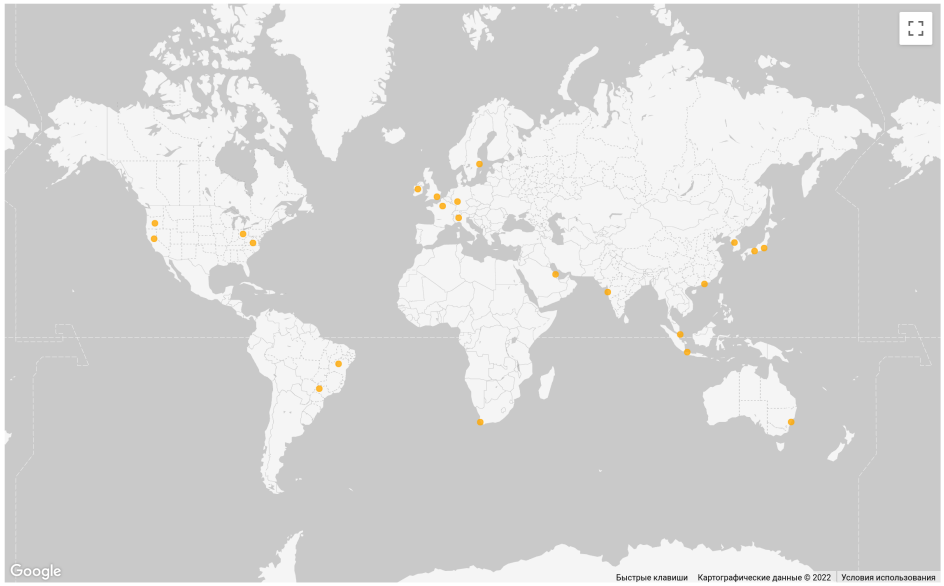

Рисунок 14 – Доступні регіони OpenSearch

На рисунку 15 зображенно доступні регіони для сервісу DynamoDB.

Рисунок 15 – Доступні регіони DynamoDB

DynamoDB використовує тіж 22 регіони що й OpenSearch. Тож усі висновки OpenSearch актуальні для DynamoDB.

5 ЕКСПЕРИМЕНТ

5.1 Опис експеримента

В рамках роботи було проведенно заміри швидкості роботи кожного з варіантів архітектури Data Lake та розрахунки вартості. Для замірів швидкості запису було використанно два набори данних. Перший набір являє собою записи IoT датчиків температури. Другий набір являє собою набір різних зображень у форматі jpeg. З кожного набору даних було обрано по тисячі файлів. Таким чином було перевірено швидкість запису для файлів форматів application/json та image/jpeg. Для замірів швидкості пошуку було використано передзавантажену тисячу json файлів. Для розрахунку вартості кожного варіанту Data Lake була змоделйована ситуація використання Data Lake з пристроями IoT. Умови розрахунку були такими: щомісяця завантажується 10 Тб даних, читається 1 Тб даних, метадані займають у 3 рази більше місця за файлі.

5.2 Результати замірів швидкості

5.2.1 GCP1

На рисунку 16 зображенно залежність середнього часу вставки файлу від його типу.

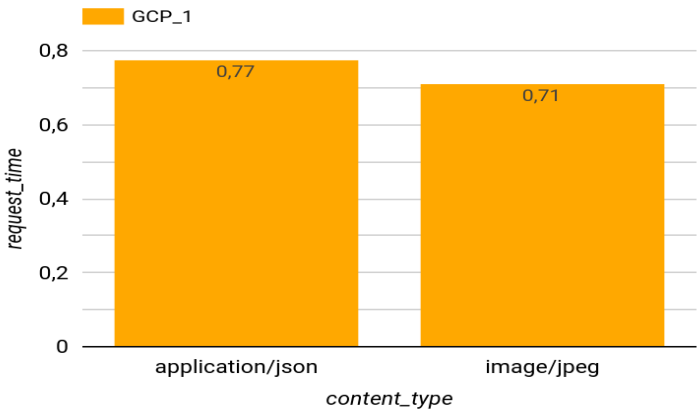

Рисунок 16 – Залежність середнього часу вставки файла від типу

На рисунку 17 зображенно залежність середнього часу вставки файлу від його розміру.

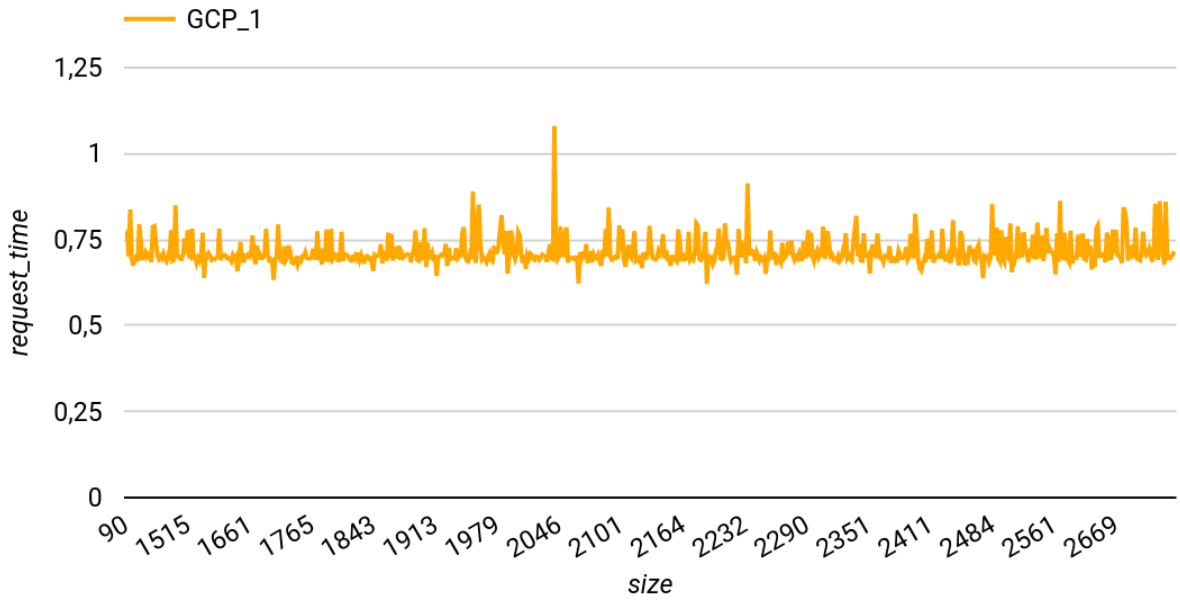

Рисунок 17 – Залежність середнього часу вставки файла від розміру

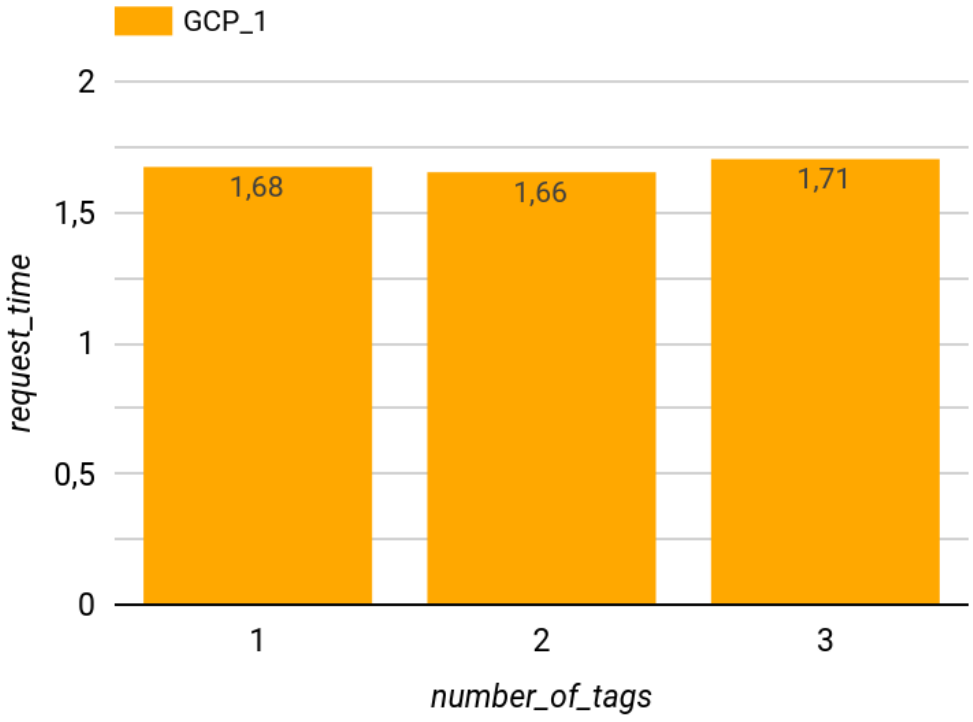

Рисунок 18 – Залежність середнього часу пошуку від кількості параметрів

На рисунку 18 та 19 зображенно залежність середнього часу пошуку від кількості параметрів та кількості знайденних файлів.

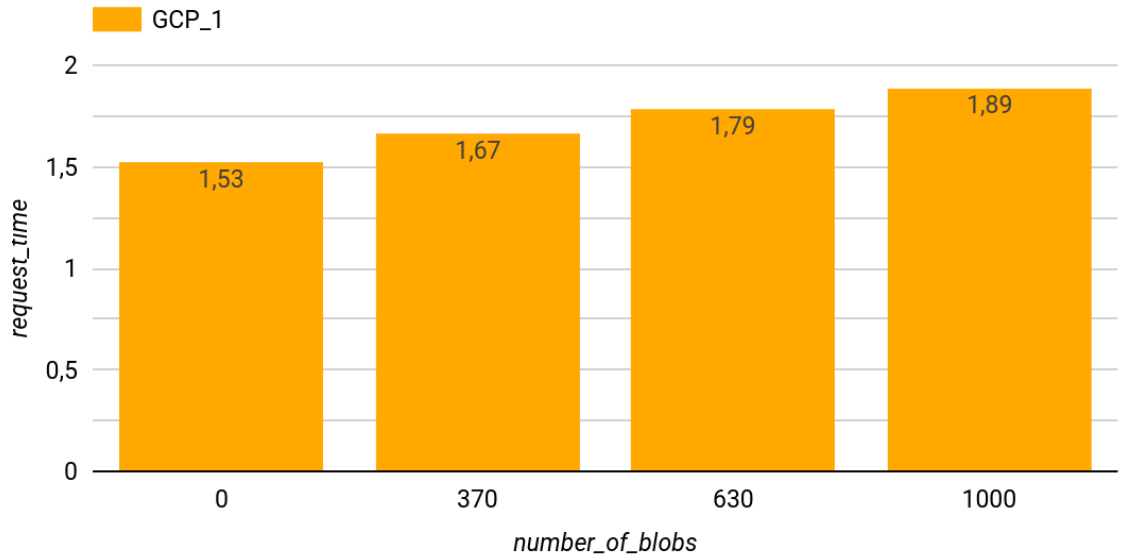

Рисунок 19 – Залежність середнього часу пошуку від кількості поверненних файлів

5.2.2 GCP2

На рисунку 20 зображенно залежність середнього часу вставки файлу від його типу.

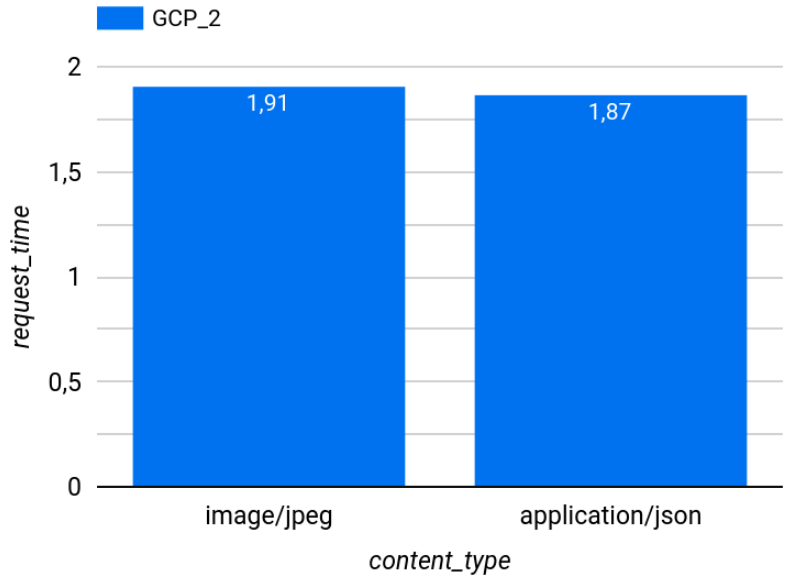

Рисунок 20 – Залежність середнього часу вставки файла від типу

На рисунку 21 зображенно залежність середнього часу вставки файлу від його розміру.

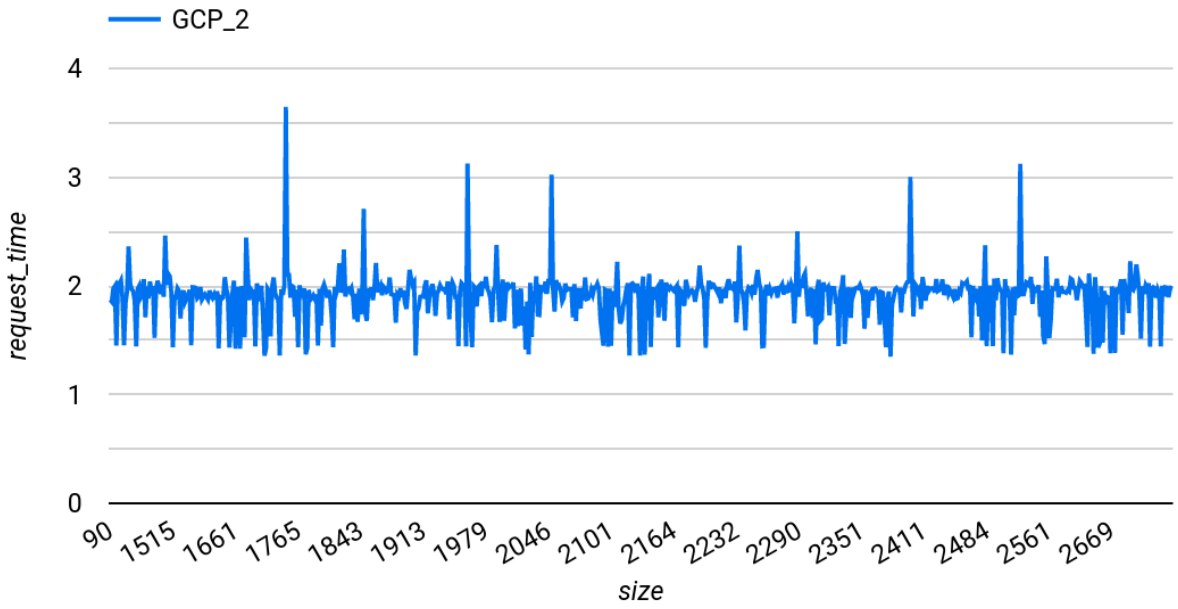

Рисунок 21 – Залежність середнього часу вставки файла від розміру

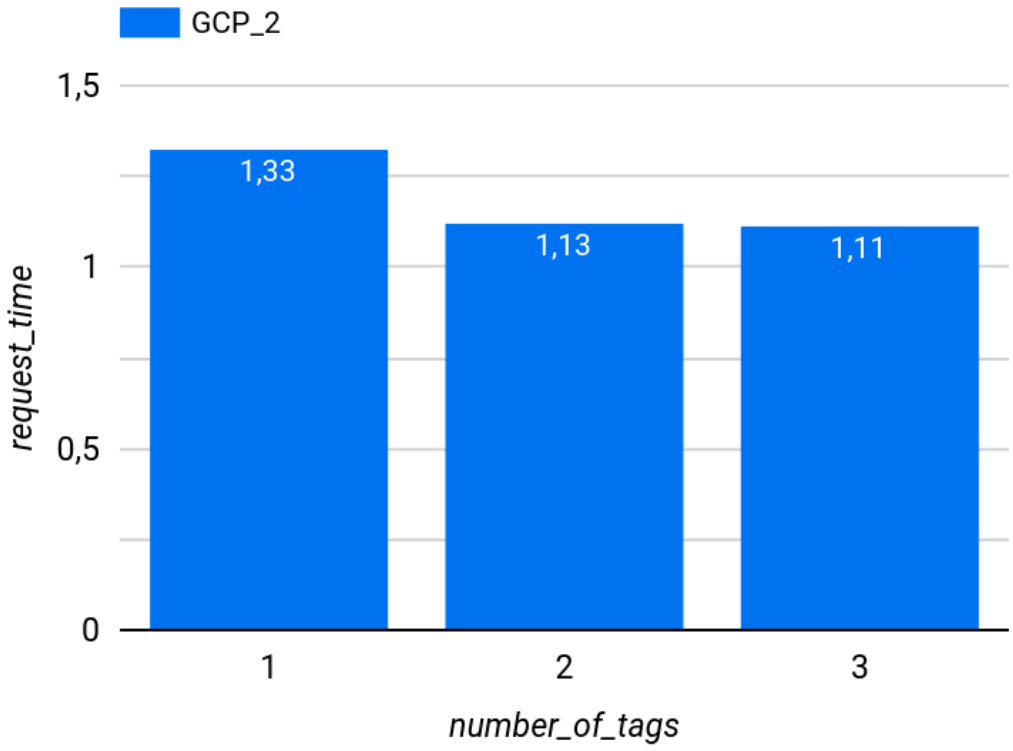

Рисунок 22 – Залежність середнього часу пошуку від кількості параметрів

На рисунку 22 та 23 зображенно залежність середнього часу пошуку від кількості параметрів та кількості знайденних файлів.

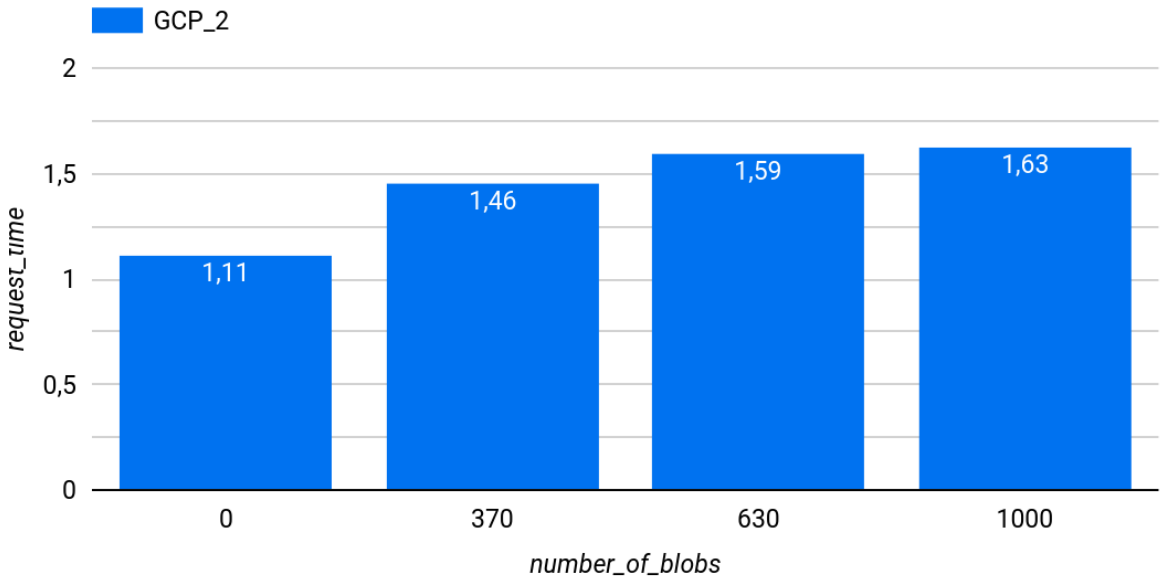

Рисунок 23 – Залежність середнього часу пошуку від кількості поверненних файлів

5.2.3 GCP3

На рисунку 24 зображенно залежність середнього часу вставки файлу від його типу.

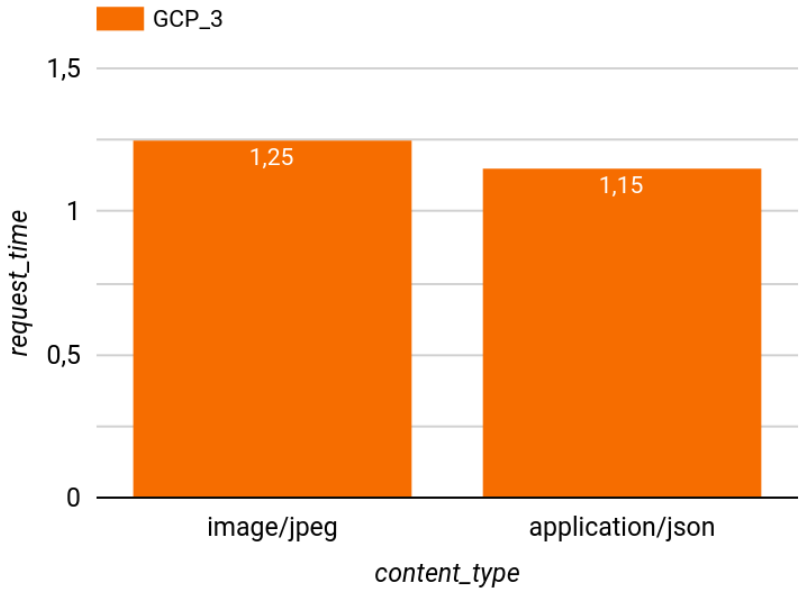

Рисунок 24 – Залежність середнього часу вставки файла від типу

На рисунку 25 зображенно залежність середнього часу вставки файлу від його розміру.

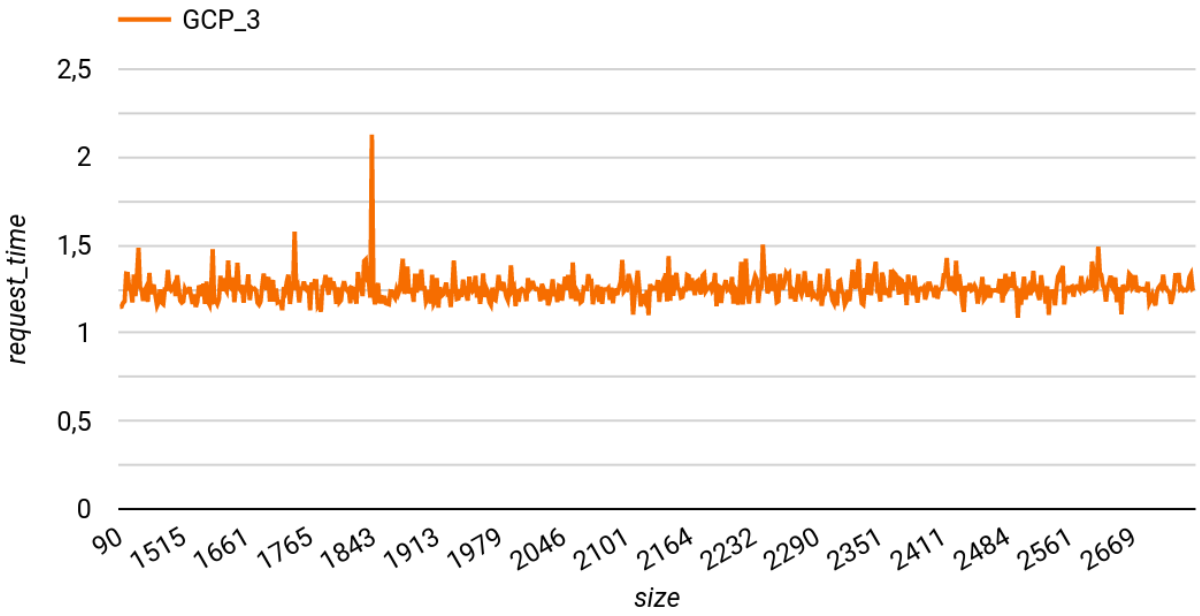

Рисунок 25 – Залежність середнього часу вставки файла від розміру

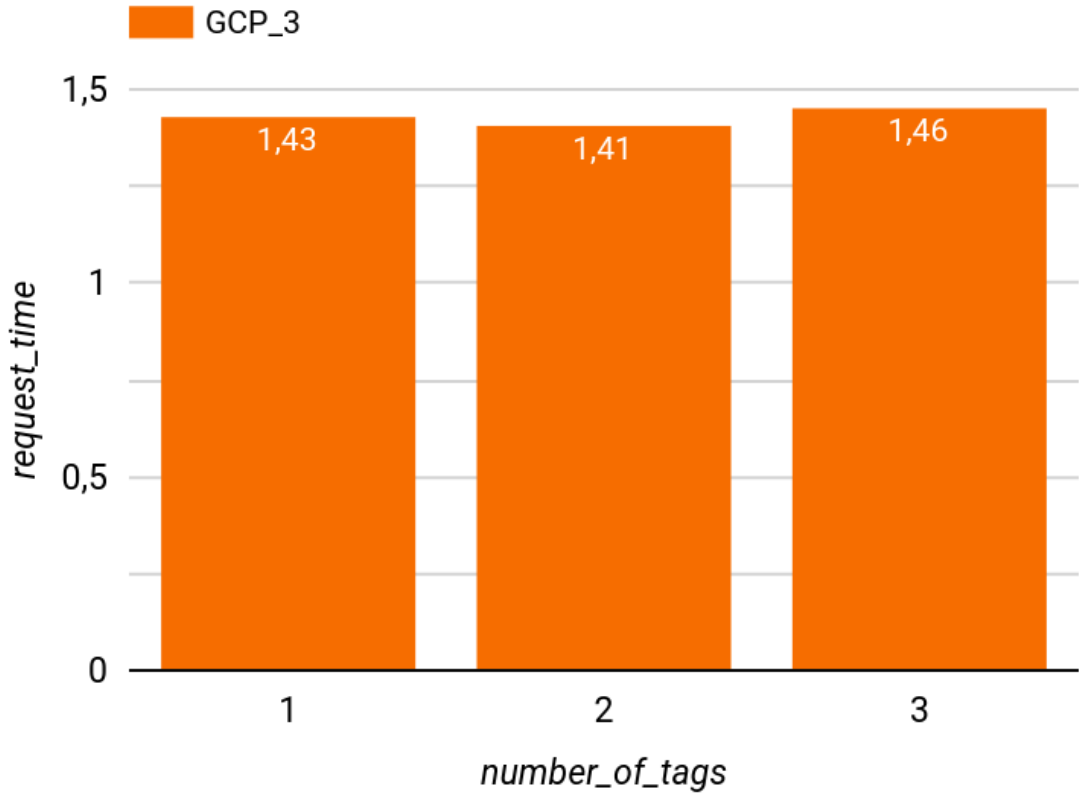

Рисунок 26 – Залежність середнього часу пошуку від кількості параметрів

На рисунку 26 та 27 зображенно залежність середнього часу пошуку від кількості параметрів та кількості знайденних файлів.

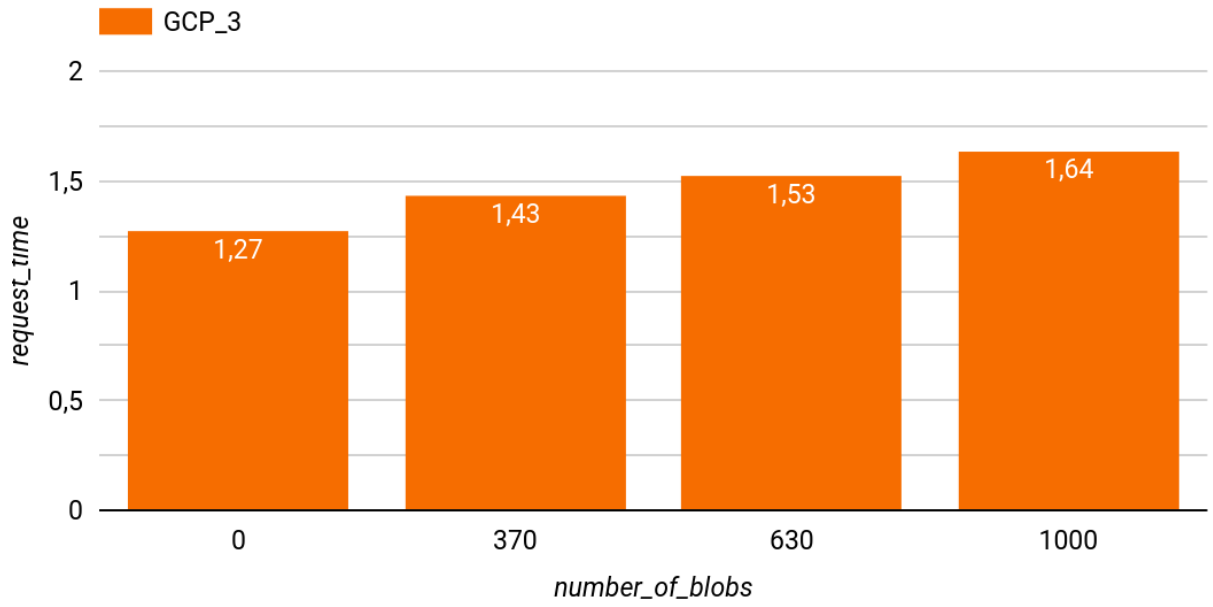

Рисунок 27 – Залежність середнього часу пошуку від кількості поверненних файлів

5.2.4 AWS1

На рисунку 28 зображенно залежність середнього часу вставки файлу від його типу.

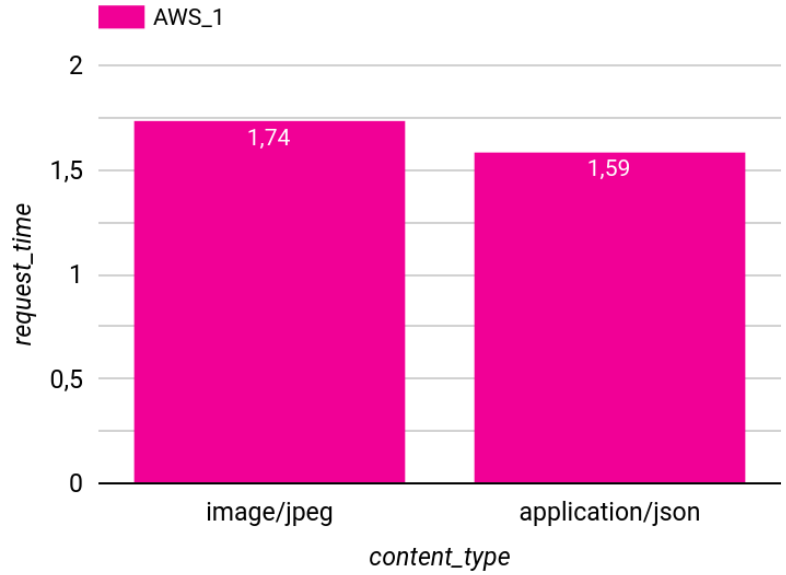

Рисунок 28 – Залежність середнього часу вставки файла від типу

На рисунку 29 зображенно залежність середнього часу вставки файлу від його розміру.

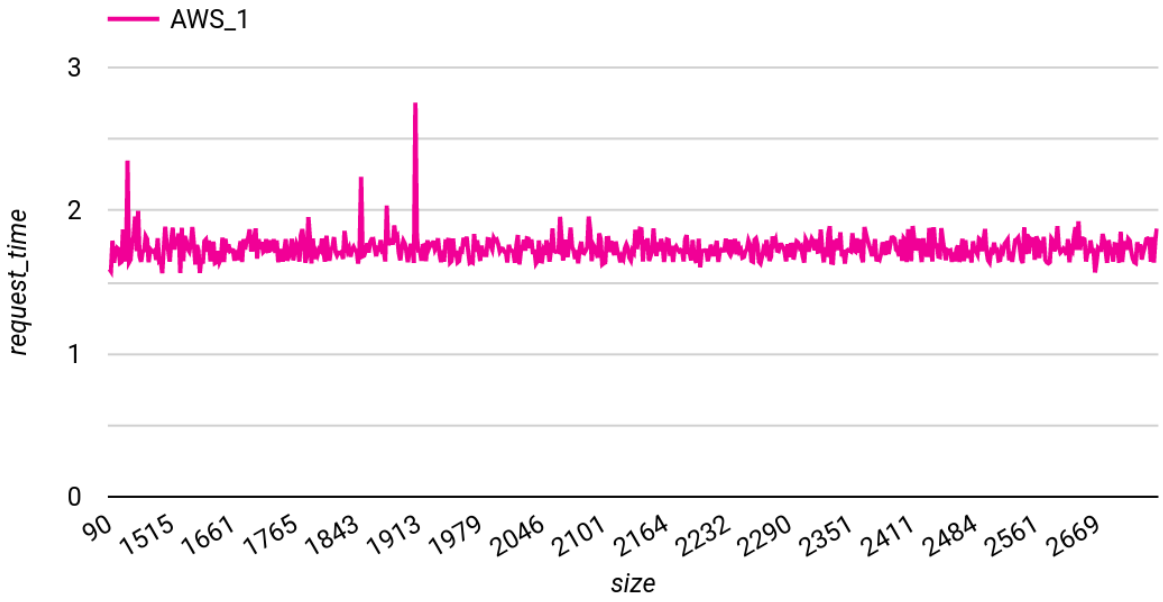

Рисунок 29 – Залежність середнього часу вставки файла від розміру

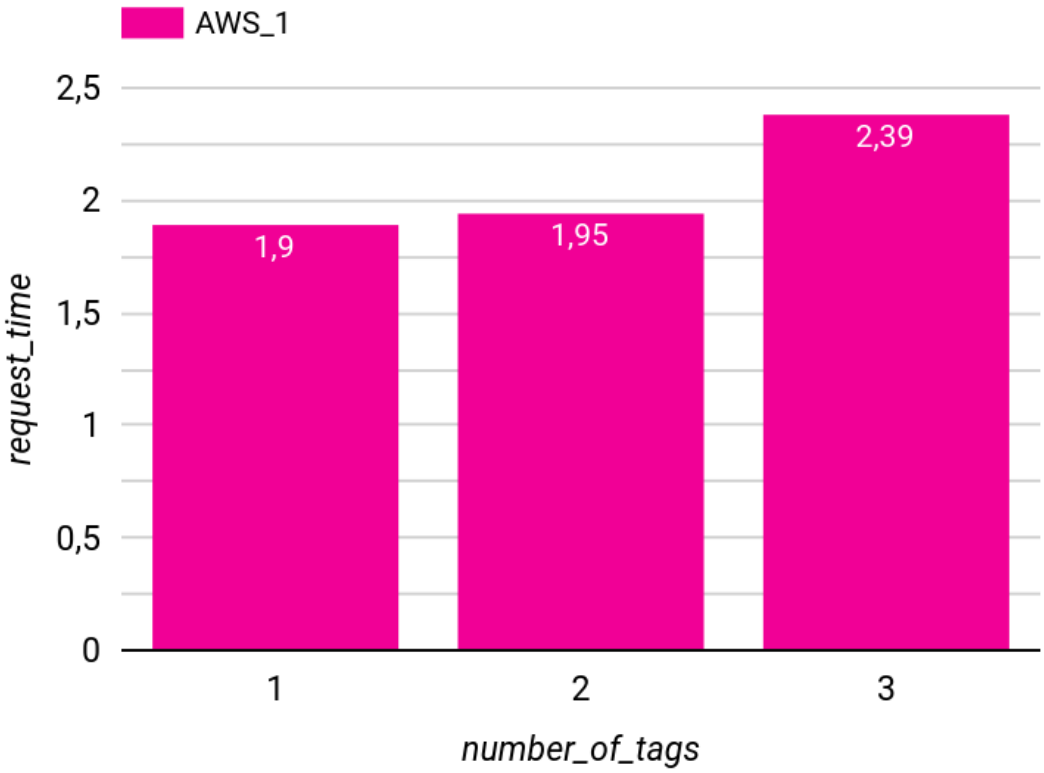

Рисунок 30 – Залежність середнього часу пошуку від кількості параметрів

На рисунку 30 та 31 зображенно залежність середнього часу пошуку від кількості параметрів та кількості знайденних файлів.

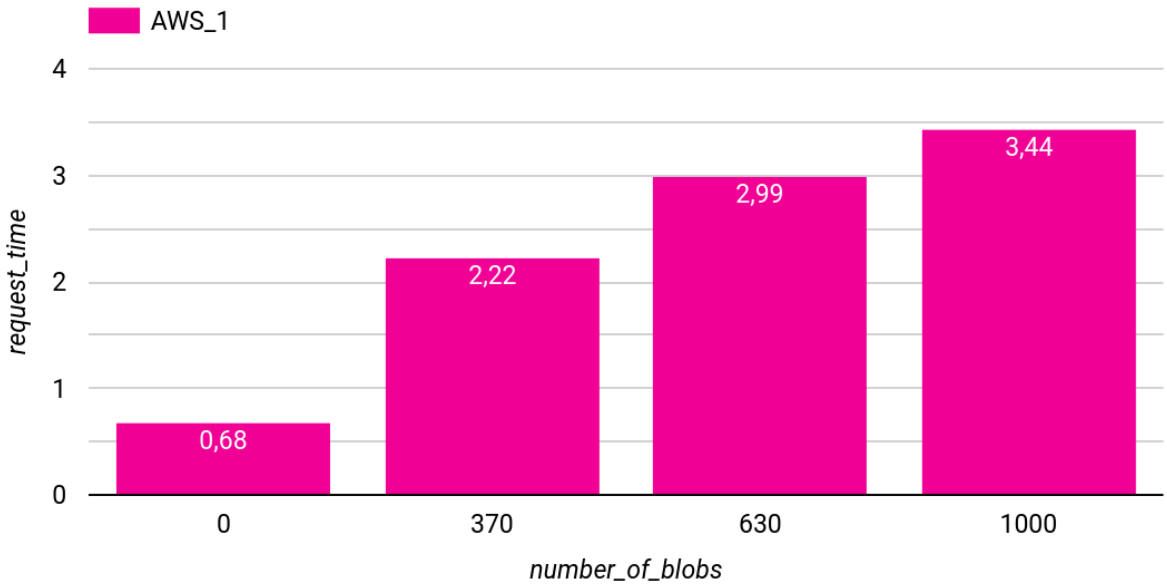

Рисунок 31 – Залежність середнього часу пошуку від кількості поверненних файлів

5.2.5 AWS2

На рисунку 32 зображенно залежність середнього часу вставки файлу від його типу.

Рисунок 32 – Залежність середнього часу вставки файла від типу

На рисунку 33 зображенно залежність середнього часу вставки файлу від його розміру.

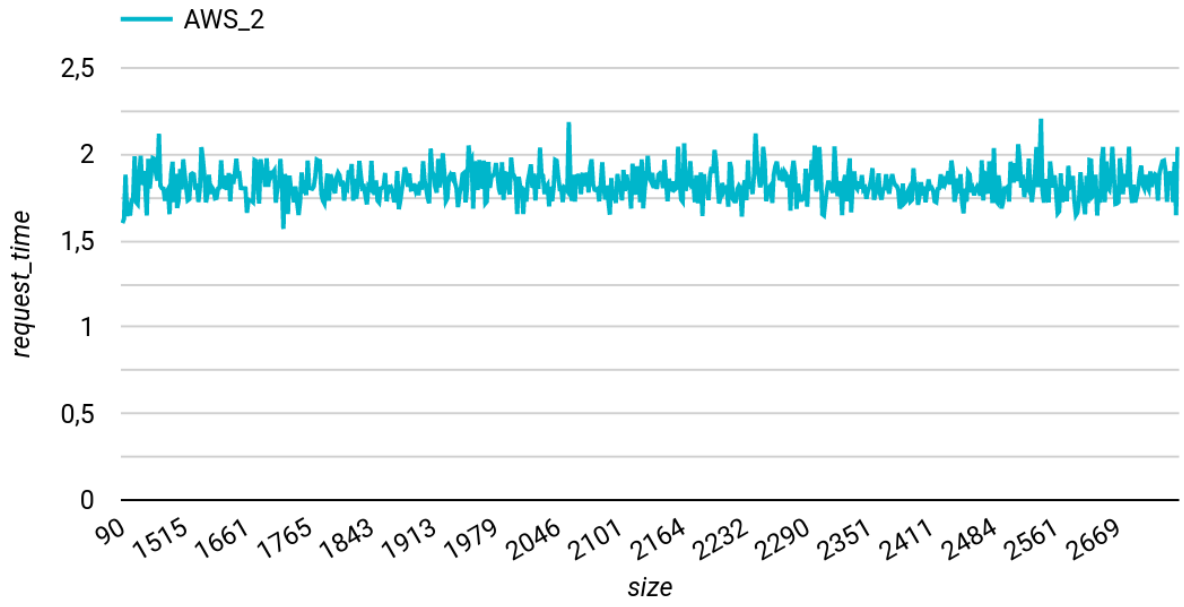

Рисунок 33 – Залежність середнього часу вставки файла від розміру

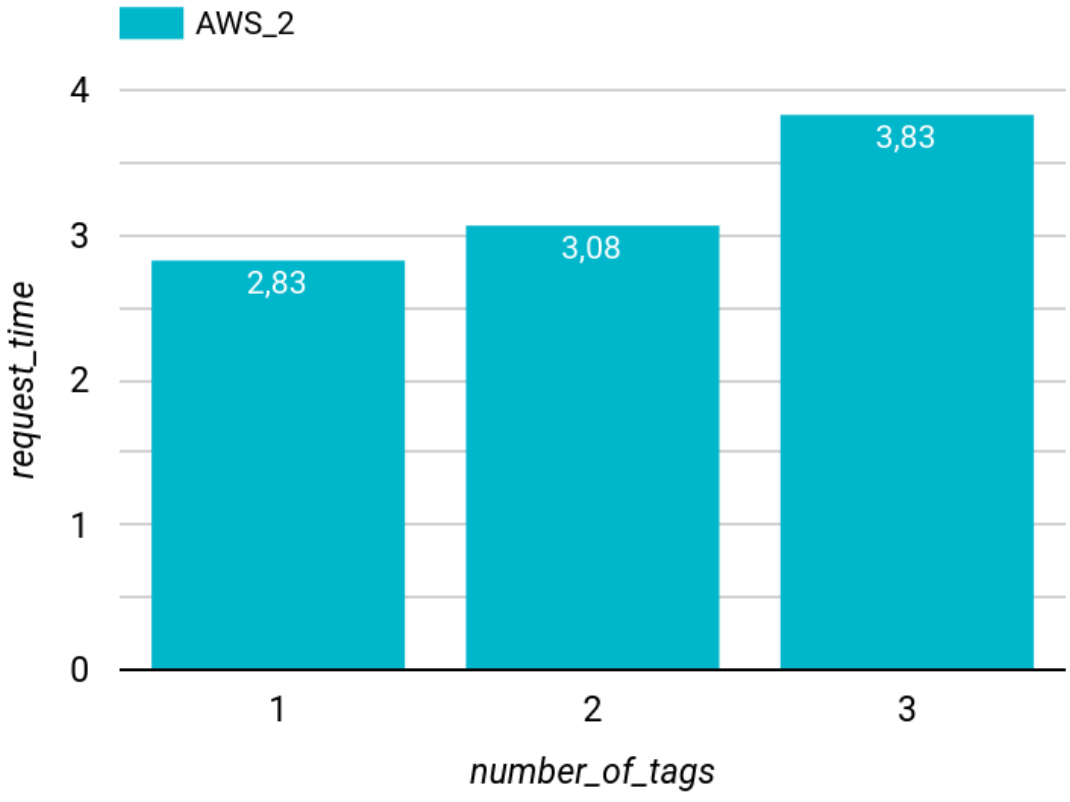

Рисунок 34 – Залежність середнього часу пошуку від кількості параметрів

На рисунку 30 та 31 зображенно залежність середнього часу пошуку від кількості параметрів та кількості знайденних файлів.

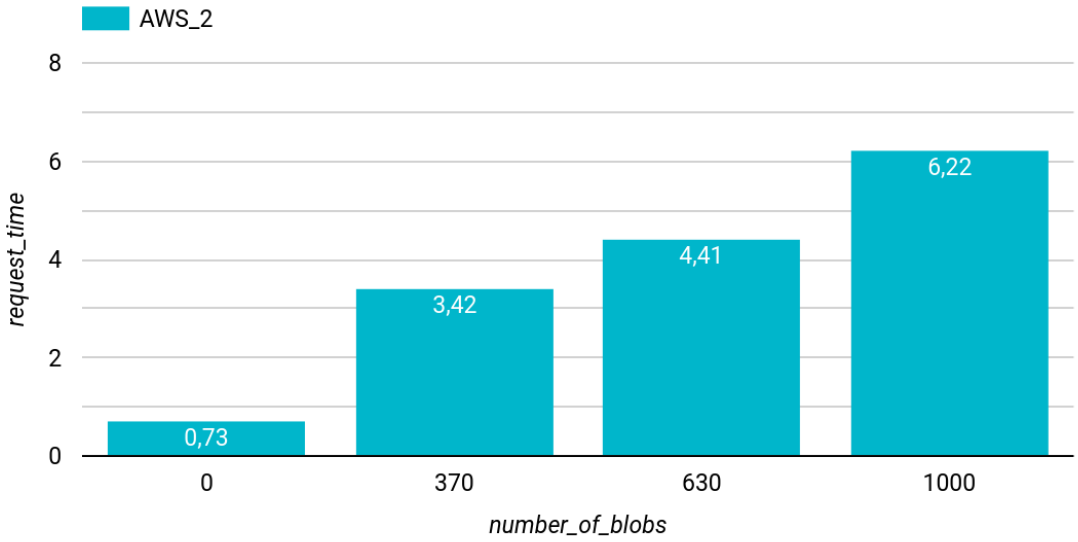

Рисунок 35 – Залежність середнього часу пошуку від кількості поверненних файлів

5.3 Результати розрахунків вартості

Для розрахунків було використанно Google Pricing Calcluator та AWS Pricing Calculator.

5.3.1 GCP

На рисунку 36 зображенно розрахунки вартості деплою GCP1 на оди місяць.

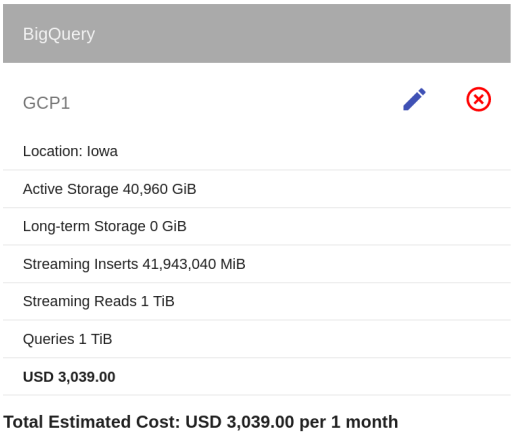

Рисунок 36 – Вартість деплою GCP1

На рисунку 37 зображенно розрахунки вартості деплою GCP2 на оди місяць.

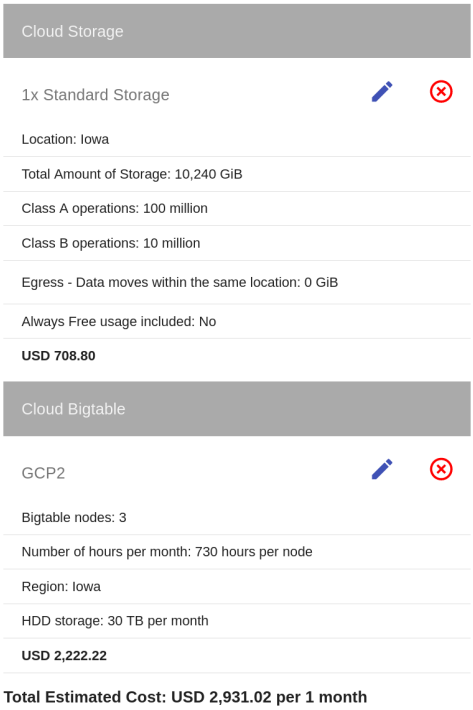

Рисунок 37 – Вартість деплою GCP2

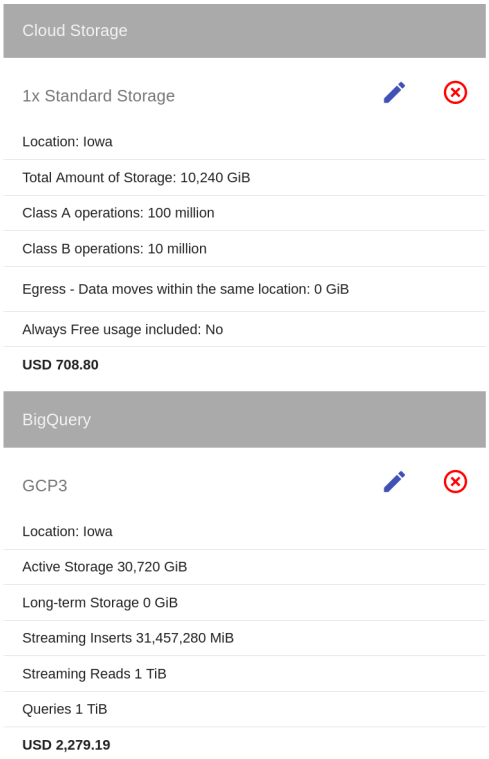

Рисунок 38 – Вартість деплою GCP3

На рисунку 38 зображенно розрахунки вартості деплою GCP3 на оди місяць.

5.3.2 AWS

На рисунку 39 зображенно розрахунки вартості деплою AWS1 на оди місяць.

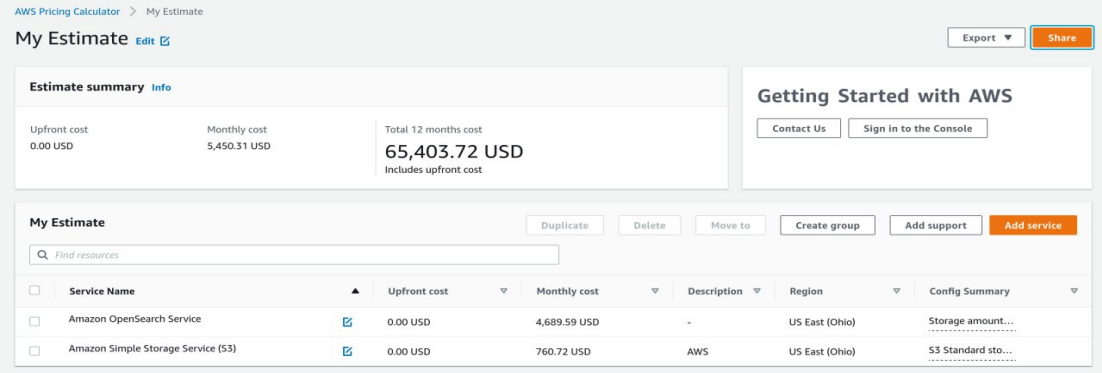

Рисунок 39 – Вартість деплою AWS1

На рисунку 40 зображенно розрахунки вартості деплою AWS2 на оди місяць.

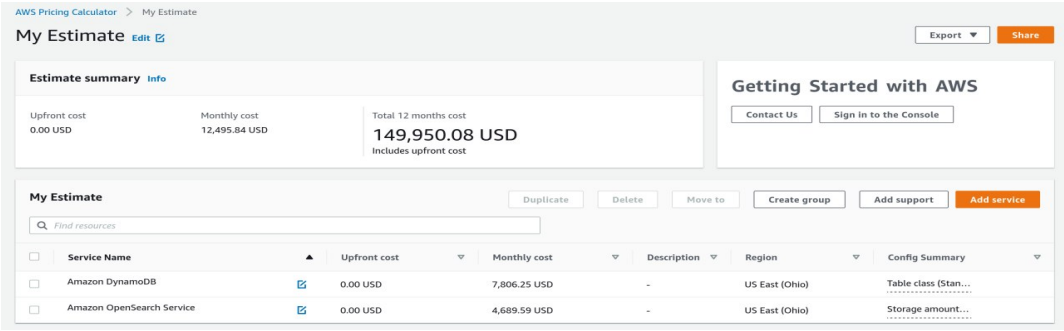

Рисунок 40 – Вартість деплою AWS2

5.4 Порівняння деплоїв GCP

5.4.1 Швидкість

Спочатку порівняємо швидкість завантаження файлів. На рисунку 41 зображенно діаграмму залужності середнього часу обробки запиту від типу файлу. На рисунку 42 зображенно графік середнього часу обробки запиту в залежності від розміру файлу.

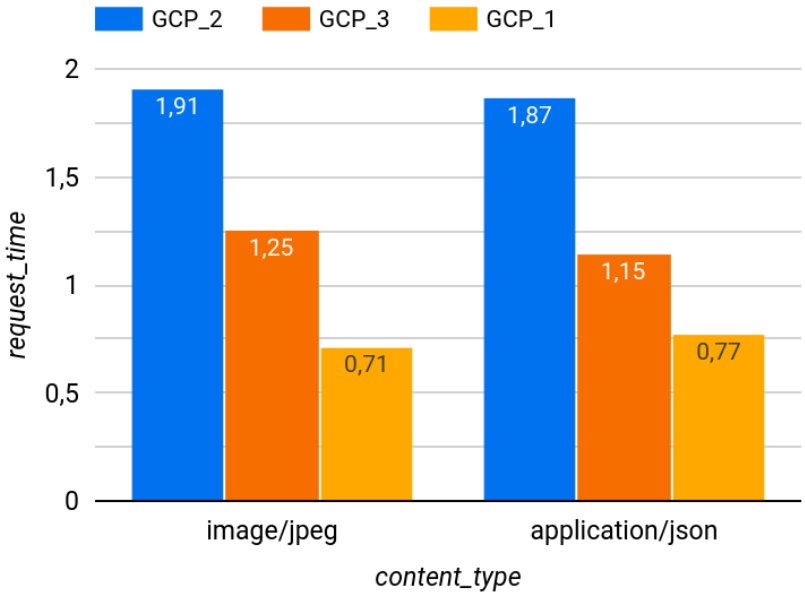

Рисунок 41 – Залежність середнього часу вставки файла від типу для всіх GCP

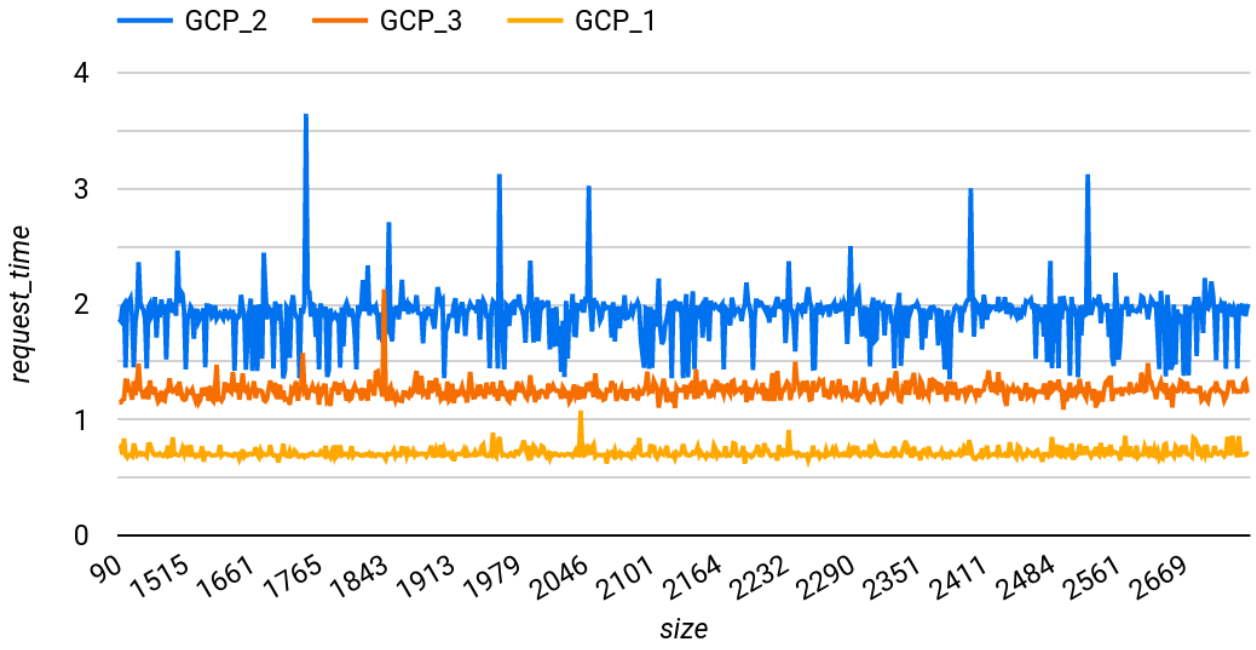

Рисунок 42 – Залежність середнього часу вставки файла від розміру для всіх GCP

З графіків можна зробити висновок що найшвидша архітектура це GCP1. Далі йде GCP3. І гірше за всіх справляється GCP2.

Скоріш за все GCP1 швидше через те що в цій архітектурі використовується тільки BigQuery. Тобто потрібно зробити лише один мережевий запит для

додання файлу, у той час як інші дві архітектури роблять по 2 запити. GCP3 швидше ніж GCP2 через оптимізацію BigQuery для потокової вставки.

Тепер роздивимося швидкість пошуку за метаданими (тегами). На рисунку 43 зображенно діаграмму залежності середьного часу обробки запиту від кількості тегів за якими відбувється пошук.

самое дорогое

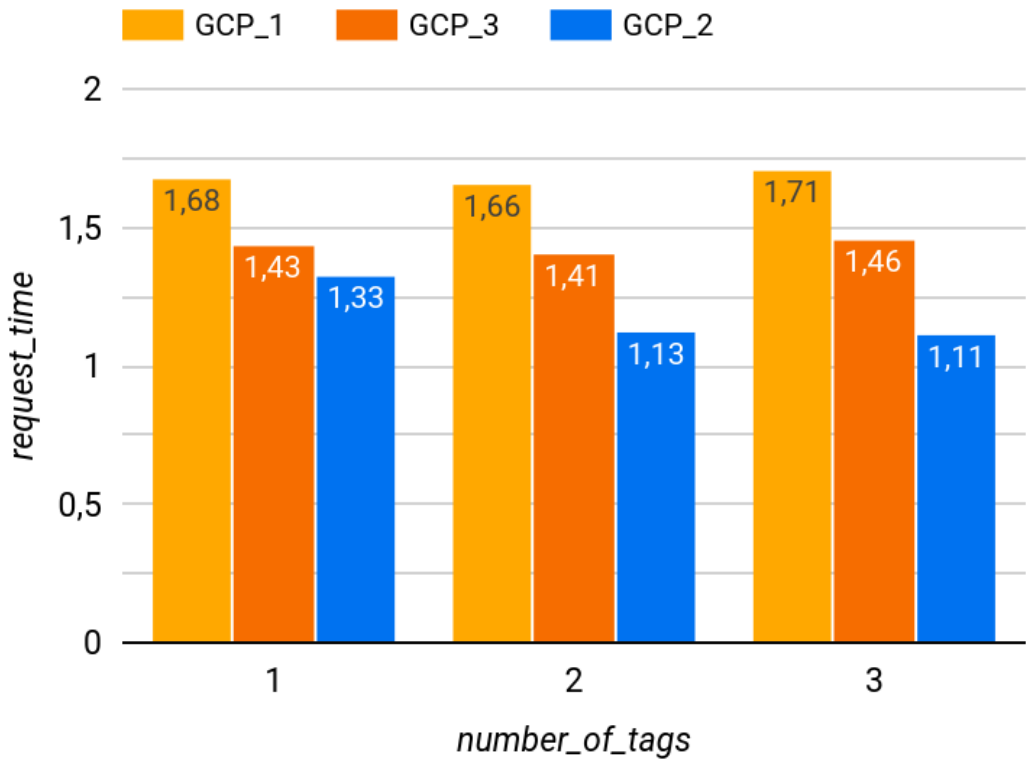

Рисунок 43 – Середній час пошуку файлів від кількості тегів для всіх GCP

На рисунку 44 зображенно діаграмму залежності середьного часу обробки запиту від кількості знайдених файлів.

Як бачимо ситуація повністю обратна на відміну від швидкості вставки. GCP1 працює довше за все. Насправді це є несподіваним результатом, оскільки BigQuery — колонкове сховище даних. Пошуковий запит GCP1 та GCP3 зовсім не відрізняється тому варто було б очікувати приблизно однакового часу. Проте експеримент виявив що запити до таблиці BigQuery з додатковою файловою колонкою довші в середьному на 17%.

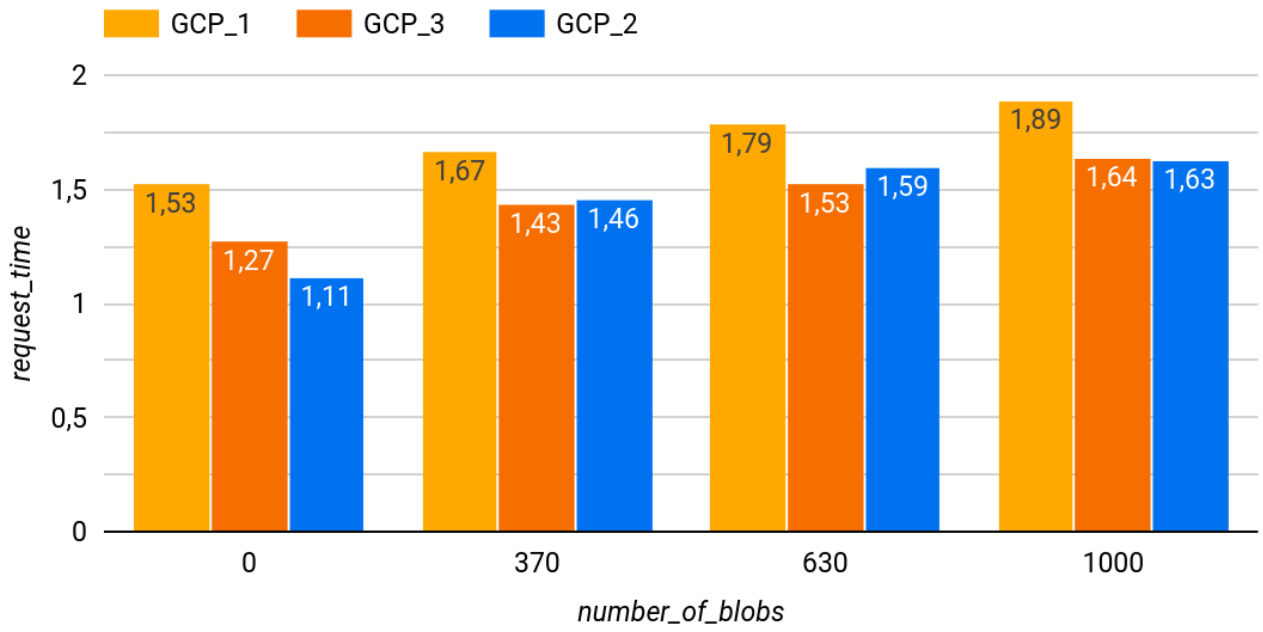

Рисунок 44 – Середній час пошуку файлів від кількості файлів для всіх GCP

Якщо дивитися тільки на діаграмму залежності часу від знайдених файлів, то можна сказати що між GCP2 та GCP3 майже не має різниці. GCP1 знов показує найгірший час.

5.4.2 Вартість

У таблиці 1 приведенно вартість архітектур.

Таблиця 1 – Вартість архітектур GCP

	GCP1	GCP2	GCP3
Вартість $	3039.00	2931.02	2279.19

За табліци становиться зрозуміло, що GCP1 – найдорожче рішення серед усіх. GCP2 коштує трохи меньше, а от GCP3 дозволить гарно зекономити порівнянно з іншими двома архітектурами.

5.4.3 Вискновок

Зробимо виснвоки щодо представленних архітектур.

GCP1 потрібно використовувати коли нам важлива швидкість додавання нових даних, наприклад коли в нас велика мережа IoT пристроїв. Але варто

пам’ятати, що це найдорожче рішення і швидкість пошуку залишає бажати кращого.

GCP2 вважаю не варто використовувати, оскільки для Data Lake важливіше швидкість додавання нової інформації, а він показав найгірший результат та коштує майже як GCP1.

GCP3 це золота середина між GCP1 та GCP2, який ще й коштує найменше. Також він має перевагу перед GCP1 з позиції DevOps. Оскільки дані зберігаються в Cloud Storage це дозволяє налаштувати версіонування та створення бекапів незалежно від метаданих.

5.5 Порівняння деплоїв AWS

5.5.1 Швидкість

Спочатку порівняємо швидкість завантаження файлів. На рисунку 45 зображенно діаграмму залужності середнього часу обробки запиту від типу файлу.

Рисунок 45 – Залежність середнього часу вставки файла від типу для всіх AWS

На рисунку 46 зображенно графік середнього часу обробки запиту в залежності від розміру файлу.

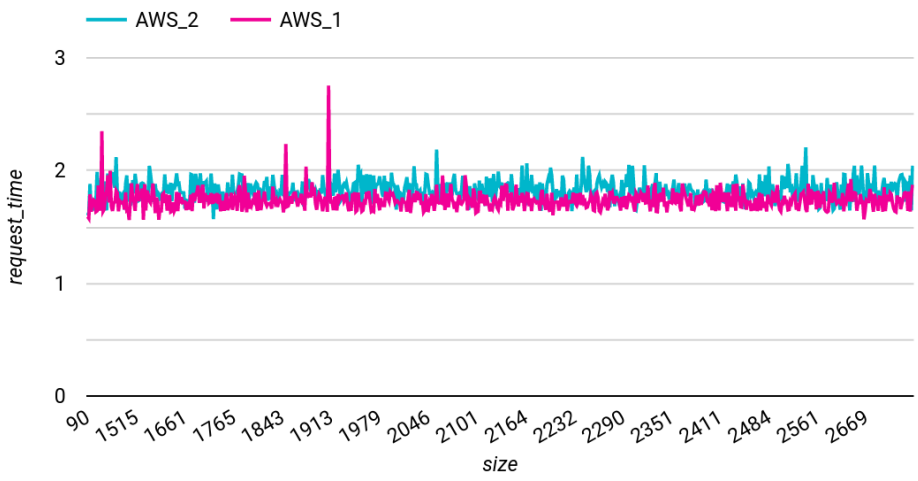

Рисунок 46 – Залежність середнього часу вставки файла від розміру для всіх AWS

Дивлячись на діаграми можна зробити висновок, що між ціма двома архітектурами існує дуже невилика різниця у кілька сотих секунд. AWS1 показує трохи кращі результати.

Роздивимося швидкість пошуку за метаданими (тегами). На рисунку 47 зображенно діаграмму залежності середьного часу обробки запиту від кількості тегів за якими відбувється пошук.

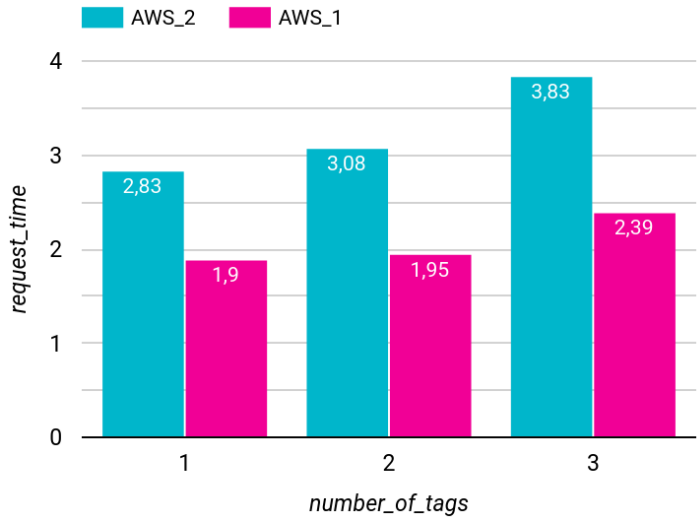

Рисунок 47 – Середній час пошуку файлів від кількості тегів для всіх AWS

На рисунку 48 зображенно діаграмму залежності середьного часу обробки запиту від кількості знайдених файлів.

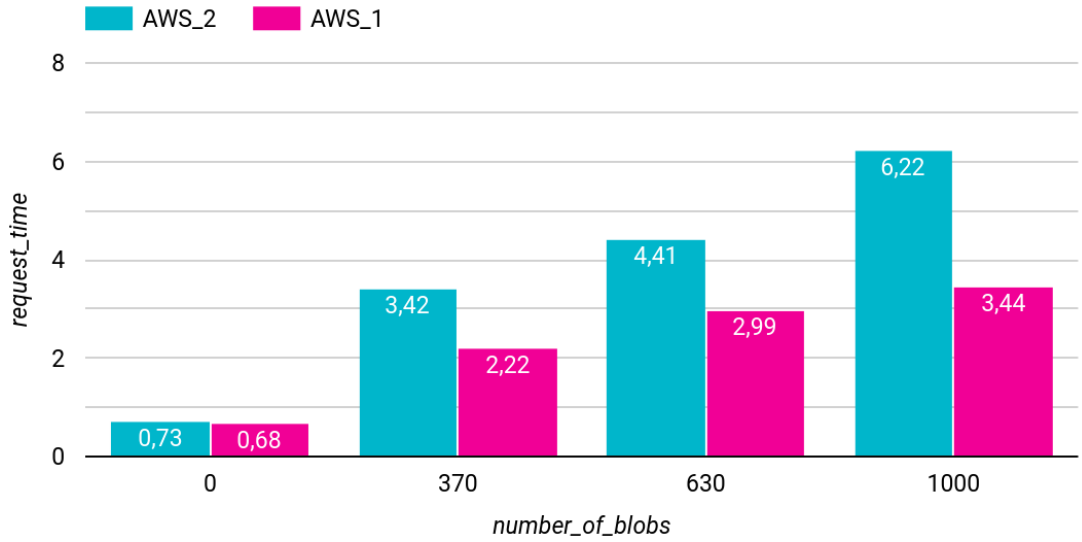

Рисунок 48 – Середній час пошуку файлів від кількості файлів для всіх AWS

З діаграм можна зробити висновок що AWS1 в середньому швидве на 50%. Чим більше файлів треба повернути тим більше розрив між ними.

5.5.2 Вартість

У таблиці 2 приведенно вартість архітектур.

Таблиця 2 – Вартість архітектур AWS

	AWS1	AWS2
Вартість $	5450.31	12495.84

5.5.3 Вискновок

Зробимо виснвоки щодо представленних архітектур. AWS1 мае невеликий виграш у швидкості вставки даних, значний виграш у швидкості пошуку та коштує більш ніж в 2 рази дешевше за AWS2.

5.6 Порівняння найкращих архітектур

У провайдера GCP найкращою архітектурою виявився GCP3, у AWS — AWS1. Порівняймо їх за тими ж параметрами що й інші аархітектури.

5.6.1 Швидкість

На рисунку 49 зображенно діаграмму залужності середнього часу обробки запиту від типу файлу.

На рисунку 50 зображенно графік середнього часу обробки запиту в залежності від розміру файлу.

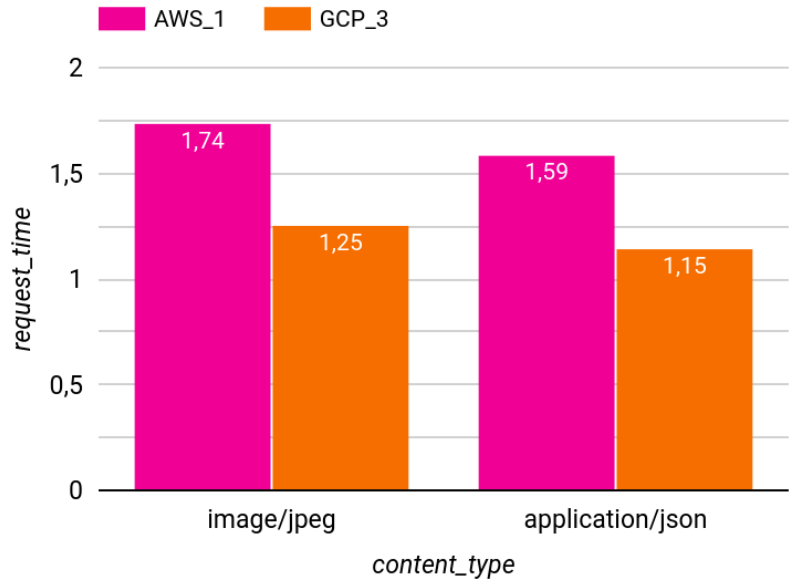

Рисунок 49 – Залежність середнього часу вставки файла від типу для AWS1 та GCP3

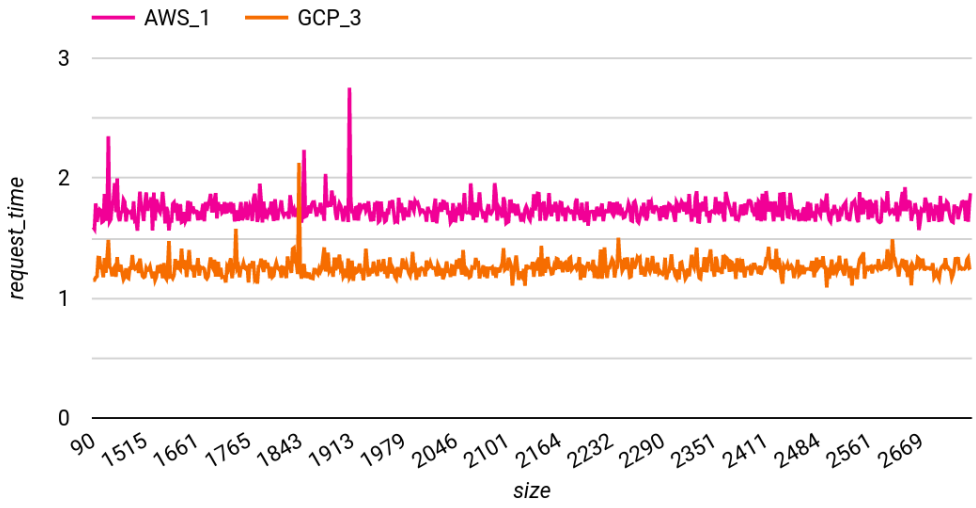

Рисунок 50 – Залежність середнього часу вставки файла від розміру для AWS1 та GCP3

На рисунку 51 зображенно діаграмму залежності середьного часу обробки запиту від кількості тегів за якими відбувється пошук.

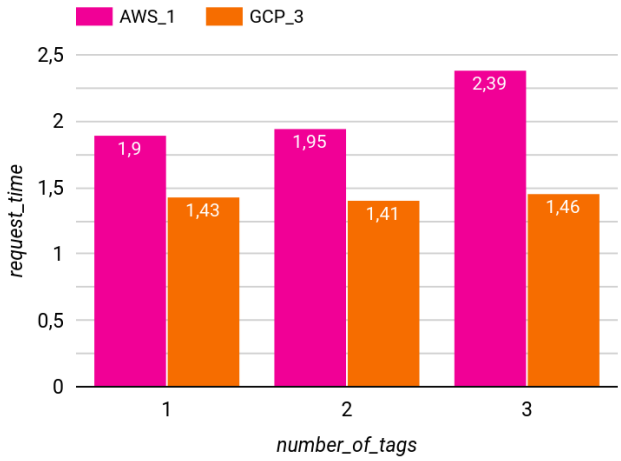

Рисунок 51 – Середній час пошуку файлів від кількості тегів для AWS1 та GCP3

На рисунку 48 зображенно діаграмму залежності середьного часу обробки запиту від кількості знайдених файлів.

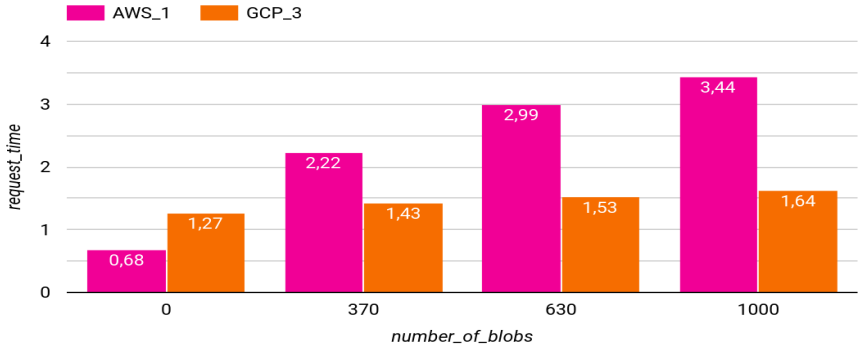

Рисунок 52 – Середній час пошуку файлів від кількості файлів для AWS1 та GCP3

5.6.2 Вартість

У таблиці 3 приведенно вартість архітектур.

Таблиця 3 – Вартість архітектур AWS1 та GCP3

	AWS1	GCP3
Вартість $	5450.31	2279.19

5.6.3 Вискновок

За всіма параметрами, окрім середьного часу пошуку файлів коли повертається 0 файлів, GCP3 кращє ніж AWS1.

ВИСНОВКИ

Під час практики було розроблено систему автоматичного розгортання data lake на потужностях хмарних провайдерів AWS, GCP. Також були проведені експерименти для порівняння швидкості за різних варіантів архітектури. Для експерименту було взято 2 набори даних. Перший набір – текстові json дані з iot пристроїв (температурного датчика). Розмір всіх файлів цього набору становить 95-96 байт. Другий набір - зображення формату jpg різного характеру та розміру. Під час експерименту проводились вимірювання швидкості другого етапу завантаження файлу та швидкості пошуку за метаданими. Всі послуги були створені в однакових регіонах (наскільки це було можливо). Результати вимірів завантажувалися в Google BigQuery, після чого проводився аналіз у Google Data Studio. В результаті було отримано чотири графіки для кожної архітектури: середня швидкість виконання другого етапу в залежності від типу файлу, середня швидкість виконання другого етапу в залежності від розміру файлу, середня швидкість пошуку в залежності від кількості тегів, середня швидкість пошуку в залежності від кількости знайдених файлів. Усі значення чаус на графіках вказано в секундах. Розміри файлів на графіку вказано у байтах. Також було теоретично розрахована вартість кожної архітектури для порівняння. При розрахунках вважалося, що кожного місяця в Data Lake завантажується 10 Tb даних та 30 Tb метаданих та шукається файлів на 1 TB

Краще за все себе показала архітектура GCP3 (BigQuery + Cloud Storage). У хмарного провайдера AWS найкращою виявилася архітекутра AWS1 (S3 + OpenSearch).

В якості пропозиції щодо подальших досліджень є додавання нових архітектур до провайдерів AWS та GCP (наприклад Dataproc Metastore у якості хранилища метаданих на GCP), додавання нових провайдерів (Azure), та тестування та розрахунок вартості рішення на власному сервері.

ПЕРЕЛІК ПОСИЛАНЬ

1. Arsenov, A., Ruban, I., Smelyakov, K., Chupryna, A. Evolution of convolutional neural network architecture in image classification problems - CEUR Workshop Proceedings, 2018, 2318, стр. 35–45

2. Новиков Б. Основы технологий баз данных / Б. Новиков, Е. Горшкова., 2022. – 233 с. – (Litres).

3. Горелік А. The Enterprise Big Data Lake: Delivering the Promise of Big Data and Data Science / Алекс Горелік.. – 224 с. – (.).

4. Уайт Т. Hadoop: The Definitive Guide / Том Уайт., 2009. – (O'Reilly Media Inc.).

5. Чамберс Б. Spark: The Definitive Guide / Б. Чамберс, М. Захарія.. – (O'Reilly Media, Inc.).

6. Дайонг Д. Apache Hive Essentials: Essential techniques to help you process, and get unique insights from, big data / Ду Дайонг., 2018. – 210 с. – (Packt Publishing).

7. Лакшманан В. Google BigQuery: The Definitive Guide: Data Warehousing, Analytics, and Machine Learning at Scale / В. Лакшманан, Д. Тигани.. – 522 с. – (O'Reilly Media, Inc.).

8. Херст Л. Hands On With Google Data Studio: A Data Citizen's Survival Guide / Лі Херст., 2020. – 432 с. – (Wiley).

8. Куруппу Д. gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes / Данеш Куруппу., 2020. – 204 с. – (O'Reilly Media, Inc.).

10. Ворон Ф. Building Data Science Applications with FastAPI: Develop, manage, and deploy efficient machine learning applications with Python / Франкольс Ворон., 2021. – 426 с. – (Packt Publishing).

11. frankie567. Overview - FastAPI Users [Електронний ресурс] / frankie567 – Режим доступу до ресурсу: https://fastapi-users.github.io/fastapi-users/configuration/overview/.

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

1. Arsenov, A., Ruban, I., Smelyakov, K., Chupryna, A. Evolution of convolutional neural network architecture in image classification problems - CEUR Workshop Proceedings, 2018, 2318, стр. 35–45