МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет комп'ютерних наук та технологій
Кафедра комп’ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ

Завідувач кафедри

Аліна САВЧЕНКО

“__”_____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

“МАГІСТР”

ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ “ІНФОРМАЦІЙНІ УПРАВЛЯЮЧІ СИСТЕМИ ТА ТЕХНОЛОГІЇ”

Тема: “Архітектура системи сховища даних на основі хмарних технологій”

Виконавець: Штипуляк Артур Миколайович

Керівник: к.т.н., доцент Колісник Олена Василівна

Нормоконтролер: Ігор РАЙЧЕВ
(підпис)

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій

Кафедра комп’ютерних інформаційних технологій

Освітній ступінь: “Магістр”

Галузь знань, спеціальність, освітньо-професійна програма:

12 “Інформаційні технології, 122 “Комп’ютерні науки”, “Інформаційні управляючі системи та технології” .

ЗАТВЕРДЖУЮ

Завідувач кафедри

Аліна САВЧЕНКО

“____” _____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи студента

Штипуляка Артура Миколайовича

(прізвище, ім’я, по батькові)

1. **Тема роботи:** «Архітектура системи сховища даних на основі хмарних технологій» затверджена наказом ректора №1976/ст. від 29.09.2023.
2. **Термін виконання роботи:** 02.10.2023 – 31.12.2023.
3. **Вихідні дані до роботи:** теоретичні відомості про системи сховищ даних з використанням хмарних технологій, реалізація системи для зберігання даних на основі хмарних технологій.
4. **Зміст пояснювальної записки (перелік питань, що підлягають розробці):** аналіз систем хмарних сховищ даних, аналіз використаних технологій для розробки інформаційної системи, розробка програмного рішення та налаштовування інфраструктури системи сховища даних з використанням хмарних технологій.
5. **Перелік обов’язкового графічного матеріалу:** діаграми, слайди презентації в MS PowerPoint.

6. **Календарний план-графік**

№ з/п	*Завдання*	*Термін виконання*	*Підпис керівника*
1.	Вивчення області дослідження та аналіз її характеристик	02.10.2023р. – 10.10.2023р.	
2	Створення формалізованого опису систем хмарних сховищ даних	11.10.2023р. – 14.10.2023р.	
3	Створення моделі інформаційної системи та визначення її ключових вимог	15.10.2023р. – 21.10.2023р.	
4	Аналіз можливостей використання хмарних технологій для реалізації системи	22.10.2023р. – 24.10.2023р.	
5	Програмна реалізація системи та корегування його формалізованої моделі	25.10.2023р. – 30.11.2023р.	
6	Створення автоматизованих тестів для реалізованої інформаційної системи	01.12.2023р. – 04.12.2023р.	
7	Створення пояснювальної записки та супровідних інформаційних матеріалів	05.12.2022р. – 15.12.2022р.	

Дата видачі завдання: 02.10.2023 р.

Керівник дипломного проекту _____ Олена КОЛІСНИК
(підпис керівника)

Завдання прийняв до виконання _____ Артур ШТИПУЛЯК
(підпис випускника)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Архітектура системи сховища даних на основі хмарних технологій» представлена на 95 сторінках, містить 37 рисунків та 2 додатки. Список бібліографічних посилань складається з 15 найменувань.

Мета дипломного проекту: дослідження систем сховищ даних, дослідження їх архітектура, розробка системи для зберігання файлових даних з використанням хмарних технологій.

Об'єкт дослідження: система хмарного сховища даних для збереження файлових даних.

Предмет дослідження: збереження файлових даних в сховищі даних у хмарі.

Метод дослідження: аналіз принципів побудови та роботи систем сховищ даних у хмарі.

Результат проекту: розроблена інформаційна система для збереження файлів у хмарі з використанням мови програмування Java, Spring Framework та хмарних сервісів від провайдера Amazon Web Services.

Ключові слова: ДОДАТОК ДЛЯ ЗБЕРЕЖЕННЯ ФАЙЛОВИХ ДАНИХ, БАЗА ДАНИХ, ІНТЕГРАЦІЯ СИСТЕМИ, WEB COMPONENTS, JAVA, SPRING FRAMEWORK, SPRING MVC, AMAZON WEB SERVICES

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API – Прикладний програмний інтерфейс

MVC – Архітектурний шаблон Модель-Вид-Контролер

HTTP – Протокол передачі гіпертексту

URL – Єдиний вказівник на інтернет ресурс

REST – Передача репрезентативного стану

JSON – Текстовий формат обміну даними

IaaS – Інфраструктура як сервіс

PaaS – Платформа як сервіс

SaaS – Програма як сервіс

AWS – Провайдер хмарних ресурсів Amazon Web Services

EC2 – Інфраструктурна служба по оренді віртуальних серверів

VPC – Сервіс надання віртуальної приватної мережі

S3 – Об’єктне сховище

SQS – Асинхронна черга

RDS – Сервіс по оренді розподіленої реляційної бази даних

VM – Віртуальна машина

ACID – Атомарність, узгодженість, ізольованість, довговічність

ORM – Об’єкто-реляційна проєкція

БД – База даних

ВСТУП

В рамках кваліфікаційної роботи було досліджено та розроблено систему сховища даних для збереження файлових даних з використанням хмарних технологій.

Хмарні системи зберігання даних дозволяють користувачам отримувати доступ до своїх даних з будь-якого місця. Вони можуть завантажувати чи вивантажувати свої файли або ділитися ними з іншими через мережу Інтернет. Завдяки персональним додаткам, таким як Dropbox, iCloud, Google Drive, які використовуються багатьма користувачами для збереження власних файлів, уявлення про доступність з будь-якого місця даних кардинально змінилося

Реалізована система дозволяє зберігати файлові дані в хмарі з можливістю для користувача доступу до них та проведення змін над ними. Основною одиницею, якою можливо оперувати в рамках системи є файлові дані, доступ до яких здійснюється через REST API. Сховище користувача відповідає по логічній структурі файловій системі операційної системи, але файлові дані та інформація про ці дані зберігаються в різних місцях.

Для збереження фрагментів файлів використано хмарний сервіс AWS S3 від провайдера Amazon Web Services. Метаінформацію про файли зберігається в окрему базу даних.

Головною особливістю системи є розбиття монолітної системи на декілька окремих сервісів, які працюють в хмарі та використовують інші сервіси від провайдера хмарних ресурсів для коректної роботи. Це дає можливість при необхідності масштабувати кількість одночасно використовуваних екземплярів сервісу, а доступ до системи для клієнта здійснюється через єдиний шлюз. Також можна виділити механізми оптимізації роботи з файлами, як завантаження лише фрагменту файлу для збереження, чи переміщення файлу між директоріями лише з відповідними змінами в базі даних, а не у самому сховищі файлів.

Для роботи клієнту доступний різний функціонал, який можна використовувати для тих чи інших змін стану сховища даних користувача. Для прикладу доступно

завантаження в сховище, вивантаження файлу або його фрагменту, копіювання, переміщення, видалення, тегування. Також доступно відповідний функціонал для роботи з директоріями, де ці файли зберігаються.

Результатом виконання роботи є програмний продукт, який складається з кількох сервісів, кожен з яких відрізняється за основним функціоналом по роботі з файловими даними та їх метаданими, і, відповідно, призначенням в рамках реалізованої системи.

РОЗДІЛ 1
АНАЛІЗ СИСТЕМ СХОВИЩА ДАНИХ У ХМАРІ

Хмарні обчислення стали одним з найбільш використовуваними терміном в технічній сфері, вони є гарячою темою в останніх дослідженнях і застосуваннях. Нині Google, Microsoft, IBM, Amazon та деякі інші відомі компанії запропонували свої рішення для хмарних обчислень і визначили хмарні обчислення як одну з найважливіших стратегій у теперішньому та майбутньому.

Хмарні обчислення - це операційна модель, що забезпечує доступ на вимогу до обчислювальних ресурсів, таких як сервери, сховища, бази даних і додатки, шляхом спільного використання їх через Інтернет. Хмарні сервіси - це революційний спосіб надання бізнесу доступу до обчислювальних ресурсів, замість того, щоб інвестувати в дороге обладнання та інфраструктуру. Це дає компаніям гнучкість у масштабуванні та економічну ефективність внаслідок оптимального використання хмарних ресурсів. Хмарні технології стали рушійною силою для бізнесу та способів взаємодії людей, від онлайн-покупок до соціальних мереж чи національної безпеки; ця технологія продовжує відігравати ключову роль у трансформації майбутнього світогляду на обчислювальні технології. Хмарні технології стали невіддільною частиною при розробці інформаційних систем.

Хмарне сховище – важлива складова будь-якої сучасної системи. Незалежно від того, чи це база даних програми, чи сховище для персональних даних клієнтів, для збереження та доступу до даних знадобиться таке рішення, яке надійно та ефективно задовольнити потреби користувачів.

Оскільки способи використання хмарних технологій продовжують розширюватися, особливо зі зростанням попиту на мережеві моделі з доступом з будь-якого місця зростає і кількість хмарних рішень для зберігання даних.

1.1. Хмарні обчислення. Хмарні сервіси

Хмарні обчислення розглядають інфраструктуру, платформу та програмне забезпечення як послуги, які надаються користувачам сервісів на основі замовлення за моделлю "оплата за час використання".

Забезпечення хмарними ресурсами організовується зі сторони провайдерів хмарних сервісів. Серед основних можна виділити Amazon Web Services, Google Cloud, IBM Cloud та Oracle Cloud. Вони надають хмарні ресурси у вигляді послуг через хмарні сервіси. У промисловості ці послуги називаються відповідно «Infrastructure as a Service» (IaaS), «Platform as a Service» (PaaS) та «Software as a Service» (SaaS).

Інфраструктура як послуга (IaaS) – це бізнес-модель надання хмарних ресурсів у вигляді IT-інфраструктури, що забезпечує доступ на вимогу до обчислювальних ресурсів, таких як сервери, сховища та мережеві ресурси з оплатою за час використання. IaaS приваблива та вигідна тим, що придбання обчислювальних ресурсів для розгортання на них додатків чи зберігання даних традиційним способом вимагає часу і капіталу. Організації повинні купувати обладнання і керувати процесом закупівель, які можуть тривати довгий термін. Вони повинні інвестувати у фізичні приміщення, як правило, спеціалізовані кімнати з електроживленням, вентиляцією та охолодженням. А після розгортання систем їм потрібні спеціалісти для управління та обслуговування всього серверного та комунікаційного обладнання. Все це складно викликає складнощі при потребі масштабування, коли попит зростає або бізнес розширюється. У компаній з’являється ризик зіткнення з нестачею потужностей або надмірним розширенням інфраструктури, що призведе до значних лишніх витрат при простоюванні фізичного обладнання. Найбільш типовими прикладами провайдерів у цій сфері є Amazon Web Service (AWS), IBM, VMware. Вони надають сервіс як послуга по оренді інфраструктури.

Платформа як послуга (PaaS) – це тип сервісної моделі хмарних обчислень, що надає користувачу гнучку та масштабовану хмарну платформу для розробки, розгортання, запуску та управління своїми додатками. Сервіс відповідає за оновлення операційної системи та інструментів розробки, і, відповідно, обслуговування апаратного забезпечення. Даний тип хмарних обчислень дозволяє уникнути потреби з встановлення апаратного чи програмного забезпечення для розробки та розміщення нових додатків клієнта даного сервісу. В результаті розробка стає простіше, швидшою та безпечнішою, що дає змогу розробникам зосередитися на реалізації додатків. Найбільш типовим прикладом додатків у цій галузі є Google App Engine та Windows Azure. Обидва вони служать розподіленою платформою

Програма як послуга (SaaS) – це найпоширеніший тип хмарних сервісів серед основних типів. SaaS (програмне забезпечення як послуга) відрізняється від PaaS, SaaS є більш цілеспрямованою послугою, яка надає обчислювальні ресурси або ресурси для зберігання даних. І SaaS також відрізняється від IaaS, який надає середовище для користувачів, щоб вони могли налаштувати додаток Saas надає лише деякі спеціальні послуги для додатків

Даний тип хмарних сервісів надає користувачам готові та розвернуті на серверах додатки, керування над якими здійснюють сторонні постачальники цих сервісів. Користувач отримує доступ до цих сервісів через мережу Інтернет. Завдяки цьому у користувача цих сервісів відпадає потреба у завантаженні або встановленні жодного програмного забезпечення чи додатків на власні пристрої. Використання сервісів за цим типом здійснюється за допомогою безоплатної моделі або моделі, яка базується на варіативних варіантах підписки на сервіс. Сторонні постачальники хмарних сервісів керують усіма можливими технічними питаннями по реалізації та підтримки роботи хмарного сервісу, контролюють проміжне програмне забезпечення, мережу, сервери, сховища. Це призводить до оптимізації обслуговування та підтримки бізнесу.

1.2. Типи хмарних архітектур

Хмарна архітектура надає користувачам хмарних послуг доступ до потужних інструментів, які можуть використовуватися для миттєвого зберігання та доступу до важливих даних за невелику вартість. Публічні, приватні, гібридні та мультихмарні рішення дозволяють компаніям підтримувати своє бачення за допомогою новітніх додатків та можливостей управління даними.

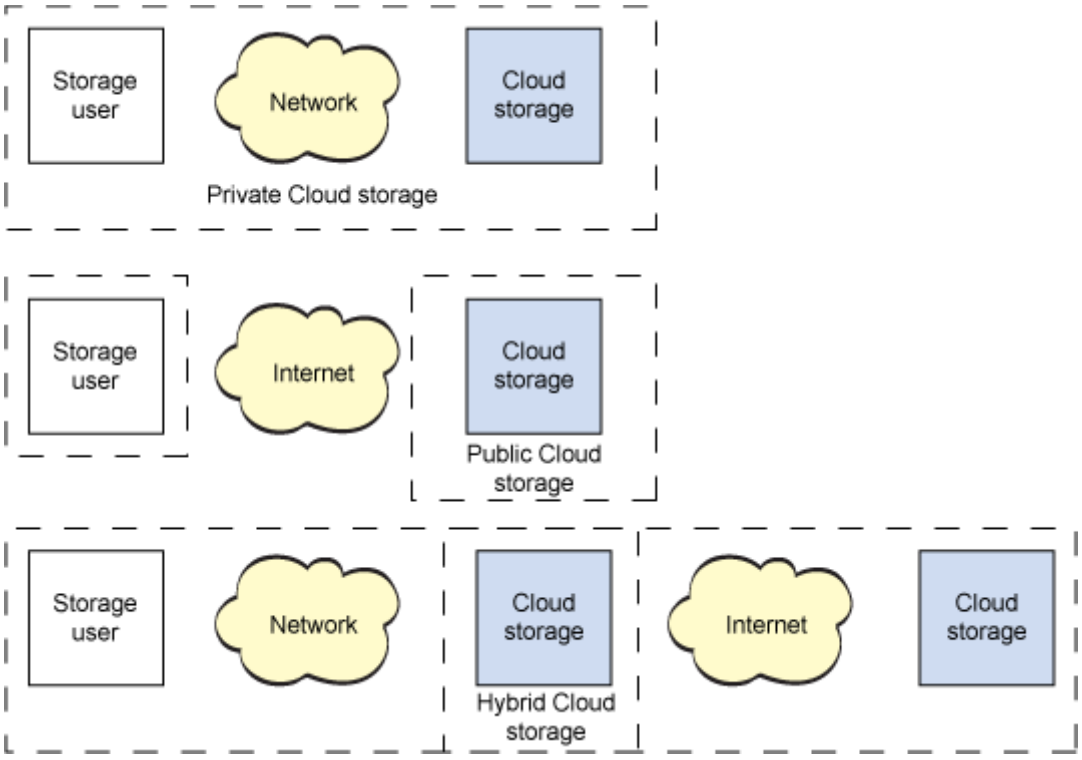

Рис. 1.1. Типи хмарних архітектур

Публічна хмара надає користувачам доступ до спільних ресурсів на віртуальних серверах, що дозволяє їм отримати доступ до більш потужного обладнання за нижчою ціною. Архітектура публічної хмари підходить для паралельного багатокористувацького доступу, і це означає, що користувачі одночасно захищені та вільні використовувати ресурси на власний розсуд.

Хоча публічні хмари пропонують економічно ефективну масштабованість, вони також становлять ризик для мережевої архітектури. Переміщення даних за межі майданчика створює точку атаки, якою можуть скористатися кіберзлочинці, щоб отримати доступ до інформації з обмеженим доступом або приватної інформації. Ця вразливість робить користувачів вразливими, коли компанії не в змозі піідтримувати безпеку зі зростаючими загрозами.

Обмежуючи хмарні сховища та обчислення локальними мережами, бізнес додатковий рівень захисту від потенційних зловмисних атак. Цей захист має важливе значення для забезпечення того, щоб важлива інформація не була втрачена, викрадена або знищена.

IT-лідери та власники малого бізнесу можуть скористатися перевагами багатьох постачальників послуг і платформ для створення спеціалізованих хмарних моделей. Мультихмарна архітектура спрощує передачу, зберігання та доступ до життєво важливих даних, як ніколи раніше. Завдяки мультихмарній архітектурі компанії можуть спростити управління даними та створювати масштабовані корпоративні рішення.

Поєднуючи віддалені та локальні хмарні сервіси, IT-лідери та малі підприємства можуть створювати спеціалізовані середовища, які органічно розподіляють ресурси між внутрішніми та зовнішніми серверами. Гібридна архітектура це поєднання інфраструктури публічної хмари та приватної хмари. Це хороша синергія між масштабованістю та економічною ефективністю публічної хмари та контролем конфіденційних даних і додатків, який пропонує приватна хмара. Гібридна хмара ідеально підходить для різних робочих навантажень і потреб в управлінні даними.

1.3. Хмарні сховища даних

Кожний хмарний сервіс з категорій IaaS, PaaS і SaaS надає доступ до сховища даних, але кожна модель використовує власні типи сховищ. Кожна з категорій сервісів і типів сховищ пов'язана з власними специфічними загрозами та міркуваннями щодо безпеки. Розробляючи та впроваджуючи архітектуру хмарного сховища даних враховується категорією послуг ви створюєте або впроваджуєте, а також унікальні характеристики безпеки даних, пов'язані з цією сервісною моделлю.

Кожна категорія хмарних сервісів надає доступ до сховища даних, але кожна модель використовує власні типи сховищ. IaaS використовує логічне (volume) та об'єктне сховище, PaaS – сховища для структурованих та неструктурованих даних, тоді як SaaS може використовувати широкий спектр типів сховищ.

Модель IaaS надає клієнтам хмарних сервісів засоби самообслуговування для доступу та управління обчислювальними ресурсами, сховищами та мережевими ресурсами. Сховище виділяється клієнтам за потреби, і клієнти платять лише за те, що вони використовують.

Модель сервісу IaaS використовує дві категорії сховищ:

- *Том (логічний диск)* – це віртуальний жорсткий диск, який можна приєднати до віртуальної машини (VM) і використовувати подібно до фізичного жорсткого диска. Операційна система віртуальної машини розглядає том, так само як будь-яка ОС розглядає фізичний жорсткий диск у традиційній моделі сервера. Віртуальний диск може бути відформатований у файловій системі FAT32 або NTFS і керуватися клієнтом. Прикладами сховищ томів є AWS Elastic Block Store (EBS), VMware Virtual Machine File System (VMFS) та Google Persistent Disk.
- *Об'єкт* – це сховище файлів, до якого можна отримати доступ безпосередньо через API або вебінтерфейс, без прив'язки до операційної системи. Дані, що зберігаються в об'єктному сховищі, включають дані об'єкта та метадані та можуть зберігати будь-яку інформацію, зокрема фотографії, відео,

документи тощо. Багато CSP мають інтерфейси, які представляють об'єктне сховище подібно до стандартних структур дерева файлів (наприклад, каталог Windows), але файли насправді є лише віртуальними об'єктами в незалежній структурі сховища, які покладаються на ключові значення для їх посилання та пошуку. Amazon S3 (Simple Storage Service) та Azure Blob Storage є популярними прикладами об'єктних сховищ.

Платформа як послуга відрізняється від IaaS тим, що хмарний провайдер відповідає за всю платформу (на відміну від IaaS, де провайдер хмарного сховища відповідає лише за розподіл обсягу сховища для даних), а клієнт керує лише додатком. Модель сервісу PaaS використовує дві категорії сховищ:

- *Структуровані* – це сховища даних в яких зберігається інформація, яка є високоорганізованою, категоризованою та нормалізованою. Цей тип даних можна помістити в реляційну базу даних або іншу систему зберігання, яка містить набори правил і структуру для пошуку та виконання операцій над даними.
- *Неструктуровані* – сховища, які зберігають неструктуровану інформацію, яку неможливо легко організувати та відформатувати для використання в жорсткій структурі даних, наприклад, у базі даних. До цього типу даних належать аудіофайли, відео, текстові документи, вебсторінка та інші форми тексту і мультимедіа.

У випадку з програмним забезпеченням як послугою (SaaS) хмарний провайдер відповідає за управління не лише всією інфраструктурою та платформою, а й самим додатком. З цієї причини користувач хмари має мінімальний контроль над тим, які типи даних потрапляють у систему; його єдина відповідальність за зберігання даних - це введення допустимих даних у додаток.

Хоча це не зовсім справжні типи даних, модель сервісу SaaS зазвичай використовує два типи (або методи) зберігання даних:

- Зберігання та управління інформацією: Цей тип зберігання даних передбачає, що клієнт вводить дані в додаток через вебінтерфейс, а додаток зберігає і керує цими даними у внутрішній базі даних. Дані також можуть генеруватися додатком від імені клієнта і зберігатися та управлятися аналогічним чином. Ці дані, згенеровані додатком, зберігаються у внутрішньому сховищі тому або об'єкта, але приховані від клієнта.
- Зберігання контенту та файлів: При такому типі зберігання даних клієнт завантажує дані через вебдодаток, але замість того, щоб зберігатися в інтегрованій базі даних, контент і файли зберігаються в інших механізмах зберігання, до яких мають доступ користувачі.

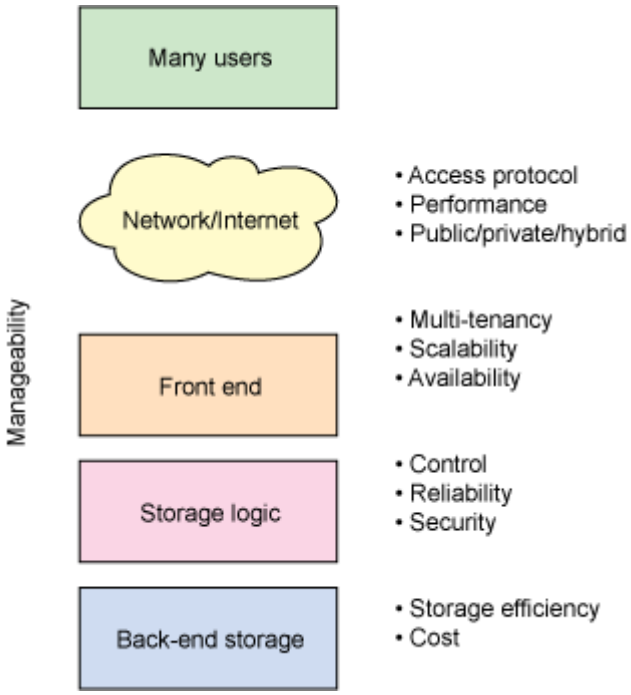

Рис. 1.2. Загальна архітектура хмарного сховища

Архітектура хмарних сховищ – це, перш за все, надання сховища на вимогу у високомасштабований і багатокористувацький спосіб. Як правило (див. Рисунок 1), архітектура хмарного сховища складається з інтерфейсу, який експортує API для доступу до сховища. У традиційних системах зберігання таким API є протокол SCSI; але в хмарі ці протоколи розвиваються. Там ви можете знайти інтерфейси веб-

сервісів, файлові інтерфейси і навіть більш традиційні інтерфейси (наприклад, Internet SCSI або iSCSI). За інтерфейсом знаходиться рівень проміжного програмного забезпечення, який я називаю логікою зберігання. Цей рівень реалізує різноманітні функції, такі як реплікація та зменшення обсягу даних, над традиційними алгоритмами розміщення даних (з урахуванням географічного розміщення). Нарешті, бек-енд реалізує фізичне зберігання даних. Це може бути внутрішній протокол, який реалізує специфічні функції, або традиційні фізичні диски.

1.4. Характеристики хмарних сховищ даних

Серед основних характеристик, що властиві хмарним сховищам масштабаність, однорідність, географічна-розподіленість, сервіс-орієнтована архітектура, ціна програмного забезпечення та безпека.

Масштабованість - одна з характеристик хмарних сховищ, яка означає, що системи хмарних обчислень використовують можливості горизонтального та вертикального розширення. Наприклад, хмарні обчислення Google налічують понад 100 мільйонів серверів, Amazon, IBM, Microsoft, Yahoo та інші пули ресурсів хмарних обчислень налічують сотні тисяч серверів. Хмарні обчислення можуть дати користувачам безпрецедентний масштаб і обчислювальну потужність. Відмовостійкі обчислення є однією з характеристик хмарного сховища, що означає, що в хмарних обчисленнях фізичний розмір ресурсу і логічний масштаб можуть бути динамічно масштабовані, що забезпечує майже лінійне зростання продуктивності. Це буде відповідати потребам додатків та зростанню кількості користувачів.

Однорідність - одна з характеристик хмарного сховища, яка означає, що масштаби хмарних обчислень призводять до того, що програмне та апаратне забезпечення, яке використовується в хмарних обчисленнях, має тенденцію до гомогенізації. Цією особливістю можна керувати для досягнення кращої автоматизації безпеки, наприклад, контролю конфігурації, виявлення вразливостей,

аудит безпеки, але з іншого боку, однорідність також дозволяє сервісам хмарних обчислень з'являтися у всіх користувачів, які користуються цим сервісом. Таким чином, це впливає на велику кількість користувачів і сервісів.

Географічна розподіленість є однією з характеристик хмарних сховищ. У сфері хмарних обчислень ресурси можуть бути розподілені по всьому центру обробки даних, міжрегіональний розподіл. Масштаби хмарних обчислень проривають межу єдиного радіозв'язку. Віртуалізація є однією з характеристик хмарного сховища, що означає, що хмарні обчислення дозволяють користувачам використовувати різні термінали в будь-якому місці, щоб користуватися послугами додатків. Запитуваний ресурс надходить з "хмари". Додаток виконується десь у "хмарі", але насправді користувачу не потрібно розуміти, де саме знаходиться додаток. Люди, які користуються ноутбуком або мобільним телефоном, можуть отримати будь-яке сховище, яке їм потрібно, через мережевий сервіс.

SOA (Сервіс-орієнтована архітектура) є однією з характеристик хмарного сховища. Це основна концепція хмарних обчислень для досягнення відкритої архітектури. SOA, як сервіс-орієнтована архітектура, є моделлю і модель і методологія проектування архітектури програмного забезпечення. SOA використовує існуючу програмну систему компанії для реорганізації нової програмної архітектури. Ця програмна архітектура буде реалізовувати бізнес-зміни в поєднанні з існуючими послугами. Вона створює нове програмне забезпечення і обслуговує всю бізнес-систему підприємства.

Недороге програмне забезпечення є однією з характеристик хмарних сховищ, що означає, що в існуючій архітектурі хмарних обчислень постачальники послуг намагаються використовувати недороге, високонадійне програмне забезпечення для побудови систем хмарних обчислень, а користувачі можуть повною мірою насолоджуватися перевагами "хмари", пов'язаними з низькою вартістю. Всього за кілька сотень доларів за кілька днів можна виконати роботи, на які потрібні десятки тисяч доларів і кілька місяців.

Безпека є однією з критичних характеристик хмарного сховища, особливо для бізнесу, оскільки в хмарному середовищі існує чотири типи моделей розгортання, на додаток до приватної хмари, інші три моделі - публічна хмара, хмара спільноти та гібридна хмара - повинні використовувати загальнодоступну мережу. Деяка інформація використовується для зберігання на ПК або сервері, однак ми можемо розмістити її в хмарі прямо зараз. Таким чином, як забезпечити безпеку облікового запису користувача, якщо додаток може переміщатися з однієї точки в іншу? Провайдери хмарних сервісів побачать деталі. Всі ці питання впливають на рішення людей використовувати хмарні обчислення, тому постачальники послуг хмарних обчислень повинні починати як з технічних, так і з управлінських аспектів.

1.5. Методи доступу

Однією з найяскравіших відмінностей між хмарним сховищем і традиційним є спосіб доступу до нього. Більшість провайдерів реалізують кілька методів доступу, але найпоширенішим є API веб-сервісів. Багато API реалізовано на основі принципів REST, які передбачають об'єктно-орієнтовану схему, розроблену поверх HTTP (з використанням HTTP як транспорту). REST API не мають статусу, а тому прості та ефективні у наданні. Багато провайдерів хмарних сховищ реалізують API REST, зокрема Amazon Simple Storage Service (Amazon S3), Windows Azure™ та Mezeo Cloud Storage Platform.

Однією з проблем API веб-сервісів є те, що вони вимагають інтеграції з додатком, щоб скористатися перевагами хмарного сховища. Тому для забезпечення негайної інтеграції з хмарними сховищами також використовуються загальні методи доступу. Наприклад, використовуються файлові протоколи, такі як NFS/Common Internet File System (CIFS) або FTP, а також блокові протоколи, такі як iSCSI. Провайдери хмарних сховищ, такі як Six Degrees, Zetta і Cleversafe, надають ці методи доступу.

Хоча згадані вище протоколи є найпоширенішими, для хмарних сховищ підходять й інші протоколи. Одним з найцікавіших є веб-розподілене авторство та керування версіями (WebDAV). WebDAV також базується на HTTP і дозволяє використовувати Інтернет як ресурс, доступний для читання і запису. Серед постачальників WebDAV - Zetta та Cleversafe, а також інші.

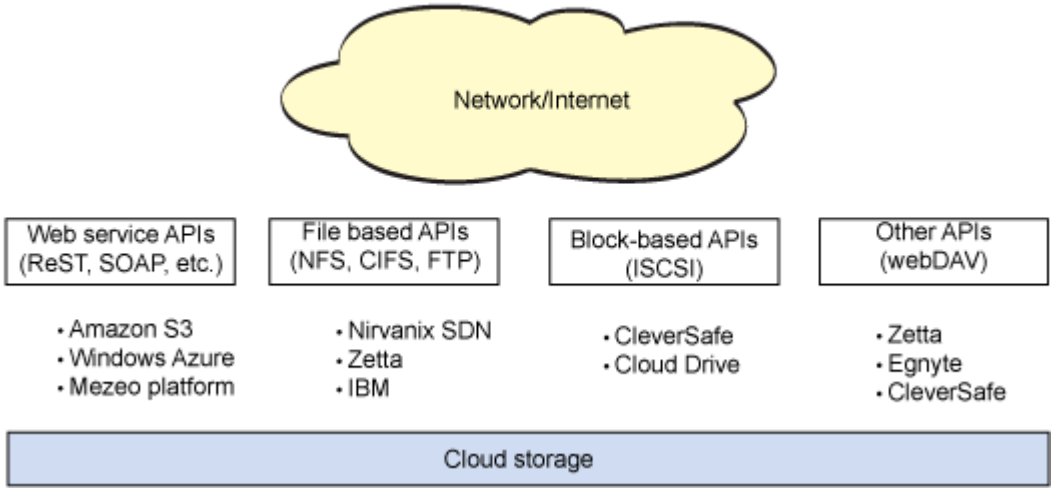

Рис. 1.3. Загальна архітектура хмарного сховища

1.6. Типи сховищ даних

Серед основних типів сховищ даних виділяють:

1. *Локальне ефемерне сховищею*

Локальне серверне сховище все ще може використовуватися в хмарних архітектурах, але його слід вважати ефемерним і недовговічним. Його слід використовувати лише для тимчасового зберігання файлів і журналів перед їх переміщенням на довгострокове зберігання. Конфігураційні файли слід отримувати з іншого джерела, ніж сервер, щоб уникнути їх втрати у випадку руйнування сервера. Бази даних, які використовують локальне ефемерне сховище, повинні бути або

негайно репліковані на інші сервери, або спроектовані так, щоб протистояти втраті даних у разі вимкнення сервера.

2. *Мережева файлова система*

Подібно до мережевого сховища (NAS), спільна файлова система дозволяє одному або декільком серверам підключатися до однієї файлової системи і читати/записувати файли. Часто спільні файлові системи експортуються за допомогою одного або декількох протоколів мережевих файлових систем: NFS, SMB/CIFS або AFP. Залежно від конкретного сервісу, деякі сервіси спільних файлових систем не дозволяють додавати ємність сховища після того, як попередньо виділений обсяг буде вичерпано.

Сервери зі спільною файловою системою використовують мережевий протокол для читання і запису файлів, що дозволяє віддаленому серверу виконувати введення/виведення безпосередньо на рівні блоків. Як наслідок, спільні файлові системи не рекомендуються для рішень, які вимагають прямого доступу до пристроїв вводу/виводу на рівні блоків (наприклад, високопродуктивні сховища даних).

Доступ до спільної файлової системи обмежується локальною мережею.

3. Мережевий пристрій/блокове сховище

Мережеве блокове сховище - це пристрій блокового рівня, який пропонується через мережу, подібно до мережі зберігання даних (SAN). Для сервера він виглядає як будь-який інший носій інформації, його можна монтувати, форматувати за допомогою будь-якої файлової системи та об'єднувати з іншими блоковими пристроями зберігання даних для забезпечення надмірності або підвищення продуктивності (наприклад, RAID 0, 1, 5, 10, 50).

Блокові сховища зазвичай використовуються для високопродуктивних сховищ даних, які потребують прямого доступу до вводу/виводу, а не доступу до файлової

системи. Важливо зазначити, що деякі сервіси блокових сховищ можуть не дозволяти збільшувати ємність сховища після досягнення попередньо виділеного ліміту пам'яті.

Доступ до блочного сховища обмежується локальною мережею.

4. Об’єктне сховище

Об'єктне сховище це тип сховища даних, який дозволяє зберігати дані та пов'язані з ними метадані. Воно розташовується над блоком або файловою системою і часто використовує сховище блоків або файлової системи під ним, хоча ці деталі не видно клієнту. Замість цього клієнти отримують доступ до об'єктів через інтерфейс прикладного програмування (API). Багато сервісів зберігання об'єктів надають API на основі REST, що дозволяє легко читати, записувати, оновлювати та видаляти об'єкти з будь-якої програми без використання файлових систем або прямого блокового вводу/виводу. Сховища об'єктів зазвичай забезпечують високу доступність і надмірність у прозорий спосіб.

Крім того, об'єктне сховище доступне з будь-якого місця, включаючи локальні та віддалені мережі, за умови надання відповідних облікових даних для доступу на читання та/або запис. Деякі сервіси об'єктного сховища дозволяють надавати веб-браузерам прямий доступ до загальнодоступних файлів через протокол HTTP, що дозволяє зберігати та обслуговувати цілі веб-сайти без потреби у виділеному веб-сервері.

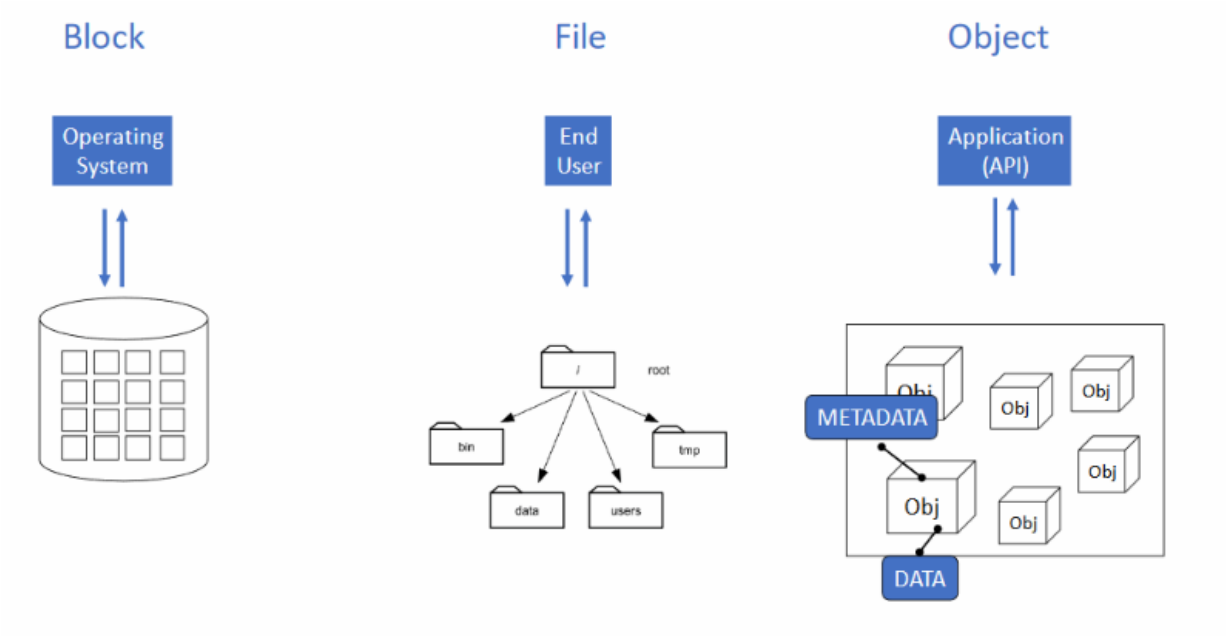

Рис. 1.4. Типи сховищ за моделю даних

Основна відмінність між блоковим, файловим та об'єктним сховищем полягає в тому, хто отримує доступ до даних:

- Блочне сховище видиме для операційних систем або гіпервізорів, які працюють на голому фізичному або віртуальному сервері. Операційні системи записують блоки даних на доріжки та сектори диска.
- Файлове сховище часто видно безпосередньо користувачам у вигляді каталогу за допомогою протоколів зберігання SMB або протоколу зберігання NFS. Користувачі повинні вирішити і знати, де зберігати файли, і пам'ятатиде їх знайти.
- Доступ до сховища об'єктів здійснюється безпосередньо з додатків за допомогою RESTful API. Об'єкт зберігається у плоскому просторі імен з усіма іншими об'єктами в тому ж просторі імен. Ім'я об'єкта використовується для запису та читання об'єктів зі сховища. Сховище об'єктів надає можливість додавати власні метадані до даних програми

1.7. Приклади хмарних сховищ

Додатки для хмарних сховищ - це постачальники хмарних сховищ, які через відкритий Інтернет надають користувачеві можливість оплачувати послуги зберігання даних. Залежно від клієнтської бази та послуг зберігання, що надаються різними способами, їх можна поділити на загальнодоступні додатки (для приватних осіб, домашніх користувачів) та корпоративні додатки для бізнес-користувачів.

Типові загальнодоступні додатки для хмарних сховищ здебільшого надаються користувачам як послуги рівня SaaS. Існує багато хмарних додатків для синхронного резервного копіювання документів, збереження в автономному режимі, завантаження в режимі онлайн, редагування та обміну документами. Типові корпоративні програми для хмарних сховищ здебільшого надаються користувачам на рівні IaaS. Надання базових сховищ і базових послуг резервного копіювання для розробників (переважно для підприємств) в оренду для полегшення роботи власного офісу або відповідно до потреб бізнесу. Орендар розробляє бізнес-рівень і в кінцевому підсумку надає хмарне сховище для бізнес-користувачів. З поступовим розвитком технології хмарних сховищ технології хмарних сховищ, все більше компаній усвідомлюють перспективи хмарних сховищ і починають запускати хмарні додаток для зберігання даних.

Як типовий представник SaaS-додатків, Dropbox раніше з'явився перед широким загалом користувачів, забезпечив функцію синхронного резервного копіювання та змінив офісні звички користувачів. Дозволяючи користувачам уникнути неприємностей, пов'язаних з тим, що для роботи за межами офісу потрібно носити з собою комп'ютер, а модель хмарного обміну файлами також наближається до відстані багатьох інтернет-користувачів.

Як типовий представник додатку рівня IaaS, Amazon S3 створив надання базових засобів зберігання та інтерфейсу для сторонніх компаній для оренди та надання додатків хмарного зберігання для громадськості.

Знаючи про деякі успішні трансформації інтернет-вендорів, телекомунікаційні оператори також запустили хмарні додатки для зберігання даних. Для телеком-операторів хмарні додатки для зберігання даних і послуги хмарних обчислень часто нероздільні. Інтегрований характер хмари та високий рівень безпеки є важкодоступними для інтернет-компаній. Ці переваги можуть стати основою для успіху телекомунікаційних операторів у застосуванні хмарних сховищ. У цій главі ми спочатку представили типові загальнодоступні та корпоративні додатки для хмарних сховищ з точки зору функцій, моделі отримання прибутку та операційних характеристик. Потім представлено бізнес телекомунікаційних операторів у сфері хмарних сховищ даних.

Більшість загальнодоступних хмарних сховищ - це мережеві диски. Основними функціями є завантаження, синхронне резервне копіювання та онлайн-редагування. Перебуваючи в платному режимі, включаючи безкоштовний бізнес і бізнес з доданою вартістю. До цього часу багато додатків мають свої особливості, наприклад, Dropbox - це особливість дружнього інтерфейсу, зручність Google Driver має багато сторонніх додатків. І має свої унікальні можливості спільного редагування та технологію пошуку з розпізнаванням зображень, SkyDrive покладається на користувачів Microsoft для запуску нових продуктів, а iCloud об'єднує сервіси та обладнання Apple.

Dropbox започаткував бізнес онлайн-сховищ у 2007 році. Одне з найбільших і найперших впроваджень хмарних сховищ на дискових накопичувачах. Компанія Dropbox існує вже 9 років, кількість користувачів становить понад 80 мільйонів, сукупно збережено понад 100 мільярдів файлів. Хоча значна кількість користувачів є безкоштовними, операційний дохід все ж таки є значним. Пристрій продуктів Dropbox під Amazon, простий в управлінні, з високою надійністю сервісу та швидкою швидкістю передачі даних. Ці переваги змушують компанію Dropbox залучати більше користувачів. Серед користувачів Dropbox є керівники підприємств, пенсіонери та студенти.

Перша функція - синхронне резервне копіювання. Вона дозволяє користувачеві створити папку синхронного резервного копіювання на терміналі, щоб досягти

синхронізації з резервною копією на мережевому диску. За допомогою простого встановлення плагіну, користувачі можуть автоматично синхронізувати пошту та інші робочі документи з мережевим диском. Коли користувач відкриває і входить на комп'ютер, папка синхронізації в нових або змінених файлах автоматично синхронізується з Dropbox. Коли користувач входить до Dropbox, файл автоматично синхронізується з Dropbox на комп'ютер. Коли файл у папці змінюється, Dropbox завантажує зміни лише до файлової частини, що має високу ефективність. Друга функція - пакетне завантаження. Dropbox підтримує пакетне завантаження файлів. Завантаження веб-сторінки має обмеження для одного файлу в 350 МБ. Клієнтське завантаження необмежене, різні пакети мають різний верхній ліміт трафіку для завантаження і вивантаження. Третя функція - підтримка декількох версій. Dropbox підтримує збереження кількох версій, навіть якщо файл видалено.

Його також можна відновити до будь-якої історичної версії. Так що навіть якщо кількість людей, які брали участь у редагуванні файлу, змінюється. Різні користувачі можуть знайти потрібну їм історичну версію, яка не буде заплутаною.

1.8. Висновки до розділу 1

Хмарн створюють унікальні вимоги до даних про продуктивність безпеки, доступність захисту даних і керованість. Для того, щоб розвіяти багато занепокоєнь. Вимоги потенційних користувачів хмарних технологій повинні систематично враховуватися, а концепція віртуальних контейнерів для зберігання даних забезпечує корисну структуру для роздумів про те, як

про те, як задовольнити ці вимоги. Задовольняючи потреби постачальників сховищ або хмарних сервісів, провайдери зможуть створити багатокористувацьку інфраструктуру зберігання даних, яка буде відрізнятися гнучкістю, безпекою, високою функціональністю і інтероперабельність. Зі стрімким розвитком комунікаційних та інтернет-технологій мобільний зв'язок та інтернет стають

найбільшим потенційним ринком і найпривабливішою сферою. Використання технології розподіленого хмарного сховища для зберігання даних терміналів, забезпечуючи при цьому цілісність і безпеку даних. Загальна архітектура цієї системи побудована за моделлю, орієнтованою на термінали.

РОЗДІЛ 2
АНАЛІЗ ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СИСТЕМИ СХОВИЩА ДАНИХ

Сучасні інформаційні системи більше не реалізуються у вигляді монолітного додатка. Для бізнесу з'являється потреба в великих обчислювальних ресурсах, які можна за потреби масштабувати. Додатки, які мають бізнес значення, вимагають інтеграції з різними системами від сторонніх компаній.

Використання хмарних технологій та сервісів від провайдерів хмарних обчислювальних ресурсів з кожним роком збільшується. Все частіше програмний продукт реалізується у вигляді розподіленої системи з багатьма компонентами, де для повноцінної роботи використовуються різні архітектурні реалізації систем. В більшості випадків будь-який бізнес намагається через технічні або фінансові потреби переходити в хмару та починає використовувати хмарні сервіси для реалізації функціонала, який необхідний для стабільного виконання бізнес-задач.

Для реалізації сучасних інформаційних систем та інтеграції с хмарними сервісами часто з'являється потреба в високопродуктивних, підтримуваних та зарекомендованих часом інструментах розробки інформаційних систем. Одним з найбільш популярних програмних стеків для реалізації сучасних систем є Java разом з Spring Framework.

Серед найкращих провайдерів хмарних ресурсів можна виділити Amazon Web Services. Даний провайдер є популярним через велику варіативність в сервісах, налагодженість їх роботи та інтеграція с більшістю мов програмування та фреймворків

2.1. Amazon Web Services

Amazon Web Services (AWS) – це один з найбільших та найпоширеніших провайдерів хмарних сервісів, що займається їх управлінням та надає обчислювальні ресурси за запитом.

Популярність даного провайдера здобута завдяки своїй доступності та гнучкості у використанні. Він дозволяє бізнесу отримати обчислювальні потужності та орендувати хмарне сховище як послугу. Amazon пропонує ресурси, які необхідні для виконання конкретної задачі з оплатою погодинно або за кількістю запитів до сервісів. Доступний і варіативний інтерфейс для взаємодії з широким діапазоном різних сервісів дозволяє сфокусувати ресурси користувачів на розробці та експлуатації інформаційної системи без необхідності в налаштовувані власного фізичного середовища для коректної роботи системи. Оплата за час використання дозволяє зекономити кошти на факті простоювання ресурсів, оскільки велику частину часу фізичне обладнання буде простоювати, а також морально та фізично старіє.

Одними з найпопулярніших сервісів Amazon Web Services є AWS S3, AWS EC2, AWS Lambda, AWS RDS, AWS ECS, AWS SQS, AWS SNS, AWS Step Functions. Усі сервіси мають відповідні інтерфейси для взаємодії з ними та налаштування роботи. Доступ до сервісів можна отримати використовуючи офіційний вебдодаток AWS Console, консольну утиліту AWS CLI чи бібліотеку, що реалізована для більшості популярних мов програмування AWS SDK.

2.1.1. Amazon Simple Storage Service

Служба зберігання об’єктів *Amazon Simple Storage Service (AWS S3)* – це сервіс для зберігання об’єктів, який забезпечує одні з найкращих функціональних можливостей для збереження та захисту даних, в будь-якому обсязі для різних випадків використання, в тому числі озера даних (Data Lake), вебсайти, мобільні

додатки, резервне копіювання та відновлення даних, архівування. Також велика кількість компаній використовують даний сервіс для власних бізнес-додатків, зберігання структурованих та слабоструктурованих даних для подальшої аналітики.

Основними сутностями якими оперують в рамках сервісі є контейнер (bucket), об’єкт (object), який зберігається всередині контейнеру, та ключ (key), що прив’язаний до об’єкта. Ключ надається клієнтом при завантаженні та використовується для подальшого доступу до цього об’єкту і є унікальним для кожного збереженого об’єкта.

AWS S3 сервіс надає можливість налаштувати збереження і отримання доступу до декількох версій об’єкта, що дає можливість використовувати старіший варіант об’єкта до його оновлення.

Також для сховища класу «S3 Express One Zone» доступна можливість створення директорій, які є складовими частинами ключа до об’єкта. Це дає можливість підвищення швидкості доступу до об’єкта та транзакційного підтримання великої кількості запитів.

Серед доступних можливостей контролю доступу є можливість контролю дозволу на читання та запис авторизованим користувачам для окремих контейнерів, сегментів та об’єктів. Кожен контейнер та об’єкт має списки контролю доступу «ACL», який додається до них як піделемент. ACL визначає типи дозволеного доступу, а також дозволені для доступу облікові запити та групи AWS, що відповідають вимогам. На сьогодні для більшості випадків отримання доступу до елементів S3 сервісу не вимагають потреби налаштування доступів до сегментів та об'єктів.

AWS S3 сервіс забезпечує надійну послідовність та консистентність зчитування після запису для запитів запису та видалення для всіх регіонів. Забезпечується як для запису нових об’єктів, так і для перезапису наявних об’єктів за тим же ключем. Крім того, операції зчитування в Amazon S3 Select, списки керування доступу Amazon S3

ACL, теги об’єктів Amazon S3 і прив’язані до об’єктів метаданні є дуже консистентні та узгодженні.

Сервіс надає можливість вибору між декількома класами зберігання, розроблення в залежності від необхідності у швидкості доступу до об’єктів та надійності зберігання. Для зберігання критичних даних, які є критичними для бізнесу, і частого доступу використовують «S3 Standard» або «S3 Express One Zone». При необхідності збереження коштів на зберігання та при не частому доступі використовують «S3 Standard-IA» або «S3 One Zone-IA». Якщо є необхідність архівувати дані з найменшими витратами на це використовують S3 Glacier Instant Retrieval, S3 Glacier Flexible Retrieval та S3 Glacier Deep Archive.

Найбільш популярним є «Amazon S3 Express One Zone», який є високопродуктивним сховищем, що створене для високопродуктивних систем, які потребують швидкого надання доступу до даних за найменший час. Серед інших класів сховищ AWS S3 він надає найнижчу затримку запиту на зберігання та завантаження даних. Дані зберігаються в новому типі сегмента – сегменту каталогу, що дає збільшення швидкості доступу та підтримки великої кількості одночасних запитів за секунду.

Цей сервіс від провайдера AWS надає велику користь у використанні порівнюючи з іншими у зв’язку з наступними перевагами:

- Контейнери (bucket): об’єкти зберігаються в даному сервісі в сегментах або контейнерах, які дають можливість завчасно структурувати дані за певними критеріями та зберігати їх у відповідний контейнер. Кожен контейнер може містити необмежену кількість об’єктів, кожен об’єкт може мати максимально доступний розмір в 5 ТБ;
- Дозволи: кожному контейнеру та об’єкту всередині контейнеру можна надати певні доступи або заборонити доступ до цього елементу для завантаження в сховище або витягування за ключем. Сервіс надає можливості гнучкого налаштування механізмів для захисту даних від несанкціонованого доступу;

- Стандартизований інтерфейс доступу: AWS S3 сервіс надає стандартні інтерфейси для доступу до елементів, які збережені в сервісі. Основними інтерфейсами доступу є REST та SOAP інтерфейси, а також реалізовані інструменти та бібліотеки для різних мов програмування;
- Захищеність: сервіс дає можливість захисту від взаємодії зі сховищем від неавторизованого користувача.

Серед доступного функціоналу, який надає сервіс AWS S3 також можна виділити можливість створення точок доступу (access points). Вони спрощують доступ до даних для будь-яких сервісів AWS та клієнтських додатків, які використовують AWS S3 для збереження даних. Точки доступу – це іменовані мережеві кінцеві точки, що прив'язані до S3-контейнерів (bucket). Ці точки доступу дозволяють виконувати операції на отримання та завантаження об'єктів в середині контейнера. Кожна окрема точка доступу має окремі дозволи та засоби мережевого контролю, які AWS S3 використовує до будь-якого запиту, що надходить через цю точку доступ. До кожної точки доступу можна застосувати власну політику, що дозволить обмежити доступні запити до сервісу. Дозволено налаштувати будь-яку точку доступу на обробку запитів тільки з віртуальної приватної мережі.

2.1.2. Amazon Elastic Compute Cloud

Хмарний сервіс *Amazon Elastic Compute Cloud (AWS EC2)* – один з найпопулярніших сервісів AWS, що забезпечує можливість створення та масштабування обчислювальних потужностей на вимогу в хмарі. Сервіс дає можливість зменшити витрати на апаратне забезпечення, що дає змогу швидше розробляти та розгортати системи для роботи. Користувачу дається можливість отримати та налаштовувати ресурси, забезпечуючи цим потреби у продуктивному віртуальному сервері. Також дається широкий спектр для вибору типу віртуального сервера.

При необхідності розширення системи або збільшення ресурсів окремого віртуального сервера сервіс дає можливість автоматизувати вертикальне і горизонтальне масштабування системи, додаючи більшої кількості унікальних віртуальних серверів або збільшення потужностей конкретного сервера відповідно. Найчастіше вертикальне масштабування відбувається через зміну типу віртуального сервера, що відповідає за збільшення оперативної пам’яті та покращення типу CPU. Також при необхідності є можливість збільшення прив’язаного до віртуального сервера EBS Volume, що по своєму призначенню відповідає жорсткому диску. Серед доступної конфігурації також доступно вибір операційної системи віртуального сервера.

Кожен повний набір конфігурацій – це є підтримуваний образ машини Amazon (AMI), що надає інформацію для запуску віртуального сервера. При конфігурації сервера надається можливість вибрати відповідний AMI образ. Amazon надає вже готові образи з різними конфігураціями, але також користувачу дається можливість створити власний при необхідності. AMI включає один або декілька знімків AWS EBS Volume, операційну систему, а також певні програми, в тому числі серверні програми.

Для кожного унікального екземпляра віртуального сервера характерні наступні статуси роботи:

- *pending* – екземпляр готується до переходу в робочий стан;
- *running* – екземпляр запущений та готовий до використання;
- *stopping* – екземпляр готується до зупинки;
- *stopped* – екземпляр зупинений, не може бути використаний;
- *shutting-down* – екземпляр в процесі видалення;
- *terminated* – екземпляр остаточно видалено, його неможливо запустити.

Статуси *stopping* та *stopped* характерні тільки для екземплярів віртуальних серверів з використанням прив’язаного до сервера «AWS EBS Volume».

Нижче наведена діаграма базової архітектури AWS EC2 віртуального сервера завантаженого в віртуальну приватну мережу AWS VPC:

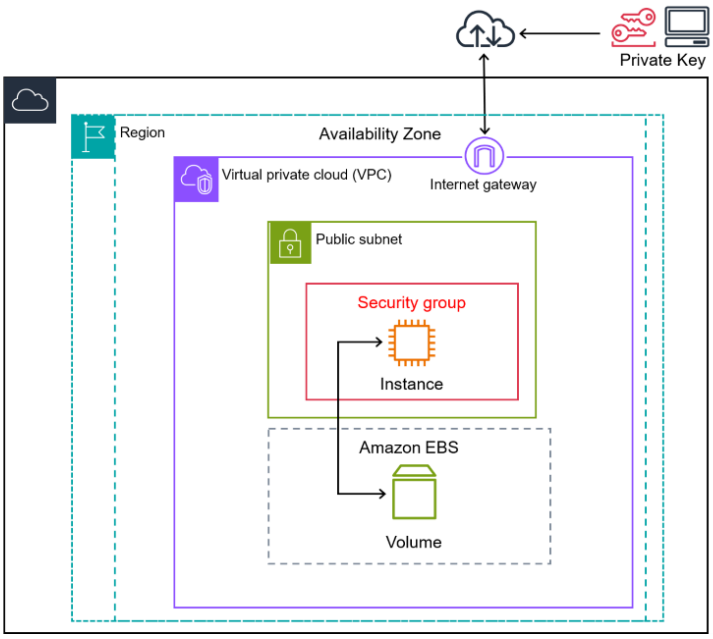

Рис. 2.1. Архітектурна діаграма AWS EC2 екземпляру в приватній мережі.

Відповідно до діаграми, що відповідає налаштуванням користувача, доступ до віртуального сервера відбувається через віртуальний фаєрвол, який відповідає за контроль вхідного та вихідного трафіку, що забезпечується групою безпеки встановленого сервера AWS EC2. Аутентифікація користувача забезпечується публічними та приватними ключами. Також до віртуального сервера прив'язаний EBS Volume, який забезпечує сервер внутрішньою пам'яттю для збереження даних та надалі може бути використаний для інших віртуальних серверів. AWS VPC Gateway забезпечує доступ з інтернет мережею.

За допомогою горизонтального масштабування часто простіше динамічно масштабувати, додаючи більше машин до наявного пулу віртуальних серверів. Вертикальне масштабування часто обмежується потужністю однієї машини, масштабування за межі цієї потужності часто передбачає простої та має верхню межу.

AWS надає послугу групового автомасштабування, призначену для забезпечення автоматичного масштабування різних служб, включаючи EC2. Автомасштабування гарантує, що у вас є достатньо екземплярів EC2 для запуску додатків, і перед запуском служби ви призначаєте групи автомасштабування. Ці групи можуть мати мінімальну або максимальну кількість екземплярів EC2, які автоматично запускаються, якщо в одному з екземплярів виникає помилка або збій.

Також користувачу надається можливість коригувати зону доступу та регіон віртуального сервера.

2.1.3. Amazon Relational Database Service

Хмарний сервіс *Amazon Relational Database Service (AWS RDS)* – це сервіс, що забезпечує можливість налаштування, керування, експлуатації, вертикальне та горизонтальне масштабування реляційної бази даних у хмарі. Сервіс дає можливість спростити налаштування і використання бази даних, а також економить час на розгортанні бази даних, що забезпечує можливість зосередитися на реалізації інформаційної системи, де відповідна база даних буде використовуватися як частина системи. Сервіс відповідає за повне інфраструктурне керування базою даних.

Amazon RDS дає можливість легко використовувати різні типи найбільш розповсюджених реляційних баз даних, для прикладу PostgreSQL, MySQL, SQL Server, Oracle Database, DB2 та інші, а також власна реляційна БД – Amazon Aurora, яка оптимізована для роботи в хмарі та надає додатковий функціонал та можливістю налаштування та масштабування у порівнянні з іншими. Сервіс забезпечує автоматичне створення резервної копії даних, підтримує актуальність версій програмного забезпечення, яке необхідне для коректної роботи БД, забезпечує можливістю адміністрування. Використання цього сервісу забезпечує також можливості міграції даних між базами даних в рамках одного регіону або міжрегіональну міграцію даних.

Серед переваг у порівнянні з іншими подібними сервісами від інших провайдерів можна виділити можливість масштабування розрахункових ресурсів і об’єму сховища, що пов’язано з робочим екземпляром бази даних. Також можна виділити спрощення процесу реплікації даних для підвищення доступності даних при великій кількості запитів на зчитування. Присутній функціонал для відновлення БД у випадку не коректного завершення роботи чи потенційної втрати даних.

Користувачу надається можливість сконфігурувати доступність бази даних, забезпечити доступ тільки в рамках хмарної віртуальної мережі в приватній підмережі, що забезпечить доступ до неї тільки тим віртуальним серверам, які також знаходяться в цій же приватній мережі.

Як і інші сервіси Amazon, оплата відбувається за факт користування та кількість годин, коли хмарна база даних використовувалася, а також за кількість звернень до реляційної бази даних.

2.1.4. Amazon Virtual Private Cloud

Хмарний сервіс *Amazon Virtual Private Cloud (AWS VPC)* – це сервіс, який надає можливість створити логічно ізольовану віртуальну мережу в публічній хмарі AWS, а також запустити в ній інфраструктурні та прив’язані до них програмні ресурси. Функціонал сервісу дає можливість ізолювати та повністю контролювати таблиці маршрутизації та мережеві шлюзи, визначати власний діапазон IP-адрес, створювати підмережі. Можна використовувати IPv4 та IPv6 для більшості ресурсів, що дає можливість отримати безпечний та зручний доступ до ресурсів і робочих додатків.

Amazon VPC сервіс надає функціонал для повного відтворення локальної мережі та забезпечення необмеженого масштабування, а також інструменти для реалізації цієї можливості. Принцип роботи та логіка налаштувань мережі відтворює налаштування локальної мережі, що існує фізично з відповідним апаратним та програмним забезпеченням. Сервіс гарантує високу надійність використання

хмарних ресурсів, забезпечує їх ізольованість від загального інтернету, дозволяє з'єднати та забезпечити працездатність компонентів віртуальної приватної мережі, а також дає можливість налаштувати зв'язок приватної мережі з мережею Інтернет.

Ізольованість ресурсів забезпечується за рахунок розділення хмарної мережі на підмережі, які діляться на публічні та приватні. Компоненти системи в публічній підмережі мають доступ в Інтернет через Elastuc IP та Internet Gateway. У приватній підмережі компоненти не мають прямого доступу в Інтернет, тільки через публічні підмережі та NAT.

2.1.5. Amazon Simple Queue Service

Хмарний сервіс *Amazon Simple Queue Service (AWS SQS)* – служба черги повідомлень, яку використовують для асинхронного надсилання, зберігання та отримання повідомлень. Функціональні можливості сервісу дають можливість виділити окремі мікросервіси, розподілені системи та без серверних додатків один від одного та дозволити їх масштабування. Це дає змогу користувачу не реалізовувати власну чергу та забезпечує готовий сервіс для цієї задачі з безпечним обміном повідомлень між компонентами інформаційної системи.

Найрозповсюдженішим способом інтеграції з цим сервісом є через використання програмних інтерфейсів, який доступний для багатьох мов програмування, або виконуючи прямі HTTP запити до сервера.

Amazon SQS підтримує асинхронні задачі, що дає можливість не використовувати прямі запити до інших компонентів системи та чекати відповіді від них, а надіслати повідомлення в чергу, яке через певний проміжок часу у міру проходження повідомлення в черзі та зайнятості відповідного сервісу, буде зчитане сервісом, що відповідає за отримання повідомлень з черги та на основі них виконує відповідну бізнес-задачу.

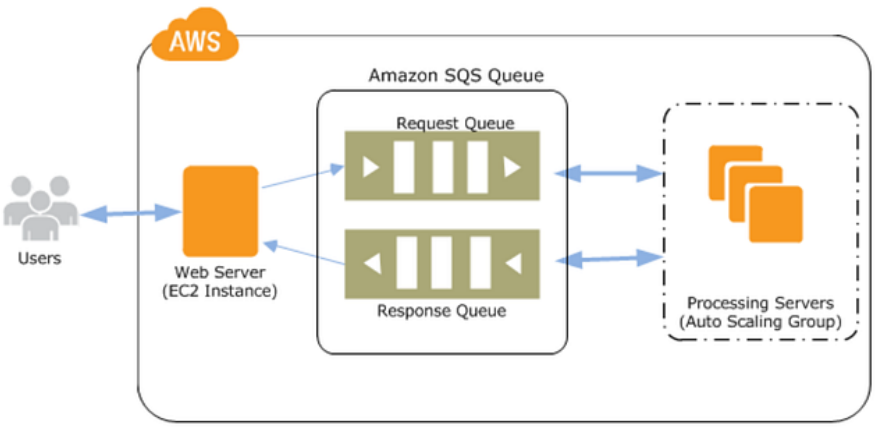

Рис. 2.2. Приклад використання Amazon SQS черги.

Виділяють два типи черг в даному сервісі – «First-In, First-Out» (FIFO) та стандартні черги:

- FIFO черга – це черга де повідомлення залишаються в тому самому порядку, в якому вони були надіслані, що є критичним для систем, де послідовність повідомлень важлива. Вони підтримують до 300-от надсилань, отримань та видалень повідомлень в черзі в секунду. Черга може одночасно оперувати 20 тисячами унікальних повідомлень.

Рис. 2.3. FIFO черга.

- Стандартні черги зберігають повідомлення в тому самому порядку, в якому вони були надіслані, але доступна можливість зміни послідовності обробки повідомлень або зміни їх вихідного порядку. В стандартній черзі може одночасно бути до 120 тисяч повідомлень.

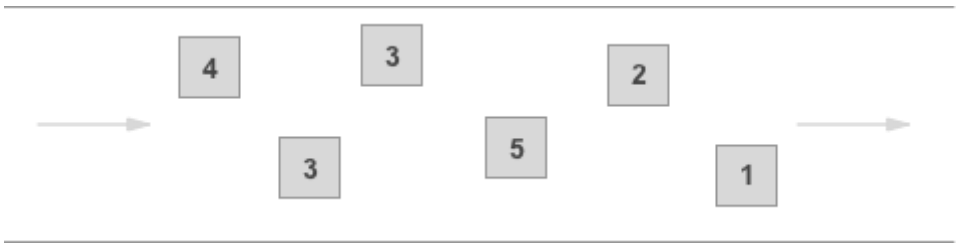

Рис. 2.4. Стандартна черга.

Частота доставлення повідомлень відрізняється між стандартною чергою та чергою FIFO, оскільки повідомлення FIFO доставляються рівно один раз, тоді як у стандартних чергах повідомлення доставляються принаймні один раз.

До переваг використання цього сервісу відносять:

- Забезпечення користувача вже реалізованим сервісом, що відповідає за концепт черг. Це дає можливість розробляти лише компоненти інформаційної системи, які відповідають за бізнес-логіку.
- Amazon SQS забезпечує високу продуктивність через здатність відокремлювати окремі компоненти системи, даючи цим компонентам працювати та обробляти асинхронно та незалежно. При умові масштабування окремих компонентів, які орієнтуються на отримання повідомлень з черги, вдається забезпечити відмовостійкість системи.
- Сервіс має широку інтеграцію з іншими сервісами AWS, такими як AWS EC2, AWS Step Functions, AWS Lambda та AWS RDS.
- Можливість встановлення затримки на відправлення та зчитування повідомлень з черги, а також доступність модифікації видимості та TTL повідомлення в черзі;
- Для забезпечення надійності повідомлень Amazon SQS зберігає їх на декількох серверах. Стандартні черги підтримують доставлення повідомлень принаймні один раз, а черги FIFO підтримують обробку повідомлень точно один раз і режим високої пропускної здатності.

Повідомлення може зберігається в черзі тривалий час, від однієї хвилини до 14 діб. Після цього повідомлення автоматично видаляються з черги. Користувач може створити необмежену кількість черг, що дає змогу виділити черги за типом інформації в повідомленні.

Для забезпечення безпеки повідомлення користувач може використовувати шифрування повідомлення перед відправленням його в чергу, що забезпечить конфіденційність інформації в повідомленні, а при зчитуванні повідомлення іншим сервісом необхідно буде його дешифрувати та отримати цим оригінальне повідомлення.

2.1.6. Amazon Elastic Container Service

Найпопулярніший оркестратор контейнерів – це Kubernetes. Для використання доступні також і інші, такі як Docker Swarm та AWS Elastic Container Service.

Хмарний сервіс *Amazon Elastic Container Service (AWS ECS)* – це сервіс для керування та оркестрування контейнерів, що дає можливість легко розгортати, керувати та масштабувати контейнерні додатки. Інтегрований як з інструментами та сервісами AWS, наприклад з Amazon Elastic Container Registry, так і зі сторонніми інструментами, для прикладу Docker. Це дозволяє спростити конфігураційні ресурси та надає можливість розробникам зосередити уваги на імплементації додатка.

Сервіс дає можливість запускати та масштабувати контейнерні робочі навантаження без складнощів з управління площиною контролю в регіонах AWS у хмарі та локально.

Вибір при налаштовуванні конфігурації поділяється на два інфраструктурних типи для запуску контейнерів при запуску окремих завдань та створення служб:

- AWS Fargate - це без серверний спосіб розміщення робочих навантажень Amazon ECS. Ви можете запускати контейнерні додатки, використовуючи тип запуску Fargate, без резервування та керування базовою інфраструктурою.
- AWS EC2 – використовується для запуску контейнерних додатків на екземплярах Amazon EC2. Дає можливість зберегти контроль інфраструктури, при цьому підтримка, масштабування та контролювання екземплярів EC2 переходить на вас.

2.1.7. AWS SDK

Проєкт AWS Software Development Kit (AWS SDK) – це програмне рішення для різних мов програмування, що дає можливість використовувати сервіси AWS через зручний інтерфейс взаємодії з ними. Цей продукт спрощує розробку додатка, підтримуючи та надаючи набір узгоджених і знайомих бібліотек розробникам. Він забезпечує підтримку життєвого циклу API, таких як управління обліковими даними, повторні запити, серіалізація та маршалінг даних. AWS SDK підтримує високорівневі абстракції для спрощення розробки.

SDK для Java містить декілька функцій, які спрощують програмування для сервісів AWS:

- SDK приховує складні механізми, що стоять за отриманням посторінкових результатів та ресурсів;
- Асинхронне програмування з не блокуючим вводом/виводом допомагає писати паралельний код з кращою продуктивністю. SDK надає переваги HTTP/2, такі як зменшення затримок, де це можливо;
- Java SDK може генерувати метрики, які допомагають відстежувати робочий стан додатків.

Серед доступних AWS сервісів AWS SDK надає доступ до сервісів CloudWatch, Amazon Cognito, AWS RDS, AWS DynamoDB, AWS EC2, AWS IAM, AWS Kinesis, AWS Lambda, AWS Step Functions, AWS S3, AWS Simple Notification Service та AWS Simple Queue Service.

До загальних функцій, згаданих раніше, Java SDK надає можливості для специфічних сервісів AWS:

- Amazon S3 - Щоб спростити роботу з файлами та каталогами в Amazon S3, SDK надає S3 Transfer Manager. Для підвищення продуктивності та надійності при використанні стандартного асинхронного S3 API, SDK пропонує S3-клієнт на основі AWS CRT;
- DynamoDB - об'єктно-орієнтована, можливість відображення надається за допомогою DynamoDB Enhanced Client API.
- IAM - API IAM Policy Builder надає об'єктно-орієнтований спосіб створення IAM-політик зі збереженням типів політики.

2.2. Високорівнева мова програмування Java

Мова програмування *«Java»* - це об'єктноорієнтована мова програмування та платформа, одна з найпопулярніших мов програмування. На сьогодні Java займає одну з найвищих позицій з використання для реалізацій складних корпоративних додатків, а також за кількістю активних проєктів, поруч з мовами програмування як C#, JavaScript, Python та C++.

Важливою особливістю мови Java, яка відрізняє її від багатьох інших, є те, що вона розроблена таким чином, що код, написаний на Java, може бути запущений на будь-якій системі, на якій може працювати віртуальна машина Java (JVM). Мова Java слідує концепції «Write Once Run Anywhere», що означає, що додаток, який написаний на одній системі, може працювати також і на інший. Ця особливість та,

відповідно, наявність віртуальної машини була використана як гасло для просування багатоплатформних можливостей Java. Основна філософія її створення - сумісність між різними пристроями - залишається найсильнішим аргументом на користь Java для нових корпоративних додатків. Об'єктноорієнтована архітектура Java дозволяє створювати модульні програми та багаторазовий код, скорочуючи цикли розробки та подовжуючи термін служби корпоративних додатків.

Java - це технологія, що складається з мови програмування та програмної платформи. Написаний код програми компілюється та перетворюється на байт-код Java. Байт-код – це набір інструкцій для JVM, яка є частиною середовища виконання Java (JRE). Байт-код Java працює без змін на будь-якій системі, яка підтримує JVM, що дозволяє запускати Java проєкт будь-де.

Програмна платформа Java складається з JVM, Java API та повного середовища розробки. JVM аналізує та виконує (інтерпретує) байт-код Java. Java API складається з великого набору бібліотек, включаючи базові об'єкти, мережеві функції та функції безпеки; генерацію розширюваної мови розмітки (XML); та вебсервіси. Разом мова Java та програмна платформа Java створюють потужну, перевірену технологію для розробки корпоративного програмного забезпечення.

Коли справа доходить до вибору мови програмування та середовища для корпоративного додатку, існують вагомі технічні причини розглянути Java, включаючи інтероперабельність, масштабованість та адаптивність.

Коли справа доходить до вибору мови програмування та середовища для корпоративного додатка, існують вагомі технічні причини розглянути Java, включаючи інтероперабельність, масштабованість та адаптивність.

Масштабованість платформи є ключовим атрибутом Java. За допомогою Java можливо використовувати одну платформу для широкого спектра випадків використання. Десктопні додатки можна легко адаптувати для запуску на менших пристроях з обмеженими ресурсами. Можлива міграція додатків з мобільних пристроїв на настільні комп'ютери, розробляючи бізнес-додатки для платформи

Android, а потім інтегрувати їх у поточну систему для настільних комп'ютерів, оминаючи довгі та дорогі цикли розробки.

Сьогодні мова Java набула широкого використання для розробки різних типів додатків та систем. Серед них виділяють:

- *Хмарні додатки*. Незалежність від платформи та масштабованість Java роблять її придатною для проєктів хмарних обчислень. Підхід Write Once Run Anywhere (WORA) дозволяє розгортати Java-додатки в різних хмарних середовищах без необхідності модифікації коду, забезпечуючи гнучкість та економію коштів для бізнесу. Java пропонує надійні засоби безпеки, а також можливості інтеграції з вебсервісами, базами даних і системами обміну повідомленнями, що робить її ідеальним вибором для хмарних додатків, які вимагають високого рівня надійності та продуктивності.
- *Мобільні додатки*. Мова Java дає можливість розробляти додатки для платформи Android, яка є найпопулярнішою операційною системою для мобільних пристроїв. Широка підтримка Java з боку спільноти розробників забезпечує доступність ресурсів та допомоги, що спрощує процес розробки та дозволяє розробникам створювати високоякісні мобільні додатки.
- *Веб додатки*. Завдяки можливостям Java Enterprise Edition та серверів Java-додатків з відкритим кодом, таких як WildFly та Apache Tomcat, Java має багату екосистему інструментів для створення та підключення масово масштабованих додатків, які підтримують роботу найбільших вебсайтів та бізнес-операцій у світі.
- *Big data додатки*. Особливості Java, які роблять її придатною для проєктів з великими даними, включають її розподілену природу, підтримку багатопотоковості, здатність ефективно обробляти великі обсяги даних та масштабованість. Ці особливості роблять Java сильним претендентом на обробку складних вимог додатків для роботи з великими даними. Крім того, розгалужена екосистема бібліотек і фреймворків Java, таких як Hadoop, може спростити розробку додатків для роботи з великими даними, гарантуючи розробникам

доступ до інструментів і ресурсів, необхідних для побудови надійних і високопродуктивних рішень.

Java також користується популярністю у стратегічних планувальників завдяки своїй здатності адаптуватися до нових випадків використання. Наприклад, Java вважається ідеальною платформою для Інтернету речей (IoT). Типовий додаток IoT об'єднує велику кількість розрізнених пристроїв - завдання, яке значно спрощується завдяки тому, що мільярди пристроїв працюють під управлінням Java. Крім того, розгалужена екосистема розробників Java постійно розробляє та ділиться новими бібліотеками з функціональністю, спеціально призначеною для розробки додатків IoT.

2.3. Spring. Spring Framework

Фреймворк *«Spring Framework»* - це програмний компонент та універсальний інструмент для реалізації програм на мові Java, а також інших мов програмування, які пов’язані з віртуальною машиною Java (JVM), наприклад Kotlin та Groovy. Цей фреймворк є каркасом для розробки додатків різного рівня архітектурного і бізнес-рівня складності, в тому числі веб-додатків для корпоративних систем, час експлуатації та підтримки яких зазвичай дуже великий.

Фреймворк володіє багатьма перевагами у порівняні з традиційними підходами до розробки програм на будь-яких мовах програмування. Серед переваг можна виділити наступні:

- *Зменшення об’єму коду*. Однією з найбільших переваг фреймворку є можливість налаштування внесення залежностей (Dependency injection), що дає змогу зменшити пряму залежність між сутностями та винести відповідальності створення об’єктів та встановлення залежних сутностей на сторону фреймворку, особливо якщо є потреба у використані вже наявного екземпляру класу багатьма

іншими об’єктами, які залежні стосовно нього, і дотримання наявності одного екземпляру класу в рамках програми. Оскільки код для подібних задач часто створює багато проблем і ускладнює систему, фреймворк дозволяє це спростити;

- *Спрощення конфігурації додатка*. Внесення залежностей в значній мірі спрощує процес конфігурації додатка. Фреймворк надає можливісті гнучкої конфігурації класів, впровадження в інші класи залежностей. Це дає можливість при необхідності змінювати залежність з однієї реалізації на іншу без значних внесень змін в код програми.

- *Можливість керувати загальними залежностями в рамках одного контексту*. Екземпляри залежностей впроваджуються там, де вони необхідні при традиційному підході до управління залежностями загальних служб, що призводить до розповсюдження залежностей на дочірні класи, які важко підтримувати надалі. При використанні впровадженні залежностей інформація про всі залежності знаходиться в окремому сховищі, що покращує керованість ними та зменшує схильність до помилок при запуску додатка.

- *Покращення тестування*. При проектувані класів на впровадження залежностей стає можливим проста заміна залежностей, що веде до покращення юніт-тестування та дозволяє спростити процес написання тестів та дотримуватися практик тестування, такі як TDD.

Окрім цього Spring Framework надає інший функціонал для спрощення розробки. Серед них можна виділити прикладні інтерфейси API, що використовуються для надання доступу до даних як до баз даних, файлових систем, чи використовуючи HTTP запити до серверів, а також функціонал шаблону MVC.

Фреймворк охоплює різні модулі, що виконують відповідні задачі та часто є складовими та залежностями по відношенню до інших модулів. Основні серед них наступні:

- *Core Container* – модуль інверсії контролю, що відповідає за впровадження залежностей до сутностей, які їх потребують, а також керує життєвим циклом кожного створеного об’єкта;

• *AOP* – компонент, що відповідає за аспектно-орієнтовну поведінку, яка дозволяє реалізовувати в одному місці скрізну логіку, тобто ту, яка використовується в багатьох частинах додатка і забезпечує її автоматичне використання усюди в додатку;

• *Data Access / Integration* – модуль для доступу до даних, включає JDBC, ORM. OXM та JMS;

• *WEB* – модуль, який дозволяє реалізувати веб-додаток з мінімальними конфігураційними змінами, забезпечує максимальну гнучкість при виборі способу реалізації користувацького інтерфейсу для веб-додатку. Найчастіше за все використовується шаблон MVC, що з розвитком фреймворку дозволило легким способом впровадити каркас веб-додатку. Наявна також підтримка протоколу WebSocket;

• *Test* – складається з функціональних компонентів, які дозволяють легко тестувати програми як з юніт тестування, так і з допомогою інтеграційного тестування. Spring надає можливості для тестування HTTP інтерфейсів веб-додатка.

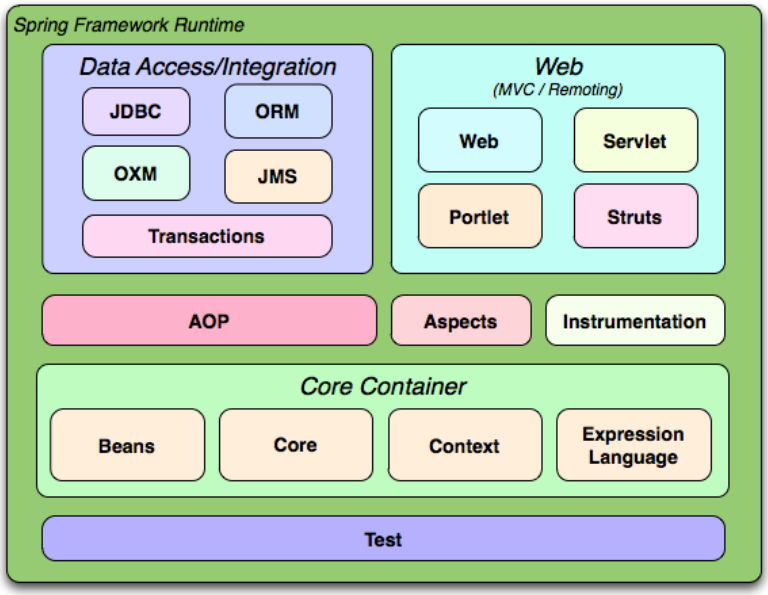

Рис. 2.5. Компоненти Spring Framework.

2.3.1. Spring MVC

Компонент «*Spring MVC*» - це модуль, який найбільше використовується для реалізації веб-додатка. Його основою є слідування архітектурному патерну MVC (Model-View-Controller). Цей патерн відповідає за дотримання специфікації в веб-додатка, що складається з наступних аспектів:

- Model (модель) – відповідає за інкапсуляцію даних веб-додатка;
- View (відображення) – відповідає за відображення даних моделі, найчастіше репрезентується як згенерований HTML, що відображається у браузері, або як набір відформатованих структурованих даних у форматі JSON або XML;
- Controller (контролер) – відповідає за обробку вхідного запиту та формування відповіді клієнту у вигляді згенерованої сторінки з даними моделі або дані у відповідному форматі.

Одним з основних внутрішніх компонентів цього модулю на якому побудована вся логіка роботи Spring MVC є Dispatcher Servlet. Його основною задачею є обробка вхідних HTTP-запитів і відправлення HTTP-відповіді клієнту.

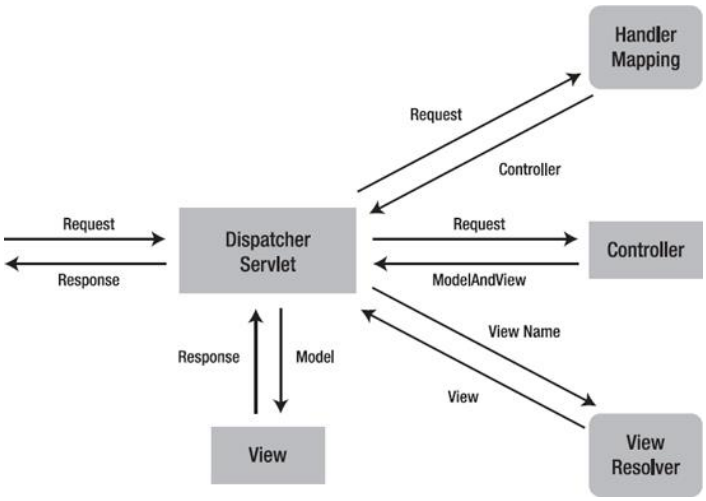

Рис. 2.6. Діаграма роботи Dispatcher Servlet.

Принцип роботи Dispatcher Servlet наступний:

- При отриманні HTTP-запиту від клієнта Dispatcher Servlet звертається до Handler Mapping, який визначає контролер, що буде відповідати за обробку запиту. Вибір визначається за вхідним методом HTTP-запиту, частиною URL, яка відповідає за шлях (URL-path), та вхідними параметрами чи тілом запиту.
- Викликаний контролер виконує відповідну обробку запиту з використанням вказаної бізнес-логіки та формує готову модель даних.
- Інтерфейс View Resolver визначає відповідну сторінку, на основі якої буде сформована сторінка з даними моделі. У випадку відсутності сторінки віддаються дані моделі. Такий підхід найбільше використовується сьогодні.
- Після створення View, Dispatcher Servlet відправляє дані моделі у вигляді атрибутів у вид. Далі готова сторінка відправляється клієнту як відповідь на запит.

Сьогодні найбільш популярним способом використання Spring MVC є побудова на основі нього RESTful веб-додатка.

2.3.2. Spring Data. Spring Data JPA

Головною задачею Spring Data є надати звичну та послідовну модель програмування на основі Spring для доступу до даних, зберігаючи при цьому особливості базового сховища даних.

Це полегшує використання технологій доступу до даних, реляційних і нереляційних баз даних, фреймворків для картографічної редукції та хмарних сервісів даних. Це парасольковий проект, який містить багато підпроектів, специфічних для конкретної бази даних. Проекти розробляються у співпраці з багатьма компаніями та розробниками, які стоять за цими захоплюючими технологіями.

Серед доступних можливостей можна виділити:

- Потужний репозиторій та власні абстракції відображення об'єктів

- Динамічне створення запитів на основі імен методів репозиторію
- Реалізація базових класів домену, що забезпечують базові властивості
- Підтримка прозорого аудиту (створено, востаннє змінено)
- Можливість інтеграції користувацького коду репозиторію
- Легка інтеграція Spring через JavaConfig та власні простори імен XML
- Розширена інтеграція з Spring MVC контролерами
- Експериментальна підтримка міжсховищної персистентності

Spring Data JPA або JPA розшифровується як Java Persistence API, тому перед тим, як розглядати його, ми повинні знати про ORM (Object Relation Mapping). Отже, відображення об'єктних зв'язків - це просто процес збереження будь-якого об'єкта Java безпосередньо в таблиці бази даних. Зазвичай, ім'я об'єкта, що зберігається, стає ім'ям таблиці, а кожне поле в цьому об'єкті стає стовпчиком. Після налаштування таблиці кожен рядок відповідає запису в додатку. Гібернація є одним із прикладів ORM. Коротше кажучи, JPA - це інтерфейс, а hibernate - це реалізація.

API persistence API надає специфікацію для збереження, читання та управління даними з вашого java-об'єкта до ваших реляційних таблиць в базі даних. JPA визначає набір правил і рекомендацій для розробки інтерфейсів, які відповідають стандартам. Відразу до суті: JPA - це лише рекомендації для реалізації ORM, і немає ніякого базового коду для реалізації. Spring Data JPA є частиною фреймворку Spring. Мета весняної абстракції сховища даних - значно зменшити кількість шаблонного коду, необхідного для реалізації рівня доступу до даних для різних сховищ персистентності. Spring Data JPA не є JPA-провайдером, це бібліотека/фреймворк, який додає додатковий рівень абстракції над нашою лінійкою JPA-провайдерів Hibernate.

Центральним інтерфейсом в абстракції сховища даних Spring Data є Repository. Він приймає доменний клас для керування, а також тип ідентифікатора доменного класу як аргументи типу. Цей інтерфейс діє насамперед як інтерфейс-маркер, щоб зафіксувати типи, з якими потрібно працювати, і допомогти вам знайти інтерфейси, які розширюють цей інтерфейс.

2.3.3. Spring Security

Фреймворк Spring Security - це компонент Spring, який забезпечує автентикацію, авторизацію та захист додатка. Завдяки першокласній підтримці захисту як імперативних, так і реактивних додатків, він є стандартом де-факто для захисту додатків на основі Spring. Даний компонент дозволяє вирішувати всі проблеми, які виникають при створенні додатків, і керує новим серверним середовищем для додатків.

Також до надання різних вбудованих опцій автентифікації та авторизації, Spring Security дозволяє нам налаштовувати процес автентифікації так, як ми цього хочемо. Починаючи зі спеціальної сторінки входу в систему і закінчуючи нашими власними постачальниками автентифікації та фільтрами автентифікації, ми можемо налаштувати кожен аспект процесу автентифікації. Ми можемо визначити наш власний процес автентифікації, який може варіюватися від базової автентифікації за допомогою імені користувача та пароля до складної, такої як двофакторна автентифікація за допомогою токенів та OTP. Також ми можемо використовувати різні бази даних - як реляційні, так і нереляційні, застосовувати різні шифратори паролів, блокувати акаунти зловмисників і так далі.

Перш за все, давайте подивимося на архітектуру Spring Security. Вона починається з сервлет-фільтрів. Ці фільтри перехоплюють запити, виконують над ними операції, а потім передають запити наступним фільтрам у ланцюжку фільтрів або обробникам запитів, або блокують їх, якщо вони не відповідають певним умовам.

Саме під час цього процесу Spring Security може аутентифікувати запити і виконувати різні перевірки автентичності запитів. Він також може запобігти доступу неавторизованих або зловмисних запитів до наших захищених ресурсів, не дозволяючи їм пройти. Таким чином, наш додаток та ресурси залишаються захищеними.

2.4. Специфікація OpenAPI. OpenAPI Generator

Специфікація OpenAPI (OAS) дозволяє описувати віддалений API, доступний через HTTP або HTTP-подібні протоколи. Цей опис, який може зберігатися у вигляді одного або декількох документів (наприклад, локальних файлів або мережевих ресурсів, доступних через HTTP), називається описом OpenAPI (OAD).

Інтерфейс прикладного програмування (API) визначає дозволену взаємодію між двома частинами програмного забезпечення, так само як користувацький інтерфейс визначає способи, якими користувач може взаємодіяти з програмою.

Інтерфейс прикладного програмування API складається зі списку можливих методів для виклику (запитів), їх параметрів, значень, що повертаються, і будь-якого формату даних, який вони вимагають (серед іншого). Це еквівалентно тому, як користувач взаємодіє з додатком для мобільного телефону обмежується кнопками, повзунками та полями в користувацькому інтерфейсі програми.

Файл опису API (іноді його називають контрактом) - це машинозчитувана специфікація API. Він повинен прагнути бути якомога повнішим і детальнішим, хоча абсолютна повнота зазвичай не є обов'язковою вимогою. Також, як і юридичні контракти, чим більш однозначною є специфікація, тим більш корисною вона стає.

Його головна перевага перед документацією, яку може прочитати лише людина, полягає в тому, що він дозволяє автоматизовану обробку, відкриваючи двері до переваг, перелічених на початку цього посібника.

OpenAPI Generator – це компонент, який на основі складеної специфікації взаємодії за OpenAPI специфікацією, дозволяє згенерувати код для подальшого використання в імплементації в програмі для опису доступних HTTP інтерфейсів. Опис доступних запитів робиться в окремому файлі YAML або JSON, на основі якого генератор створює інтерфейси для подальшої імплементації. Зазначена специфікація допомагає контролювати інтерфейс для взаємодії з іншими системи.

Фрагмент прикладу опису доступного запиту за специфікацією Open API вказаний нижче:

```yaml
paths:
  /cs-api/files/{fileId}:
    get:
      tags:
        - get-file-controller
      summary: Get file by id
      operationId: downloadFile
      parameters:
        - name: fileId
          in: path
          required: true
          schema:
            type: string
      responses:
        '200':
          description: OK
          content:
            multipart/form-data:
              schema:
                type: string
                format: binary
```

Рис. 2.7. Опис доступного запиту до сервера.

У файлі вказується HTTP метод, URL, перелік вхідних параметрів та тіло запиту і його формат. Генератор дозволяє уникнути дублікації та конфліктів з іншими доступними запитами, а також дозволяє провалідувати усі запити при препроцесингу ресурсів перед компіляцією та подальшими етапами збірки проекту.

2.5. Платформа Docker

Docker – це програмна платформа для розробки, тестування та розгорнення роботи програмного продукту в обмеженому робочому середовищі.

Докер відповідає за упаковування програмного забезпечення в стандартизовані контейнери. Ізольованість контейнерів дозволяє одночасно запускати та використовувати паралельно декілька контейнерів в рамках однієї робочої системи. Контейнери легкі і кожен з них включає все необхідне для повноцінної роботи контейнера та основного додатка в ньому (бібліотеки, код, системні інструменти та середвище для роботи).

Docker надає інструменти та платформу для керування життєвим циклом контейнерів:

- Розробка додатоку та його допоміжних компоненти, використовуючи контейнери.
- Контейнер стає одиницею для розповсюдження та тестування додатку;
- При завершенні розробки на певному етапі можливо розгорнути додаток у виробничому середовищі, як контейнер або організований сервіс. Це працює однаково, незалежно від того, чи є виробниче середовище локальним центром обробки даних, хмарним провайдером або їх гібридом.

Docker використовує архітектуру клієнт-сервер. Клієнт Docker взаємодіє з демоном Docker, який виконує необхідну роботу зі створення, запуску та розповсюдження ваших контейнерів Docker. Клієнт і демон Docker можуть працювати на одній системі, або можливо підключити клієнт Docker до віддаленого демона Docker. Клієнт і демон Docker взаємодіють за допомогою REST API, через сокети UNIX або мережевий інтерфейс. Ще одним корисним клієнтом Docker є Docker Compose, який дозволяє працювати з додатками, що складаються з набору контейнерів.

2.6. Висновок до розділу 2

В рамках другого розділу роботи було розглянуто та проведено аналіз використаних технологій для розробки системи сховища даних.

Для розробки компонентів системи використовується мова програмування Java, а також Spring Framework і відповідні програмні модулі. Це дозволяє реалізувати серверну частину кожного сервісу в системі. Серед основних переваг виділяють:

- *Легке використання*: Spring завдяки різним опціям конфігурації та конвенціям конфігурацій дозволяє розробникам легко розпочати роботу та налаштувати тільки те, що необхідне для роботи;
- *Модульність*: Spring володіє високомодульною природою, тобто має можливість використовувати лише необхідні модулі/компоненти для розробки додатка відповідного типу;
- *Підтримка специфікацій*: Spring підтримує усі специфікації Jakarta EE, а також в певних моментах покращує підтримку у порівнянні зі стандартом. Також присутня підтримка репозиторіїв на основі JPA, Spring Web Reactive, Reactive Stream та HATEOS;
- *Можливість тестування*: Spring за своєю суттю пропагує та підтримує Test Driven Development.

Також для повноцінної роботи системи використані хмарні сервіси від провайдера Amazon Web Services. Провайдер хмарних ресурсів AWS дає великий спектр різних сервісів, що надають інфраструктуру хмарних обчислень для розробки інформаційних систем. Було використано AWS EC2 для віртуальних серверів, AWS S3 для збереження об’єктів даних, AWS VPC для створення приватної віртуальної мережі, AWS RDS для конфігурації бази даних, а також AWS SQS для асинхронного обміну повідомленнями між мікросервісами системи.

Серед основних переваг виділяють:

- *Масштабованість*: AWS надає можливість легко масштабувати ресурси в залежності від потреб системи.
- *Надійність*: AWS забезпечує високий рівень доступності та надійності завдяки глобальним центрам по всьому світу та резервним механізмам.
- *Оплата за використання*: за використання хмарних послуг оплата здійснюється лише за фактичне використання ресурсів;
- *Варіативність сервісів*: AWS надає велику кількість сервісів, в які входять сервіси для створення на конфігурування баз даних, сервіси з експлуатації програмного забезпечення на них, сервіси для аналітики та мережеві сервіси.
- *Безпека*: Amazon Web Services надає стандартну та безпечну інфраструктуру з широким спектром послуг для забезпечення безпеки.

РОЗДІЛ 3
ПРОЕКТУВАННЯ ТА РОЗРОБКА СИСТЕМИ СХОВИЩА ДАНИХ

Хмарні сервіси для зберігання файлів останнім часом стали дуже популярними, оскільки вони спрощують зберігання та обмін цифровими ресурсами між різними пристроями. Вважається, що перехід від використання одного персонального комп'ютера до використання декількох пристроїв з різними платформами та операційними системами, таких як смартфони та планшети, а також портативний доступ до них з різних географічних місць у будь-який час є однією з основних причин величезної популярності хмарних сервісів зберігання.

Серед основних вимог та цілей системи виділяються основні, такі як підтримка можливості завантаження та вивантаження файлових даних користувача через надання можливості доступу до даних використовуючи публічно доступний API. Також система повинна підтримувати можливість отримання повідомлень про наявність оновлень стану сховища користувача для всіх пристроїв користувача.

Для оптимізації та зменшення трафіку між користувачем та системою необхідно надати механізми зберігання великих файлів з використанням фрагментування та контролю їх послідовності та цілісності на стороні сховища.

Система повинна дотримуватися гарантійних принципів ACID, що відповідає за атомарність, послідовність, ізольованість та довговічність для всіх операцій з файлами та їх фрагментами.

3.1. Принцип роботи системи

Як і будь-яка інша система, наше сховище даних повинно мати серверну частину. Для реалізації архітектурної моделі системи була вибрана мікросервісна архітектура, яка характеризується наявністю декількох окремих сервісів, кожен з яких знаходиться на окремому сервері та має своє унікальне функціональне призначання.

Перевага мікросервісної архітектури над монолітним сервісом полягає у можливості функціональному та технічному виділенні окремих послуг системи. Це дає змогу покращити горизонтальне та вертикальне масштабування окремого сервісу, а також можливість реалізації окремих сервісів різними командами на різних мовах програмування в залежності від функціонала сервісу та оптимізувати розподілення ресурсів і обслуговування сервісу.

Перед проєктуванням архітектурного рішення та реалізації серверної частини системи необхідно визначити функціональні вимоги до системи, серед яких можна виділити:

- Користувач повинен мати можливість завантажувати файлові дані.
- Користувач повинен мати можливість створювати/видаляти директорії та зберігати їх в системі.
- Система повинна надавати можливість синхронізувати дані між усіма пристроями користувача.
- Користувач повинен мати можливість завантажувати дані, навіть якщо пристрій знаходиться в певний момент не має доступу до мережі. Як тільки доступ до інтернету з'явиться, файли, що знаходяться в хмарному сховищі, мають буди синхронізовані з ним.

Концептуальна модель системи складається з трьох основних сервісів, серед яких файловий сервіс, сервіс по роботі з метаданими та сервіс для синхронізації стану сховищ на різних пристроях користувача. Для зберігання та консистенції метаданих

по файлах використовується реляційна база даних, а для зберігання файлів, а точніше фрагментів файлів, використовується об’єктне сховище.

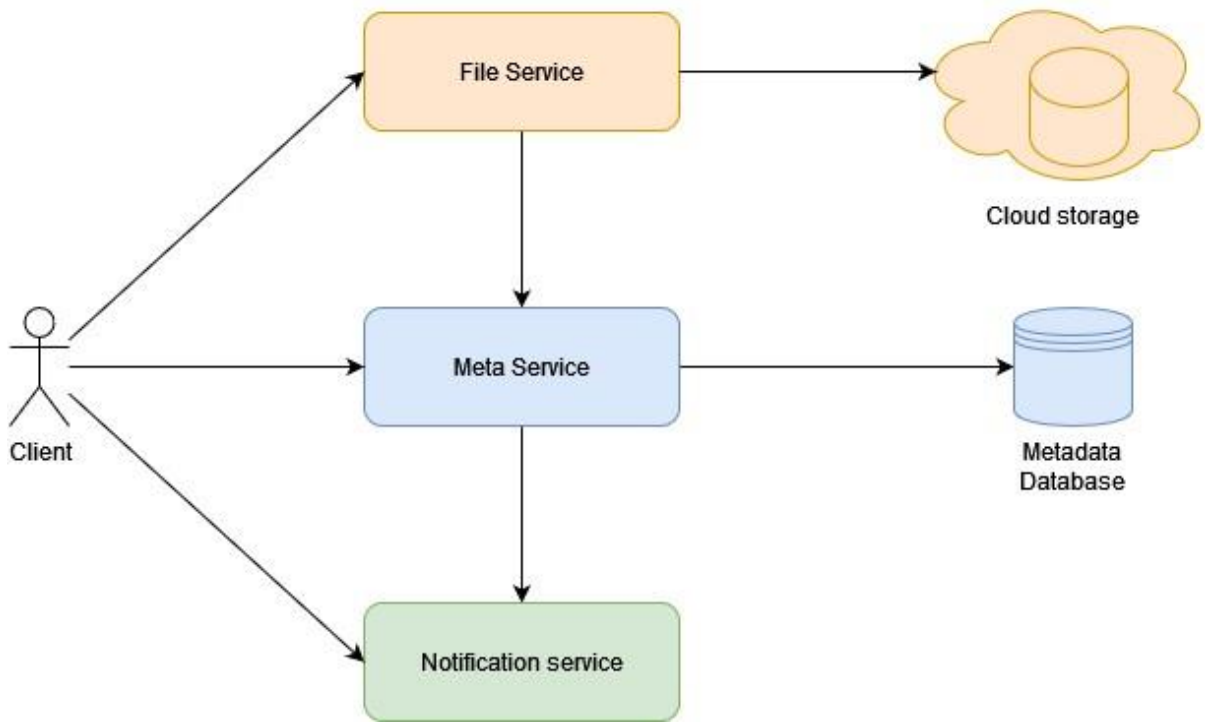

Рис. 3.1. Концептуальна модель системи

Файловий сервіс, який пов’язаний з об’єктним сховищем та мета сервісом, відповідатиме за обробку файлів та їх фрагментів користувача системи та керування сховищем об’єктів. Мета сервіс відповідає за контроль метаданих файлів користувача, а також інші сутності, які пов’язані з сховищем даних. Сервіс синхронізації забезпечує інформацією про зміни стану сховища у хмарі для інших пристрої користувача, що дає змогу цим пристроям підтянути зміни по файлам з хмари. Це забезпечує консистентність стану локальних файлів на пристроях користувача та надає можливість отримувати та використовувати актуальні дані.

3.2. Архітектура системи хмарного сховища даних

Як і раніше було вказано раніше, основу концептуальної моделі системи складає сервіс по роботі з файлами та їх фрагментами, сервіс по роботі з метаданими файлових даних, а також сервіс синхронізації або інформування про зміни в хмарному сховищі. До цієї моделі також входять база даних для збереження метаданих і об’єктне сховище.

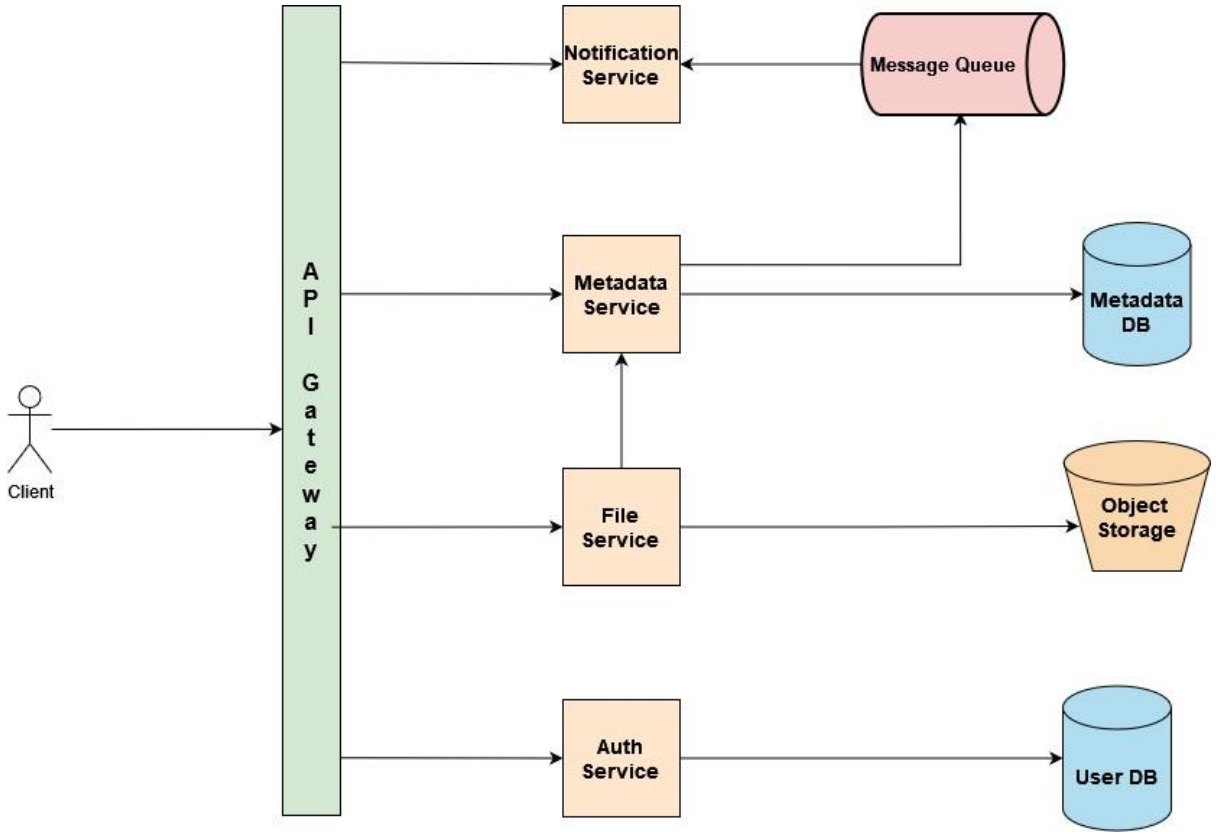

Рис. 3.2. Діаграма компонентів системи сховища даних

До початкової моделі додаються сервіс авторизації та аутентифікації користувачів для забезпечення контролю доступу до даних кожного окремого зареєстрованого користувача системи. Перевірка користувача здійснюється для усіх запитів шляхом валідації JWT токена (JSON Web Token), що є відкритим стандартом для безпечної передачі інформації між сторонами у вигляді JSON-об’єктів.

Отримання токена відбувається на початковому етапі коли користувач входить в систему або намагається отримати доступ до захищеного ресурсу, сервер генерує JWT після успішної автентифікації. Потім клієнт зберігає цей маркер, зазвичай у локальному сховищі або файлі cookie. Для кожного наступного запиту, який вимагає автентифікації, клієнт надсилає JWT в заголовках запиту. Сервер перевіряє токен, перевіряючи підпис і розшифровуючи корисне навантаження, щоб переконатися в аутентифікації та авторизації користувача.

Для зв’язку між метасервісом та сервісом нотифікації додається черга повідомлень, яка дозволяє забезпечити асинхронний зв’язок між цими сервісами для передавання повідомлень. У метасервісу не буде потреби очікувати відповіді про успішно опрацьований запит, що потребує синхронного взаємозв’язку.

Також додається єдина точка входу до системи – API Gateway. Це API шлюз, який відповідає за маршрутизацію запиту до відповідного сервісу в залежності від URL запиту, вхідних параметрів, HTTP методів чи інших можливих ідентифікаторів запиту. За цими ознаками даний компонент системи вирішує на який сервіс відправити вхідний запит. Цей додатковий пласт між клієнтом та кількома сервісами дає змогу повністю ізолювати систему та надати користувачу єдину точку входу для усіх можливих запитів.

3.2.1. Розроблення бази даних системи

Оскільки система використовує об'єктне сховище для збереження файлів у вигляді фрагментів, то виникає потреба у збереження мета інформації при операціях збереження файлів та збереженні усіх інших змін над ними.

На малюнку №3.5 наведена реляційна модель бази даних, яка складається з шести взаємопов'язаних сутностей:

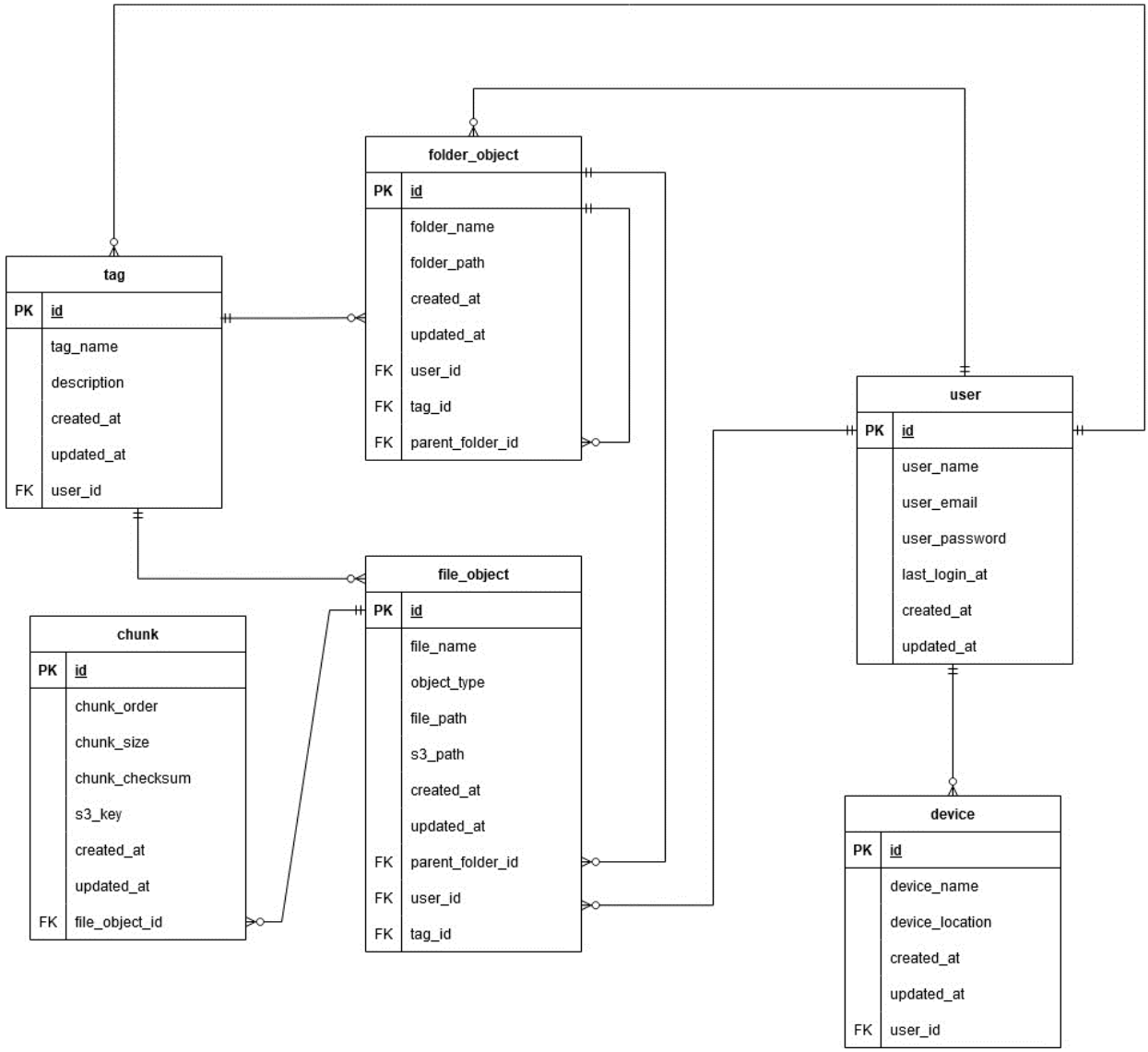

Рис.3.3. Реляційна модель бази даних

Серед наявних сутностей схеми бази даних виділено:

- Фрагмент файлу (таблиця *chunk*);
- Файл (таблиця *file_object*);
- Директорія (таблиця *folder_object*);
- Користувач (таблиця *user*);
- Пристрій (таблиця *device*);
- Тег (таблиця *tag*).

Таблиця «*chunk*» відповідає за збереження інформації про фрагмент файлу. Кожен фрагмент пов'язаний з сутністю файл відношенням багато до одного (N : 1), відповідно кожен файл має один або більше пов’язаних фрагментів. Кожен з цих фрагментів є частиною файлу. Серед полів сутності можна виділити:

- *chunk_order* – порядок фрагмента. Зберігає інформацію про порядковий номер фрагмента файлу, що дає змогу відтворити початковий файл з його фрагментів, які знаходяться в AWS S3 сховищі.
- *chunk_size* – розмір фрагмента. Максимальний розмір фрагмента сягає 4 MB.
- *chunk_checksum* – контрольна сума фрагмента, отримується за допомогою CRC32 (Cyclic Redundancy Check 32) алгоритму та використовується для перевірки неушкодженості фрагмента файлу.
- *s3_key* – строкове поле, яке зберігає ключ до фрагмента файлу. Використовується для доступу до фрагмента файлу, який знаходиться всередині AWS S3 сховища.

Таблиця «*file_object*» зберігає відповідну метаінформацію про файл, фрагменти якого знаходяться в AWS S3 сховищі. Таблиця пов’язана відношенням один до багатьох (1 : N) з фрагментами файлу, а також багато до одного (N : 1) з сутністю директорія через поле “*parent_folder_id”*.

Серед полів сутності можна виділити:

- *file_name* – повна назва файлу разом з його типом.
- *file_path* – шлях до файлу відносно директорії клієнта, яка синхронізується з даними, що зберігаються в системі.
- *s3_path* – префікс до ключів фрагментів файлу. Структурно є частиною ключа до фрагмента файлу без останньої частини, яка ідентифікує унікальний фрагмент.

Таблиця «*folder_object*» відповідає за збереження інформації про директорії, які відповідають тим, які знаходяться у користувача на локальній машині. Директорія

пов’язана відношенням один до багатьох (1 : N) з сутністю файл, оскільки в рамках однієї директорії може бути декілька файлів, а також у відношенні з самою собою, оскільки може містити декілька інших директорій або бути дочірньою до іншої.

У таблицю «*user*» зберігається інформація про користувача, а також через зв’язок один до багатьох (1 : N) пов’язаний з таблицями «*file_object*» та «*folder_object*», що дає змогу робити вибірку доступних об’єктів в сховищі тільки для пов’язаного з ними користувачем.

У таблицю «*device*» зберігається інформація про пристрої, якими володіє користувач. При умові змін стану сховища через зміну стану на пристрої користувача системі необхідно буде перевірити наявність інших пристроїв. У разі наявності таких буде відправлене повідомлення у сервіс синхронізації (notification service), в якому буде вказано зміни в стані сховища користувача та за якими клієнтський додаток на пристрої зможе узгодити стан локального сховища зі сховищем у хмарі через відповідні дії.

Таблиця «*tag*» зберігає інформацію про теги, які використовуються для зв’язування з файлів та директорій між собою за певною назвою. Виступає в ролі маркування.

Для збереження метаінформації використовується реляційна база даних з підтримкою властивостей ACID (атомарність, узгодженість, ізоляція, довговічність) і створено схему відповідну моделі сутностей. Серед доступних вибрано RDBMS PostgreSQL. Для роботи в хмарі використовується хмарний сервіс AWS RDS, який дозволяє сконфігурувати та запустити базу даних у хмарі без необхідності інфраструктурному налаштовані з боку користувача. До БД можливо під’єднатися за реквізитами для входу, які вказані будуть користувачу при закінченні конфігурування бази даних.

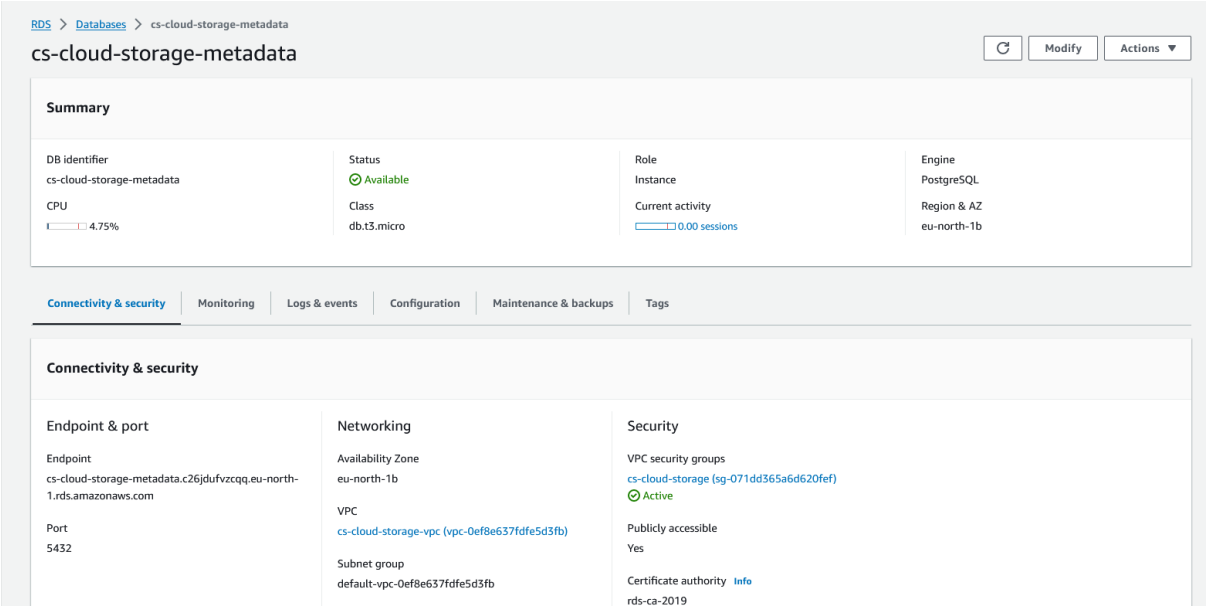

Рис. 3.4. Конфігурація бази даних в AWS RDS сервісі

Взаємозв'язок з базою даних доступний тільки сервісу «*Meta Service*», який вносить відповідні зміни в стан бази даних з відповідальністю за транзакційні та консистентності операцій при обробці мета інформації за бізнес-логікою пов'язаною з моделлю даних в системі.

3.2.2. Структура збереження фрагментів даних в системі

Збереження файлів відбувається у вигляді окремих фрагментів, що зберігаються всередині об'єктного сховища сервісу AWS S3, і мають максимальний розмір в 4MB.

Amazon S3 - це сховище об'єктів, яке використовує унікальні ключові значення для зберігання будь-якої кількості об'єктів. Ці фрагменти зберігаються у вигляді об'єктів у S3 контейнер (bucket), де зберігаються усі фрагменти файлів для усіх користувачі системи, прив'язуючи до нього відповідний ключ. Цей ключ зберігається у базі даних та використовується для отримання доступу до самого фрагмента файла при запитах від користувача.

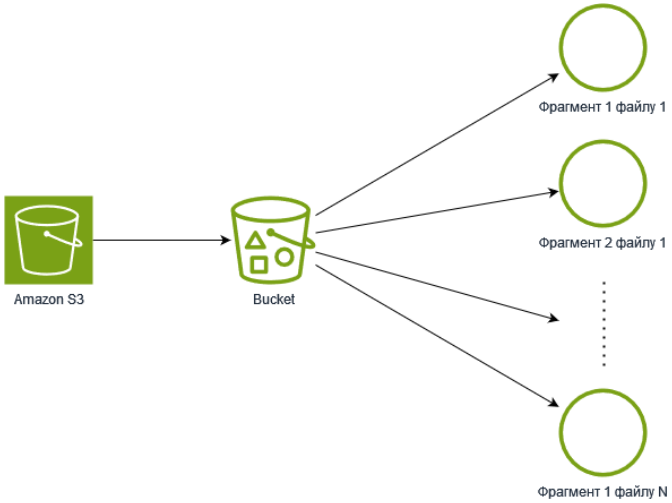

Рис. 3.5. Фрагментнти файлів всередині сховища.

Сформований S3-ключ для збереженя фрагменту та подальшої роботи з ним складається з наступних частин, які розділяються знаком «/» та розташовані у наступному порядку:

1. Остача ділення результату хеш функції ідентифікатора користувача на тисячу, поділена на 10;
2. Остача ділення результату хеш функції ідентифікатора користувача на десять;
3. Ідентифікатор користувача;
4. Ідентифікатор файлу;
5. Ідентифікатор фрагменту файлу.

Приклад ключа, який використовується файловим сервісом для отримання доступу до фрагменту файлу:

Рис. 3.6. Приклад S3-ключа.

Такий композитний ключ дає прийнятне партиціювання збережених фрагментів файлів що веде до підвищення швидкіскості пошуку об’єкта в об’єтному сховищі. Аргументовано це механізмами AWS S3, що дозволяють пришвидшити цей пошук за ключем.

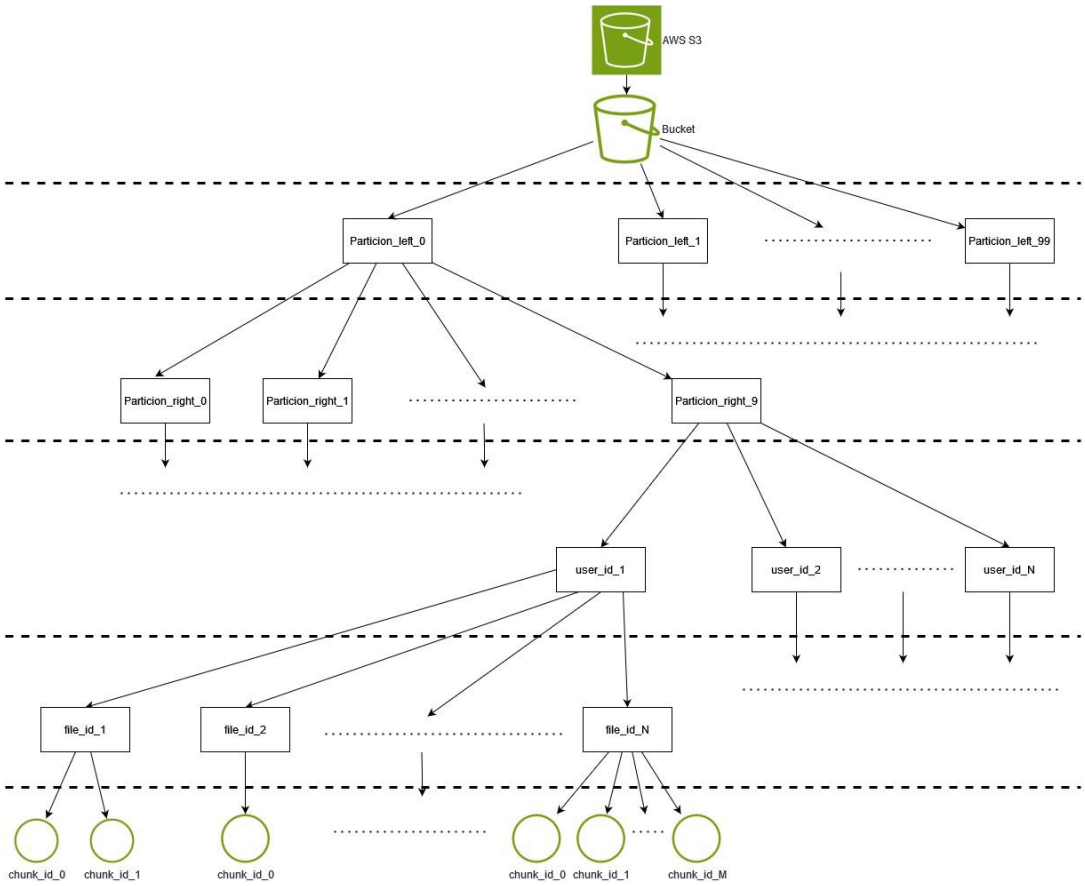

Рис. 3.7. Структура збереження фрагментів за композитним ключем

3.2.3. Реалізація сервісу «File Service»

Сервіс «*File Service*» в рамках системи сховища даних виконує основне завдання по обробці файлових даних користувача. Даний сервіс являє собою вебсервер з відповідним переліком доступних запитів до нього. Публічний API цього сервісу реалізований за RESTful API специфікацією. Доступ до сервісу здійснюється через єдиний шлюз доступу (API Gateway) до системи, як і інші активні сервіси.

Для роботи сервіс використовує хмарний сервіс для збереження об’єктів від Amazon Web Services – AWS S3. Цей хмарний сервіс провайдера дозволяє зберігати будь-які об’єкти та отримувати доступ до них використовуючи пов’язаний з об’єктом ключ. Для зв’язку з сервісом реалізований відповідний функціонал використовуючи бібліотеку AWS SDK, що дає змогу виконувати запита до різних сервісів Amazon. Для роботи з AWS S3 необхідно вказати конфігураційні дані, що дозволить роботи запити до нього, а також реалізувати методи для роботи з ним. Приклад конфігурації клієнта для роботи з AWS S3 вказаний у додатку А, а приклад реалізації методів для роботи з сервісом знаходиться у додатку Б.

Сервіс дозволяє виконати наступні запити:

- Завантаження файлу або кількох файлів;
- Завантаження фрагмента або кількох фрагментів файлу з вказанням позиції фрагмента та ідентифікатора файлу;
- Скачування файлу або декількох файлів як архів;
- Скачування фрагмента або кількох фрагментів файлу як архів;
- Видалення файлу та фрагменту файлу;
- Видалення кількох файлів та кількох фрагментів;
- Копіювання файлу або кількох файлів між директоріями.

Сервіс пов'язаний з мета сервісом, з якого файловий сервіс отримує мета інформацію, яка пов'язана з файловими даними, фрагментами файлових даних та інформацію про їх збереження в хмарному сервісі AWS S3. Взаємодія клієнта з цим сервісом здійснюється через API Gateway, який виступає у ролі шлюзу для системи в цілому. В будь-якому HTTP запиті від користувача після валідації JWT токену проходить добавлення у заголовку HTTP-повідомлення ідентифікатора користувача, що дає змогу проводити зміни тільки над відповідними файловими даними користувача.

Нижче наведена модельна діаграма компонентів файлового сервісу та зв’язок з іншими складовими системи:

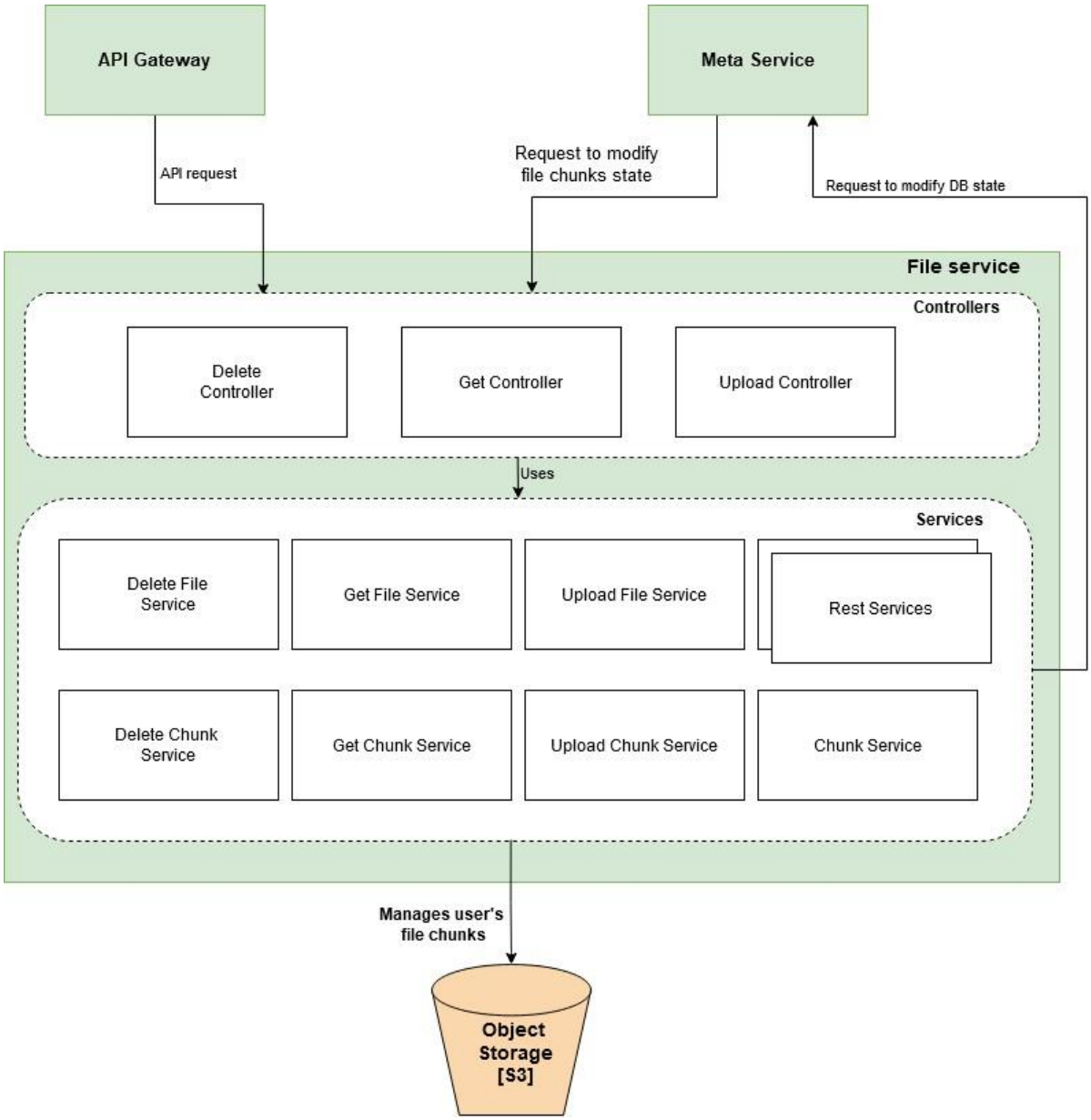

Рис. 3.8. Компоненти файлового сервісу

3.2.4. Реалізація сервісу «Meta Service»

Сервіс «*Meta Service*» відповідає за контроль метаданих файлів користувача та їх фрагментів. Доступ до сервісу, як і до інших сервісів, здійснюється через єдину точку доступу до системи – API Gateway.

Сервіс дозволяє виконати наступні запити:

- CRUD операції та можливість переміщення над метаданими файлу;
- CRUD операції наж метаданими фрагменту файлу;
- CRUD операції та можливість переміщення над директоріями користувача;
- CRUD операції над даними користувача;
- CRUD операції над пристроями користувача;
- CRUD операції по маркуванню сутностей;
- Пошук сутностей по ідентифікаторам або по шаблонам їх назви.

Мета сервіс підтримується базою даних, що зберігає інформацію про усі сутності в системі. База даних відповідає за збереження та контроль метаданих про файли і їх фрагменти, а також за дані, що пов'язані з директоріями, даними користувачів та їх пристроям.

Також відповідає за з'ясування набору змін для різних клієнтів користувача і трансляцію їх до сервісу сповіщення. Якщо клієнт оновлює файл, мета-сервіс знову визначає набір змін для інших клієнтів, які переглядають цей файл, і транслює цей набір змін через службу сповіщень.

Сервіс при будь-яких змінах стану сховища користувача відправляє відповідне повідомлення про зміни до хмарного сервісу AWS SQS, який є FIFO чергою, що збергіє у порядку надходження повідомлення. Можливість використання хмарного сервісу AWS SQS забезпечується бібліотекою AWS SDK.

Нижче наведена модельна діаграма компонентів файлового сервісу та зв’язок з іншими складовими системи:

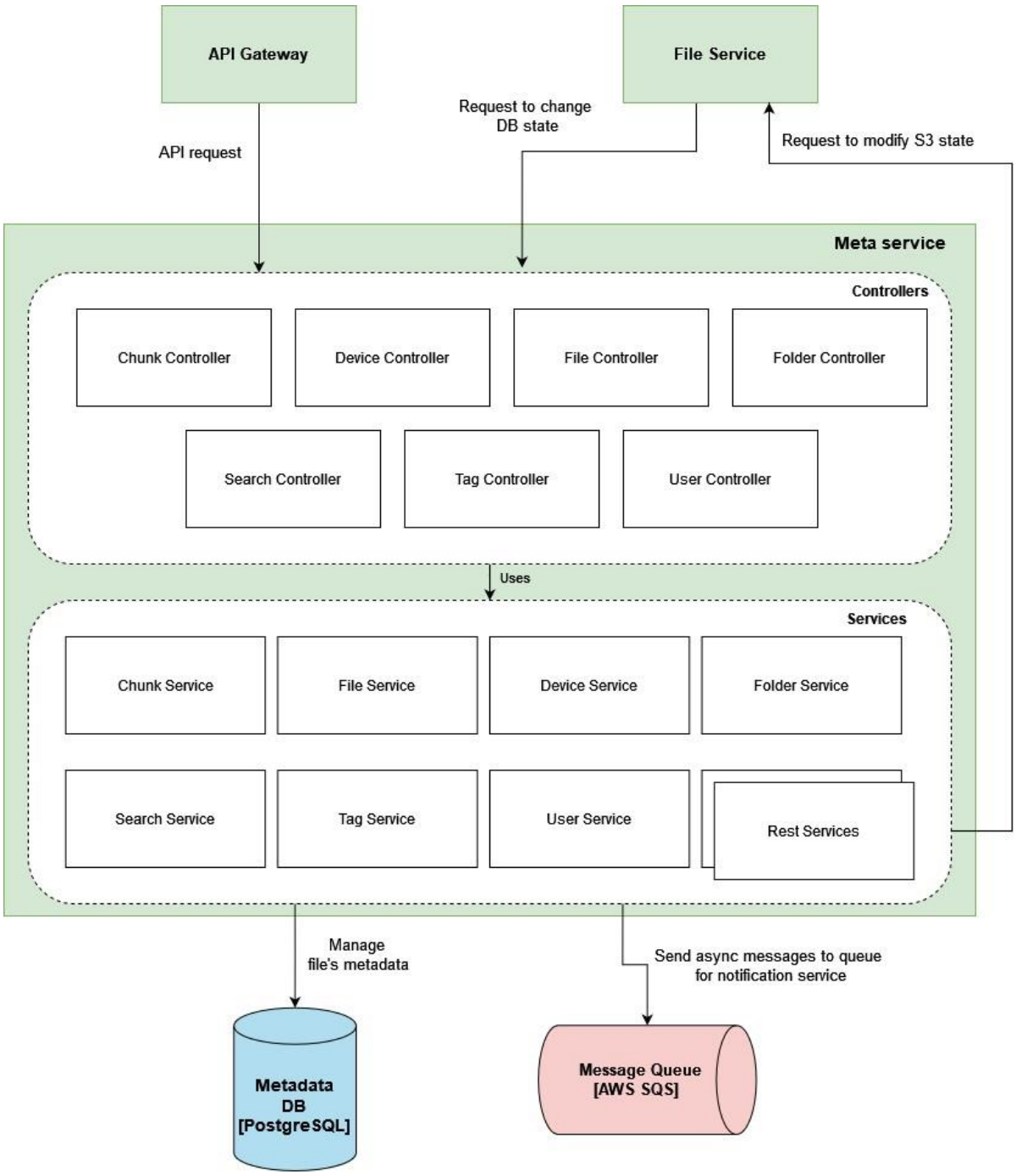

Рис. 3.9. Компоненти мета сервісу

3.2.5. Реалізація сервісу «Notification Service»

Сервіс «Notification Service» відповідає за забезпечення інформацією про зміну стану хмарного сховища користувача транслюючи зміни про стан сховища

підключеним клієнтам, гарантуючи, що будь-яка зміна у файлі буде миттєво відображена на всіх клієнтах, що спостерігають за ним.

Сервіс дозволяє іншому клієнту отримати інформацію про зміни в сховищі, наприклад, коли були якісь зміни на одному пристрою, то через цей сервіс інший пристрій, при наявності таких, може зчитати ці зміни та синхронізувати локальне сховище з тим, що знаходиться у хмарі.

Зв’язок з сервісом доступний через єдиний HTTP запит, який в результаті повертає інформацію про зміни, що далі може бути використана для синхронізації інших локальних сховищ клієнта. Повідомлення мають інформацію про зміни і асинхронно зчитуються сервісом і агрегуються усі отримані зміни, ніформацію про яких при потребі сервіс зберігає в локальному кеші.

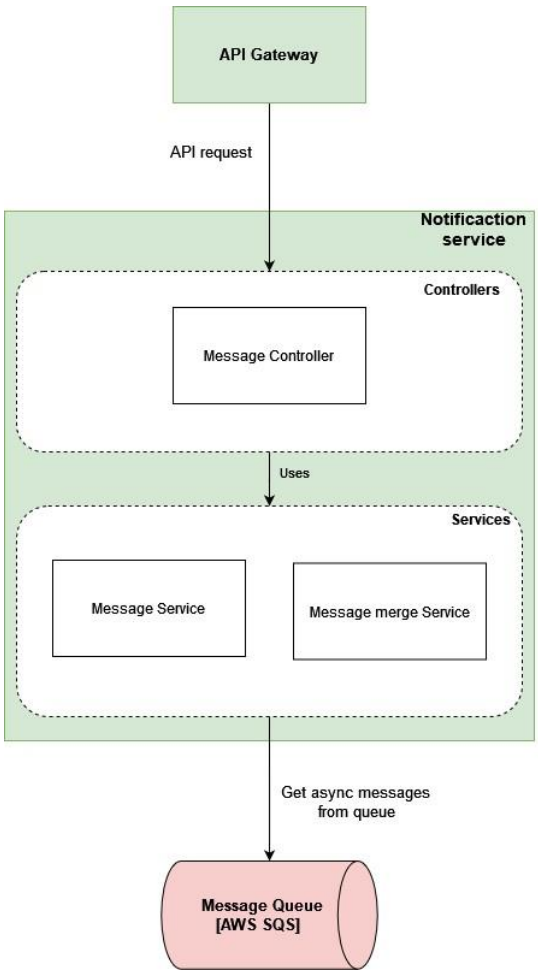

Рис. 3.10. Компоненти сервісу синхронізації

Служба сповіщень може бути реалізована за допомогою довгого опитування HTTP, вебсокетів або подій, що надсилаються сервером. Веб сокети встановлюють постійне дуплексне з'єднання між клієнтом і сервером. Це не найкращий вибір у цьому сценарії, оскільки немає потреби у двосторонньому зв'язку. Нам потрібно лише транслювати повідомлення від сервісу до клієнта, а отже, це надмірний функціонал від протоколу.

HTTP Long Polling є найкращим серед інших можливих варіантів вибором, оскільки сервер тримає з'єднання в підвішеному стані до тих пір, поки дані не будуть доступні для кінцевого клієнта. Як тільки дані стають доступними, сервер відправляє їх, закриваючи з'єднання. Після закриття з'єднання клієнт повинен знову встановити нове з'єднання. Як правило, для кожного довгого запиту на опитування існує тайм-аут, і клієнт повинен встановити нове з'єднання після закінчення тайм-ауту.

У режимі надсилання подій сервером клієнт встановлює довготривале постійне з'єднання з сервером. Це з'єднання використовується для надсилання подій від сервера до клієнта. Тайм-аут відсутній, і вміст залишається активним, поки клієнт залишається в мережі. Це ідеально підходить для нашого випадку використання і буде гарним вибором для розробки служби сповіщень. Хоча події, що надсилаються сервером, підтримуються не всіма браузерами, це не є проблемою для нас, оскільки у нас є спеціальні десктопні та мобільні клієнти, де ми можемо використовувати цю функцію.

Служба сповіщень перед відправленням даних клієнтам зчитує повідомлення з черги повідомлень. Для роботи вибраний хмарний сервіс AWS SQS, але ця черга може бути реалізована також за допомогою RabbitMQ, Apache ActiveMQ або Kafka. Черга повідомлень забезпечує асинхронне середовище зв'язку між метасервісом та сервісом сповіщень і, таким чином, мета сервісу не потрібно чекати, поки сповіщення буде надіслано клієнтам. Служба сповіщень може продовжувати споживати повідомлення у власному темпі, не впливаючи на продуктивність мета сервісу. Таке розділення також дозволяє нам масштабувати обидва сервіси незалежно.

3.3. Доступний функціонал системи сховища даних

Доступний користувачу функціонал, яким володіє система, можна розділити на ті, які включають роботу лише з метаданих і їх зміну, а також на той функціонал, що включає зміни метаданих та фрагментів файлів, які знаходяться в об’єктному сховищі.

Серед основних функцій, що включають зміну стану як об’єктного сховища, так і бази даних, можна виділити:

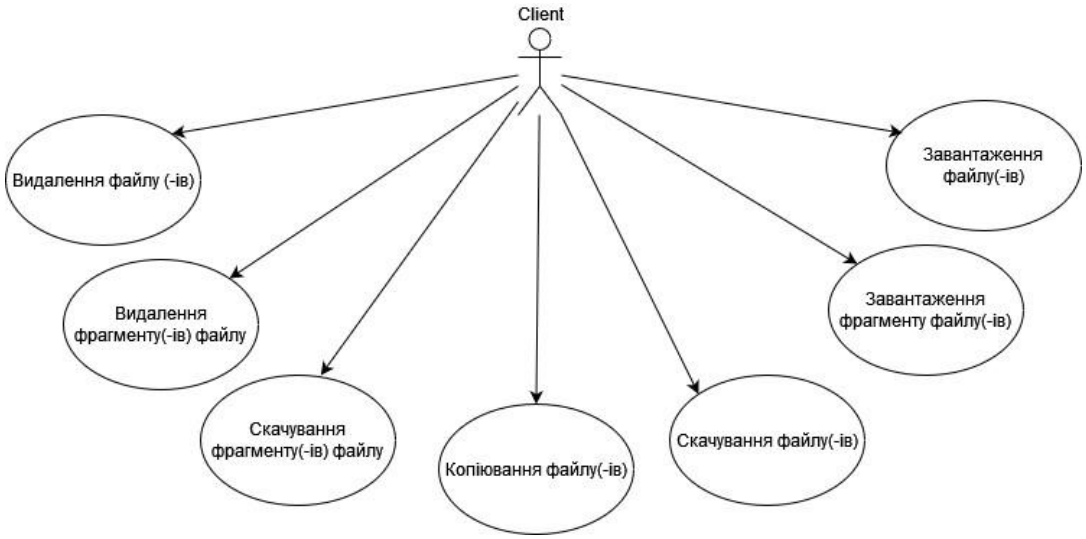

Рис. 3.11. Діаграма варіантів використання

Функціонал, який вимагає зміни як в об'єктному сховищі, так і в базі даних вимагають використання файлового та мета сервісів. Основну обробку цих варіантів використання виконуються файловим сервісом, який виконує зміну в AWS S3 контейнері, а за зміни в базі даних відповідає мета сервіс, що робить відповідна запити в базу даних.

При будь-яких змінах стану бази даних відбувається відправлення повідомлення у чергу з інформацією про зміни в сховищі.

Варіанти використання, що включають тільки роботу з мета сервісом та пов’язаною з нею базою даних:

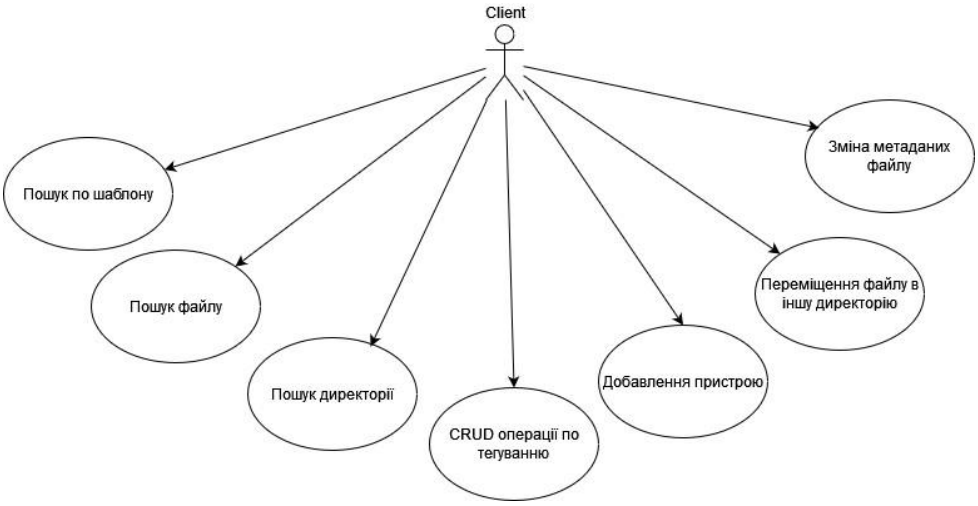

Рис. 3.12. Діаграма варіантів використання

3.4. Сценарії роботи хмарного сховища

Реалізоване хмарне сховище дає можливість взаємодіяти з ним через виконання HTTP-запитів. Для тестування коректності роботи можна використовувати інші мови програмування або бібліотеки для формування запитів, що дає змогу реалізувати клієнтську частину для користування системою. Для цього використано API-платформу Postman, яка має широкий функціонал для взаємодії з API, які доступні за різними специфікаціями та мережевими протоколами, в тому числі дає можливість формувати та виконувати HTTP запити використовуючи зручний інтерфейс.

3.4.1. Завантаження файлу

Процес завантаження та збереження файлових даних включає усі компоненти системи. Після того як клієнт встановив HTTP з’єданння та відправив запит для завантаження нового файлу в систему, повідомлення, отримане API-шлюзом з додатковим процесом аутентифікації користувача, ретранслюється на файловий сервіс. При обробці відбувається валідація вхідного файлу та перевірка на наявність супутніх даних.

Файл буде фрагментований на декілька частин для подальшого збереження у хмарному сховищі сервісу AWS S3. Дані про файл та його фрагменти відправляються запитом на мета-сервіс, який відповідно збереже ці дані в базу даних. Також мета сервіс відправить повідомлення у чергу для інших пристроїв користувача, якщо такі наявні. Після отримання підтвердження від сервісу про збереження, файловий сервіс збереже фрагменти як об’єкти у хмарне сховище з прив’язкою до ключів, за якими потім можна отримати доступ до них.

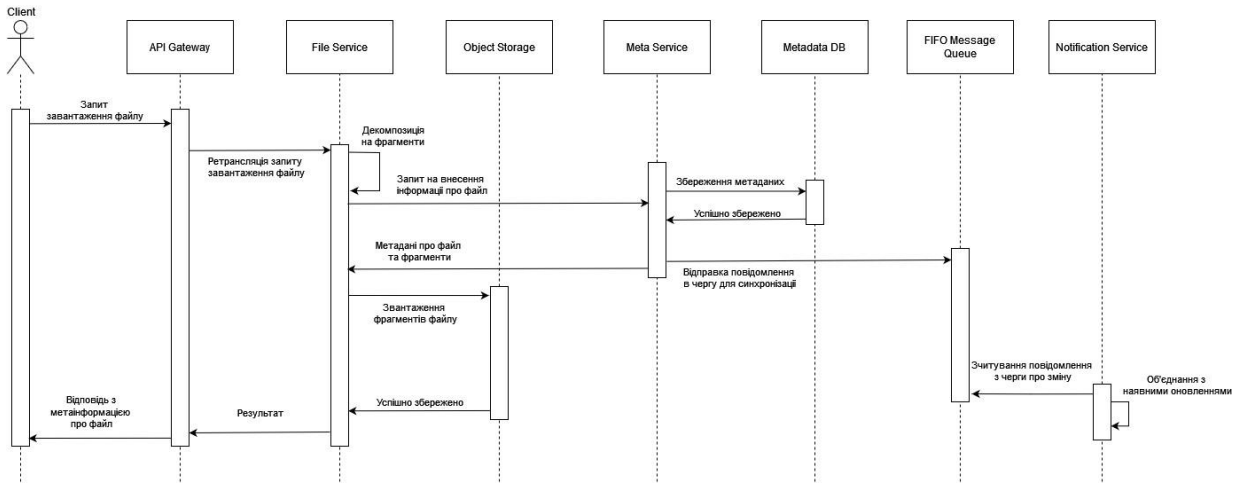

Рис. 3.13. Діаграма послідовностей для завантаження файлу

Здійсненя запиту до системи для завантаження файлу здійснюється за наступним запитом:

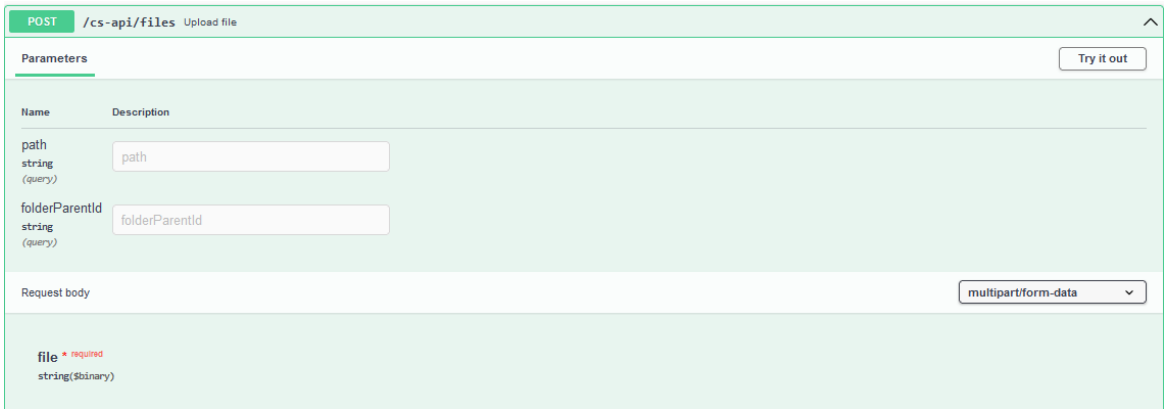

Рис. 3.14. Запит на завантаження файлу в Swagger-ui

Для прикладу було відправлено файл з розміром менше чотирьох мегабайт, що призведе лише на формування одного фрагменту файлу, тобто в хмарний сервіс AWS S3 буде збережено лише один об'єкт, що повністю відповідає початковому цілому файлу.

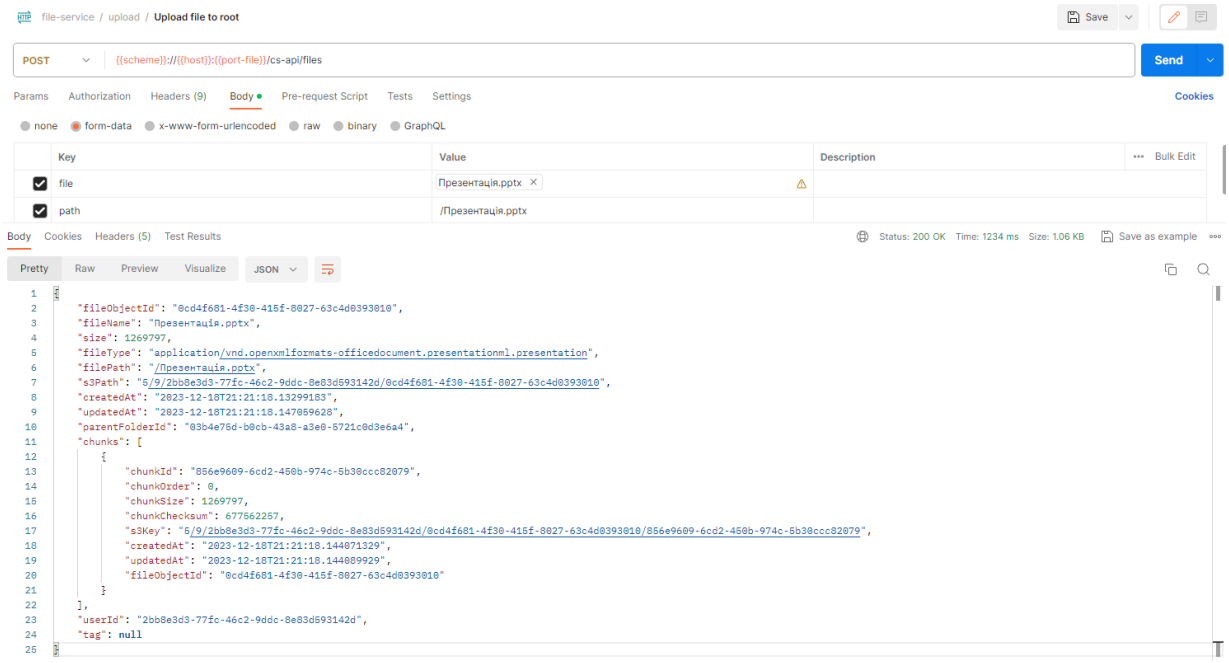

Рис. 3.15. Приклад запиту на збереження

У вікні нижче знаходиться результат виконання запиту до сховища, що може в подальшому бути використаний клієнтським додатком для інших операцій над цим файлом. У результаті ми отримуємо мета інформацію про збережений файл, його фрагмент, а також інші дані про місце знаходження, дати, розмір фрагменту, ідентифікатор користувача, тощо.

У результаті виконання в об'єктному сховищі AWS S3 збережено єдиний фрагмент файлу з відповідним ключем.

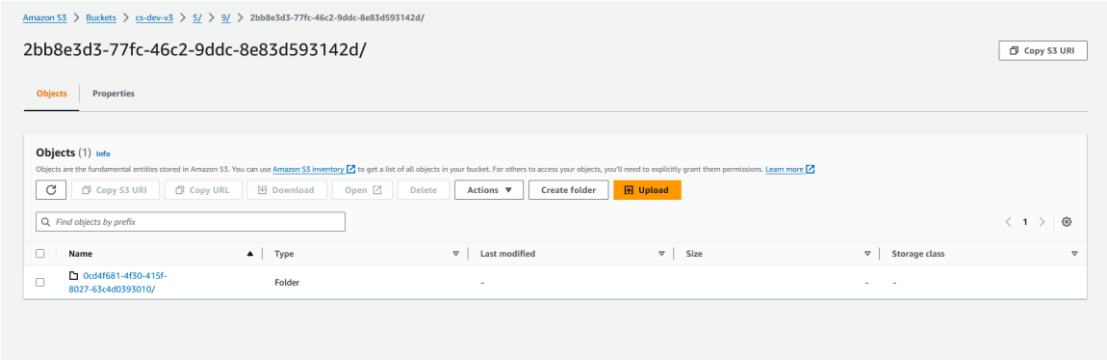

Рис. 3.16. Збережений фрагмент файла у сховищі.

3.4.2. Скачування файлу

Запит на скачування файлу потребує передавання тільки ідентифікатора файлу, що дозволить знайти дані про нього в системі та в результаті повернути файл, який нас цікавить.

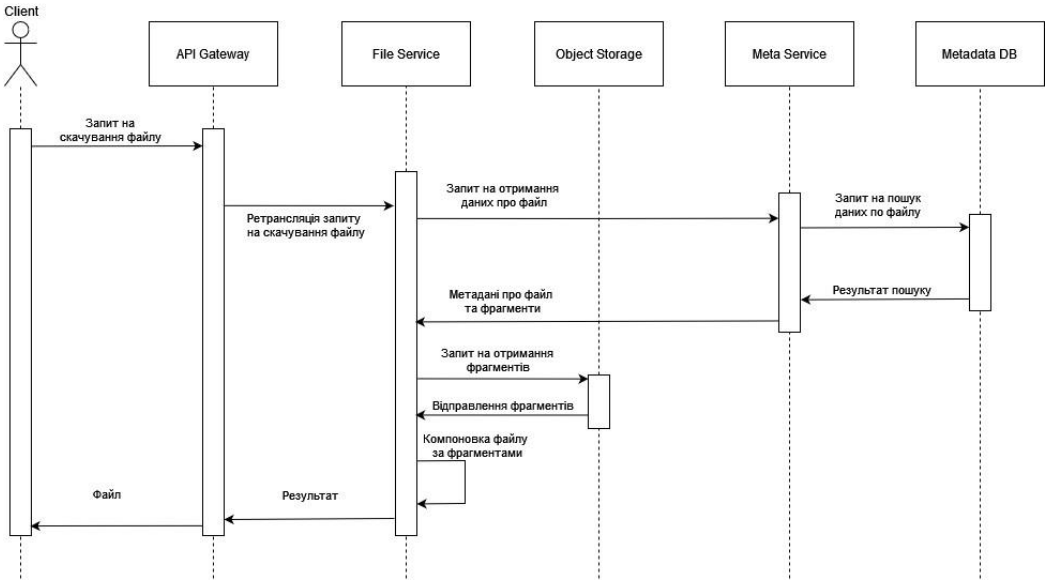

Рис. 3.17. Діаграма послідовностей для скачування файлу

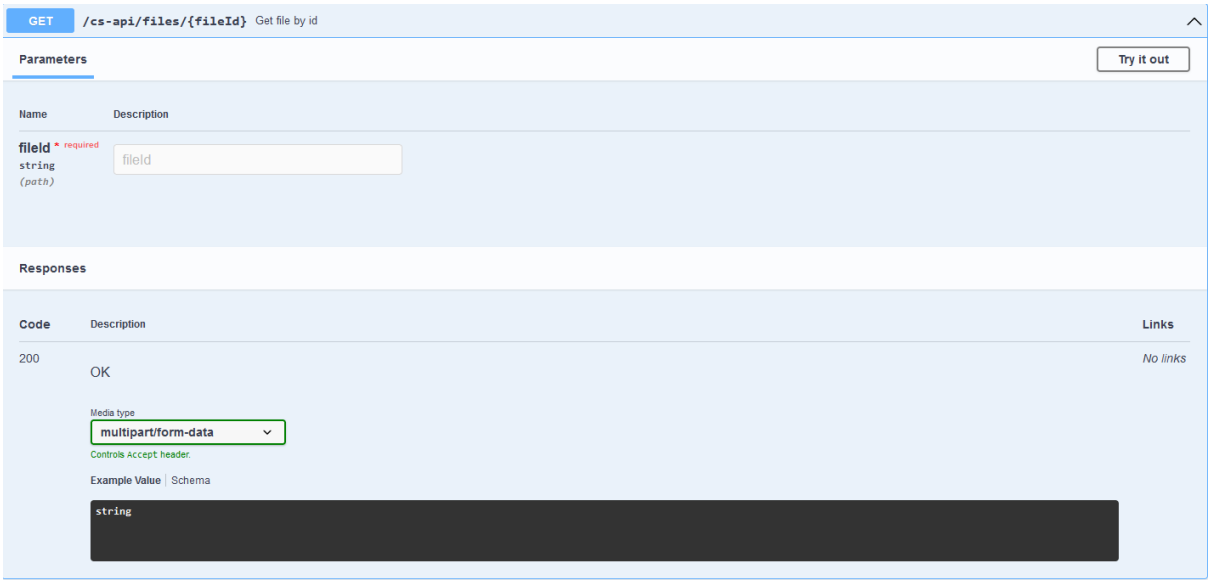

Рис. 3.18. Запит на скачування файлу в Swagger-ui

Після проходження API Gateway та ретрансляції запиту до файлового сервісу, при обробці запиту буде надісланий відповідний запит до мета сервісу на пошук інформації про файл за переданим ідентифікатором.

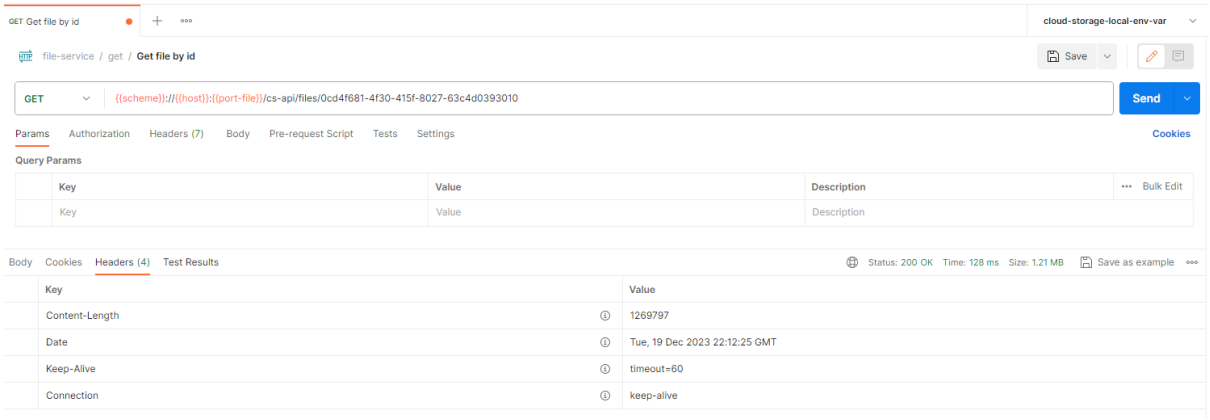

Рис. 3.19. Результат виконання запиту скачування файлу

В результаті виконання запиту отримуємо файл, фрагменти якого були скомпоновані відіслані як цілий файл. Розмір файлу та його зміст відповідає початковому файлу, який ми завантажували.

3.4.3. Видалення файлу

Видалення відбувається через відповідний запит, де ми передаємо ідентифікатор файла, який хочемо видалити з системи.

Спершу інформація буде видалена з бази даних через метасервіс. Після успішного видалення інформації про файл клієнт отримає відповідь про успішне виконання запиту. Паралельно з цим файловий сервіс асинхронно виконає видалення фрагментів файлу з об’єктного сховища.

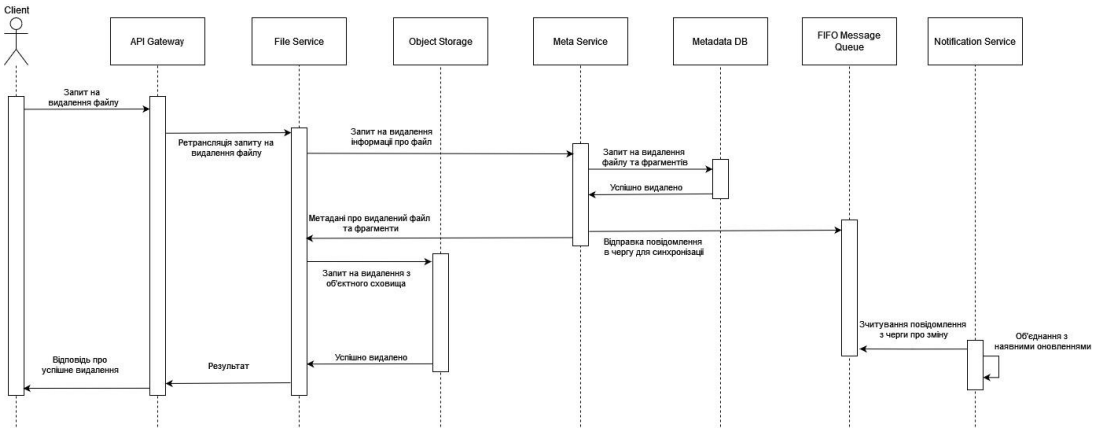

Рис. 3.20. Діаграма послідовностей для видалення файлу

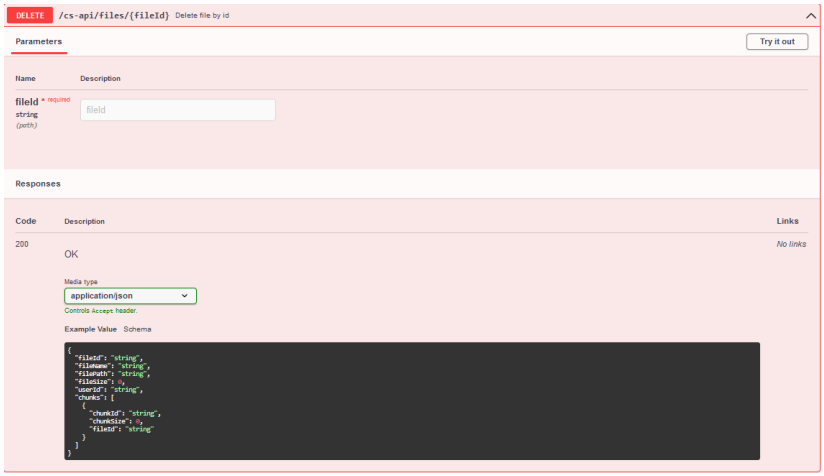

Рис. 3.21. Запит на видалення файлу в Swagger-ui

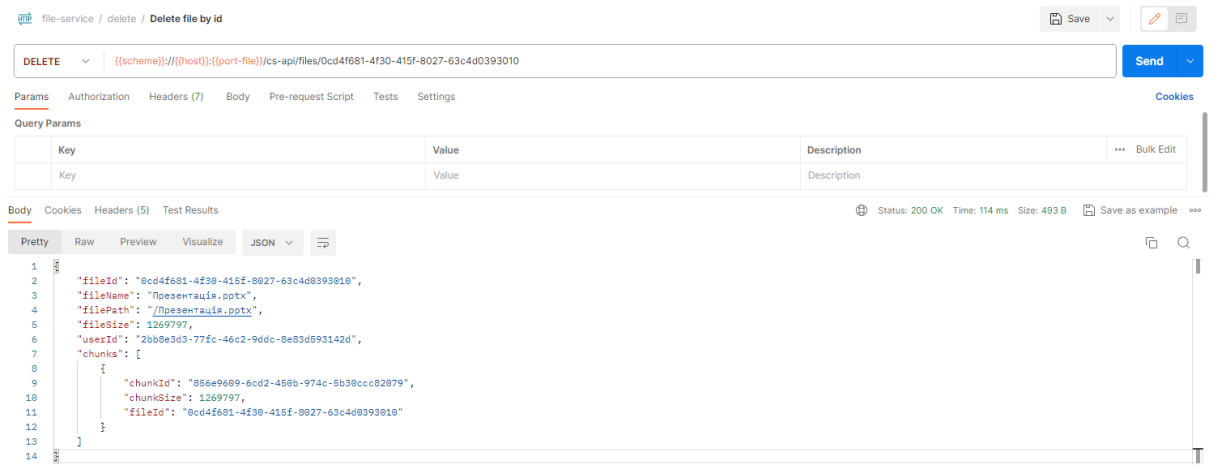

Рис. 3.22. Результат виконання запиту на видалення файлу

3.4.4. Запит на пошук метаданих по файлу

Для доступу до інформації доступні запити, які стосуються тільки мета сервісу, оскільки тільки він відповідає за забезпечення інформацією про метадані файлів та інших сутностей.

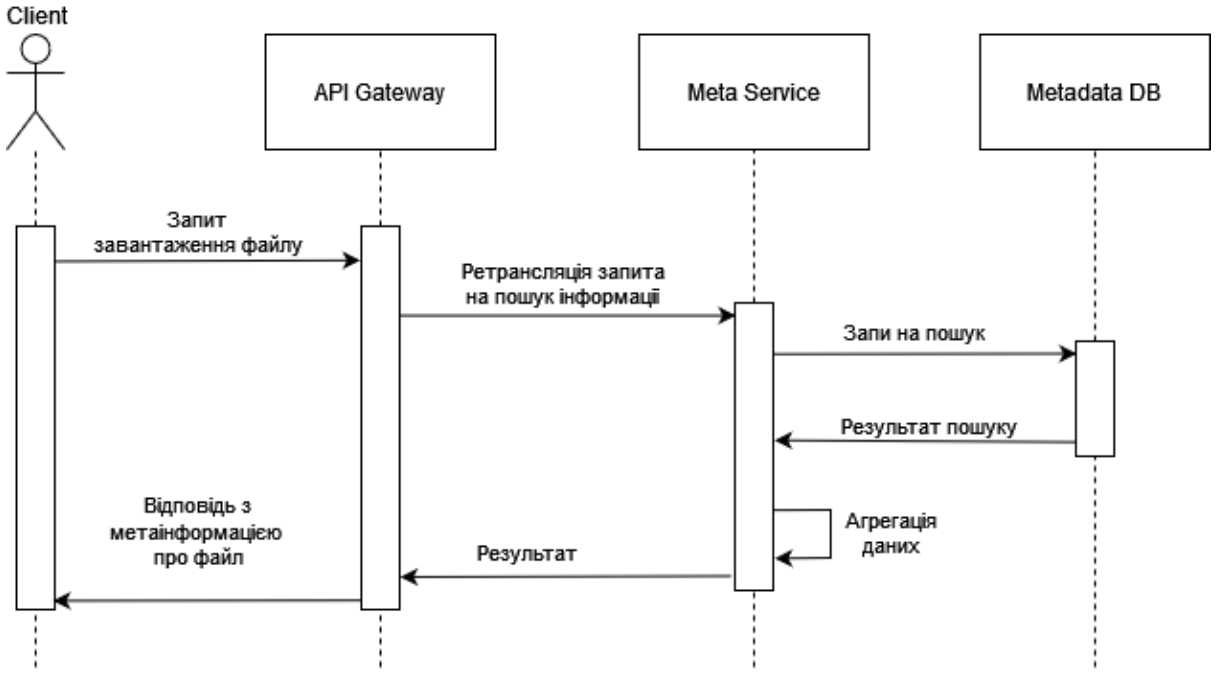

Рис. 3.21. Діаграма послідовностей на пошук метаданих

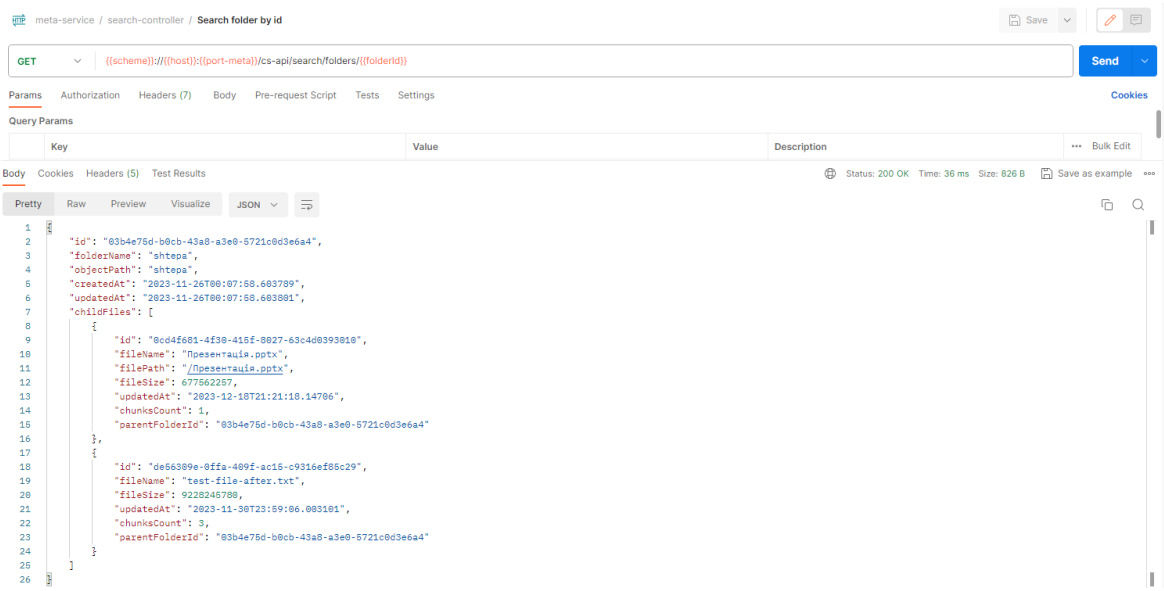

Рис. 3.22. Результат виконання запиту пошуку директорії

В даному прикладі запиту ми отримуємо інформацію про наявні сутності всередині корінній директорії. В результаті отримали, що в сховищі знаходиться два файли в корінній директорії користувача. Один з файлів менше ніж 4 MB і, відповідно, складається лише з одного фрагменту. Інший файл розміром 10 MB складається з трьох фрагментів.

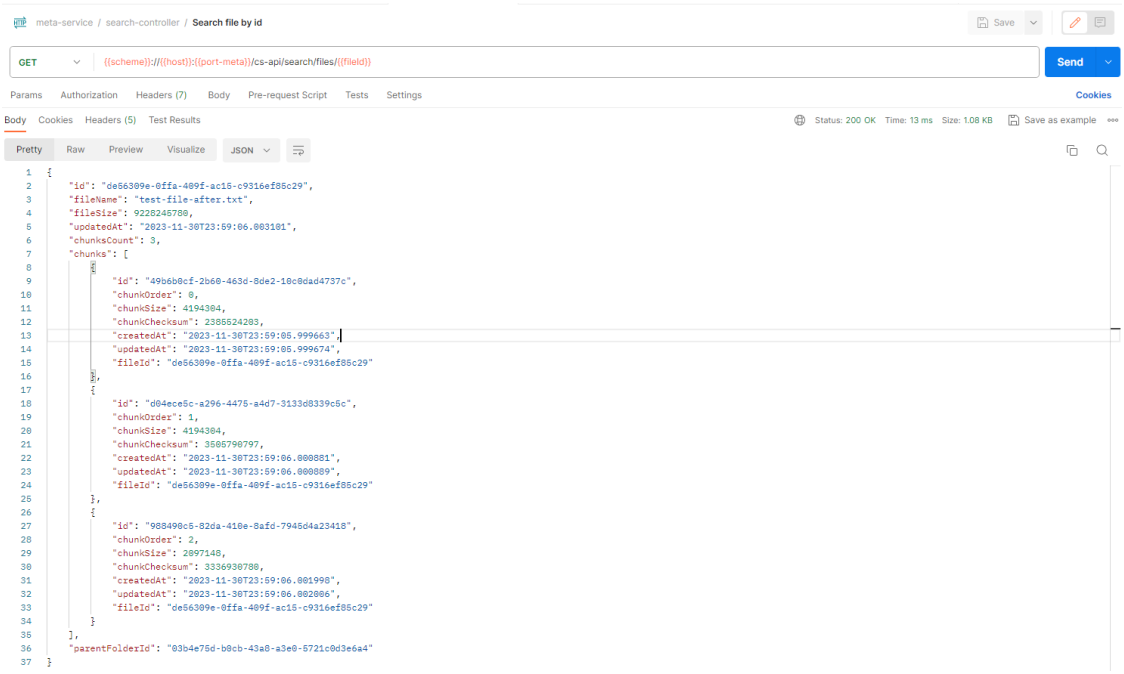

Рис. 3.23. Результат виконання запиту пошуку файлу

В запиті на пошук метаданих по файлу ми отримали разом з загальними даними про файл також і дані про фрагменти цього файлу. Файл розбитий на три впорядкованих фрагменти, два з яких розміром по 4 MB, а один має розмір 2 MB.

При наступних змінах у файлі, які скоріше за все будуть вкінці нього, для клієнта буде необхідність оновити у системі лише останній фрагмент з новими даними, який він завантажить через відповідний запит для завантажування фрагментів файлу.

3.5. Інтеграція з Amazon Web Services

Як і будь-який програмний продукт, реалізована система для повноцінної роботи з клієнтськими додатками та користувачам потребує наявності доступу через інтернет. Для цього її необхідно розгорнути у хмарі, що дасть можливість забезпечити доступ до неє усім, а також система отримає усі переваги використання хмарних сервісів.

Оскільки система вже використовує хмарні провайдера Amazon Web Service, раціонально використати інші сервіси, який надається компанією. Серед використаних сервісів будуть:

- AWS EC2 – сервіс, що забезпечує можливість створення віртуальних серверів;
- AWS VPC – сервіс для створення віртуальної приватної мережі;
- AWS API Gateway – сервіс для конфігурування API шлюзу;
- AWS S3 – об’єктне сховище;
- AWS RDS – сервіс для створення та конфігурування БД;
- AWS SQS – сервіс, який надає функціонал черги.

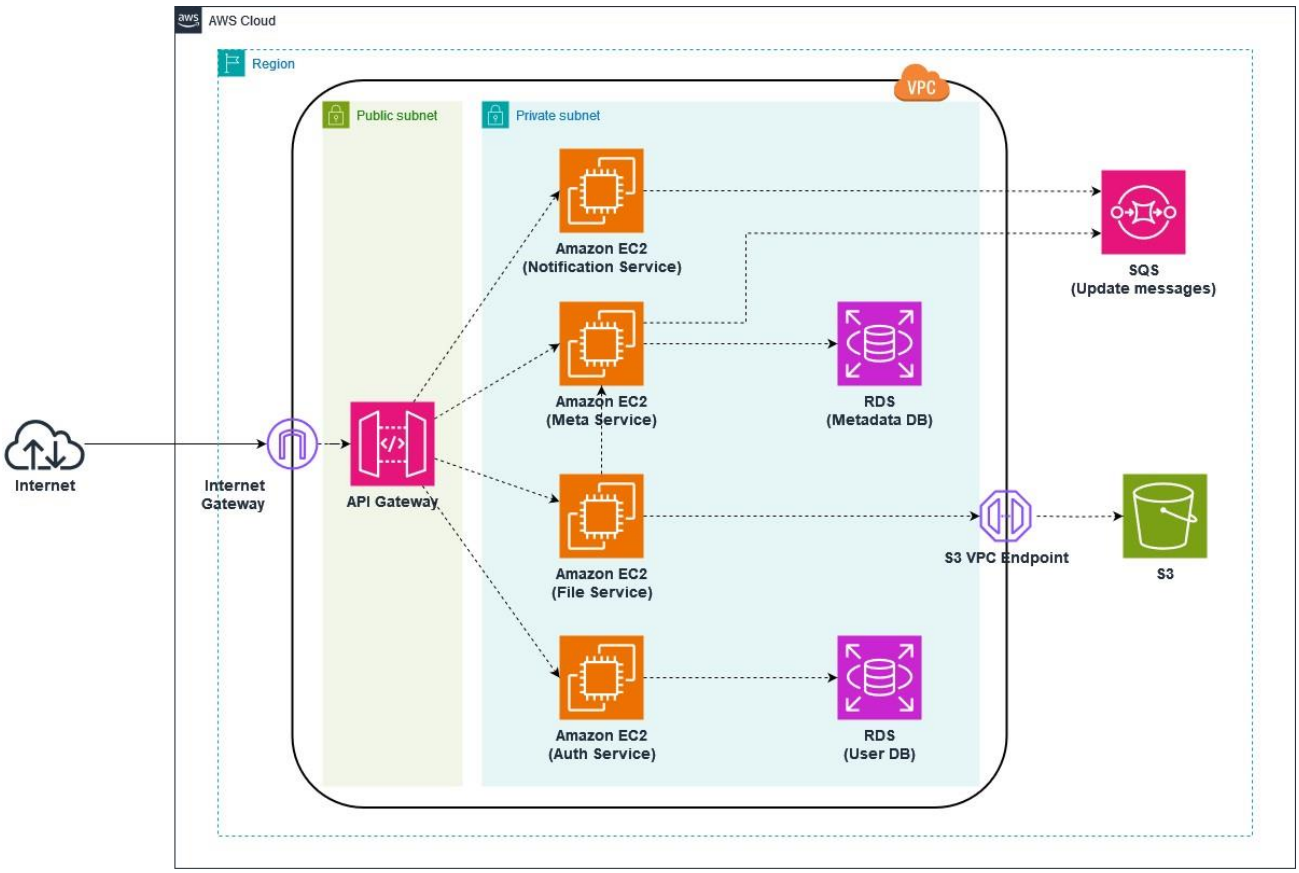

Рис.3. 24. Архітектура системи з використанням хмарних сервісів

Для розгортання мікросервісів буде використовуватися хмарний сервіс віртуальних серверів AWS EC2. Цей сервіс дає можливість запускати додатки у публічній хмарі та забезпечує відповідним функціоналом для конфігурування та контролю віртуальних серверів. Для роботи знадобиться по одному віртуальному серверу на кожний з чотирьох сервісів.

Мікросервіси, які запущені на віртуальних серверах, знаходяться в ізольованій мережі, що забезпечено AWS VPC сервісом, який дає змогу сконфігурувати віртуальну приватну мережу з обмеженням доступу окремих серверів до зовнішньої мережі.

AWS S3 сервіс використовується для збереження фрагментів файлів, які збережені в сховищі як об’єкти, доступ до яких можливий за ключем, що виступає в ролі ідентифікаторі.

Сервіс AWS RDS забезпечує можливість створення та конфігурації реляційної бази даних, яка використовується для збереження даних про файлові дані користувача,

а також і іншу інформацію.

Доступ до сервісів буде забезпечуватися через використання AWS API Gateway, що надасть єдину точку доступу до усіх наших сервісів, які знаходяться у приватній мережі.

Сервіс AWS SQS забезпечує можливість асинхронного обміну повідомленнями між сервісами. В нашому випадку, між сервісом контролю за метаінформацією та сервісом синхронізації станів локальних сховищ клієнтів.

3.6. Висновки до розділу 3

В результаті виконання третього розділу кваліфікаційної роботи було досліджено концепт системи, зроблено модель та реалізовано серверну частину системи сховища даних у хмарі з використанням мови Java та Spring Framework, а також сервісів провайдера хмарних обчислень Amazon Web Services.

Система надає функціонал по роботі та взаємодії з файловими даними користувача. Серед них можна виділити можливість роботи з одним або декількома файлами, функції по заміни та контролю його фрагментів, оптимізовані механізми переміщення та копіювання файлів.

При імплементації було використано різні механізми оптимізації в збереженні та маніпулюванні даними, які знаходяться в сховищі. Мікросервісна архітектура дозволила розділити процеси обробки на декілька окремих сервісів з подальшою можливістю горизонтального масштабування, тобто збільшення кількості окремих одиниць кожного сервісу. Збереження файлів відбувається через його фрагментування, що дає можливість подальшого завантаження не цілого зміненого файлу, а лише його фрагментів. Також слід відзначити додавання функціонала по роботі з директоріями, що дозволить зберігати повну копію локального сховища користувача у хмарі.

Створене програмне рішення дозволить використовувати його для збереження та підтримання консистентності даних користувача, а також забезпечити синхронізацію даних між пристроями для уніфікації стану локальних сховищ.

ВИСНОВКИ

В рамках виконання кваліфікаційної роботи було досліджено системи сховища даних, їх архітектурні та функціональні рішення. За результатами дослідження систем для збереження даних можна зробити висновок, що ці системи є комплексними та реалізують різні архітектурні та хмарні рішення. Кожна система для збереження даних відрізняється за способами збереження даних, їх форматом та організацією простору збереження. Системи виділяються наявністю додаткових сервісів для надання додаткових функцій, таких як можливість синхронізації файлових даних між декількома пристроями користувача, можливість проведення операцій лише над фрагментами файлів тощо.

Для реалізації досліджено різні інструменти для розробки проєкт, таких як мова програмування Java, Spring Framework та супутні модулі для реалізації серверної частини, а також використано хмарні сервіси AWS S3 для збереження фрагментів файлів та AWS SQS для обміну повідомленнями.

Створене програмне забезпечення, яке складається с кількох мікросервісів, дає змогу подальшого розвитку системи, в тому числі розвиток незалежно кожного її компонента. Серед переваг можна виділити:

- мікросервісна архітектура;
- можливість горизонтального та вертикального масштабування;
- функціонал для синхронізації даних;
- використання хмарних сервісів для роботи.

В перспективі система потребує реалізації клієнтського додатка, що дозволить використовувати функціонал, який надається реалізованою системою, автоматично при будь-яких змінах локального сховища.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. «Cloud Computing 101. Understanding the Basics and key Concepts». [Електронний ресурс] - Режим доступу: https://www.ness.com/cloud-computing-101-understanding-the-basics-and-key-concepts/ (дата звернення 23.11.23р) – Назва екрана.
2. «IBM Cloud Object Storage Concepts and Architecture System Edition». [Електронний ресурс] - Режим доступу: https://www.redbooks.ibm.com/redpapers/pdfs/redp5537.pdf (дата звернення 23.11.23р) – Назва екрана.
3. Chengzhang Peng, Zejun Jiang «Building a Cloud Storage Service System». [Електронний ресурс] - Режим доступу: https://www.sciencedirect.com/science/article/pii/S1878029611003069 (дата звернення 23.11.23р) – Назва екрана.
4. Cheng Yan «Cloud Storage Services». [Електронний ресурс] - Режим доступу: https://www.theseus.fi/bitstream/handle/10024/132555/Yan_Cheng.pdf?sequence=1&isAllowed=y (дата звернення 23.11.23р) – Назва екрана.
5. Arthur J.Deane. CCSP For Dummies with Online Practice 2nd Edition — Сан-Франциско: Вид-во For Dummies., 2020. — 62 с. : с. 83–95.
6. «What is Amazon EC2?». [Електронний ресурс] - Режим доступу: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (дата звернення 20.11.23р) – Назва екрана.
7. «What is Amazon S3?». [Електронний ресурс] - Режим доступу: https://docs.aws.amazon.com/AmazonS3/latest/userguide (дата звернення 20.11.23р) – Назва екрана.
8. «Managing data access with Amazon S3 access point». [Електронний ресурс] – Режим доступу: https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-points.html (дата звернення 20.11.23р) – Назва екрана.
9. «Amazon VPC features». [Електронний ресурс] - Режим доступу: https://aws.amazon.com/vpc/features/?nc1=h_ls (дата звернення 20.11.23р) – Назва екрана.

10. «What is Amazon Elastic Container Service?».». [Електронний ресурс] - Режим доступу: https://docs.aws.amazon.com/AmazonECS/latest/developerguide (дата звернення 20.11.23р) – Назва екрана.

11. «AWS SDK for Java 2.x: Developer Guide for version 2.x». [Електронний ресурс] - Режим доступу: https://docs.aws.amazon.com/pdfs/sdk-for-java/latest/developer-guide/aws-sdk-java-dg-v2.pdf (дата звернення 07.11.23р) – Назва екрана.

12. «Pros and Cons of Java Development in 2023». [Електронний ресурс] - Режим доступу: https://www.netguru.com/blog/java-pros-and-cons (дата звернення 07.11.23р) – Назва екрана.

13. «Spring Framework Overview». [Електронний ресурс] - Режим доступу: https://docs.spring.io/springframework/docs/5.3.x/reference/html/overview.html#overview (дата звернення 10.12.23р) – Назва екрана.

14. Роб Харроп, Кріс Шейфер. Pro Spring 6: An In-Depth Guide to the Spring Framework 6th Edition — Сан-Франциско: Вид-во Apress., 2023. — 62 с. : с. 83–95.

15. «Docker Overview». [Електронний ресурс] - Режим доступу: https://docs.docker.com/get-started/overview/ (дата звернення 11.12.23р) – Назва екрана.

ДОДАТОК А

```
@Configuration
public class AwsS3Config {
    private final String awsRegion;
    private final String awsS3FsEndpoint;

    @Autowired
    public AwsS3Config(@Value(value = "${aws.region}") String awsRegion,
                       @Value(value = "${aws.s3.fs.endpoint}") String awsS3FsEndpoint) {
        this.awsRegion = awsRegion;
        this.awsS3FsEndpoint = awsS3FsEndpoint;
    }

    @Bean
    public S3Client s3Client(AwsCredentialsProvider awsCredentialsProvider,
                             ClientOverrideConfiguration clientOverrideConfiguration) {
        return S3Client.builder()
                .region(Region.of(awsRegion))
                .endpointOverride(URI.create(awsS3FsEndpoint))
                .credentialsProvider(awsCredentialsProvider)
                .overrideConfiguration(clientOverrideConfiguration)
                .httpClient(ApacheHttpClient.create())
                .forcePathStyle(true)
                .build();
    }

    @Bean
    public S3AsyncClient s3AsyncClient(AwsCredentialsProvider awsCredentialsProvider,
```

```
ClientOverrideConfiguration clientOverrideConfiguration) {
    return S3AsyncClient.builder()
            .region(Region.of(awsRegion))
            .endpointOverride(URI.create(awsS3FsEndpoint))
            .credentialsProvider(awsCredentialsProvider)
            .overrideConfiguration(clientOverrideConfiguration)
            .forcePathStyle(true)
            .build();
  }

  @Bean
  @Primary
  protected ClientOverrideConfiguration clientOverrideConfiguration() {
    return ClientOverrideConfiguration.builder()
            .apiCallAttemptTimeout(Duration.ofSeconds(1))
            .retryPolicy(RetryPolicy.builder().numRetries(10).build())
            .build();
  }
  @Bean
  @Primary
  protected AwsCredentialsProvider awsCredentialsProvider() {
    return DefaultCredentialsProvider.create();
  }
```

ДОДАТОК Б

```
package edu.nau.cs.file.service.s3.impl;

import edu.nau.cs.file.service.dto.payload.S3FileChunkPayload;
import edu.nau.cs.file.service.exception.CsFileServiceS3FileIOException;
import edu.nau.cs.file.service.s3.AwsS3ErrorCode;
import edu.nau.cs.file.service.s3.AwsS3Service;
import edu.nau.cs.file.service.s3.S3Item;
import lombok.RequiredArgsConstructor;
import lombok.SneakyThrows;
import org.apache.tomcat.util.http.fileupload.IOUtils;
import org.springframework.stereotype.Service;
import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.ResponseInputStream;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.core.sync.ResponseTransformer;
import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.DeleteObjectResponse;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.NoSuchKeyException;
import software.amazon.awssdk.services.s3.model.PutObjectResponse;

import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.util.List;
import java.util.Objects;

@RequiredArgsConstructor
@Service
public class AwsS3ServiceImpl implements AwsS3Service {

    private final S3Client s3Client;

    @Override
    public byte[] getObject(String keyName, String bucketName) throws
CsFileServiceS3FileIOException {
        GetObjectRequest getObjectRequest = GetObjectRequest.builder()
                .key(keyName)
```

```
            .bucket(bucketName)
            .build();
        try (ResponseInputStream<GetObjectResponse> responseInputStream =
this.s3Client.getObject(getObjectRequest, ResponseTransformer.toInputStream());
             ByteArrayOutputStream outStream = new ByteArrayOutputStream()) {
            IOUtils.copy(responseInputStream, outStream);
            return outStream.toByteArray();
        } catch (Exception e) {
            if (e instanceof NoSuchKeyException &&
AwsS3ErrorCode.NO_SUCH_KEY.getStatus() == ((NoSuchKeyException)
e).statusCode()) {
                return null;
            }
            throw new CsFileServiceS3FileIOException(e.getMessage());
        }
    }

    @SneakyThrows
    @Override
    public S3Item uploadObject(S3FileChunkPayload fileUploadPayload, String bucket) {
        PutObjectResponse response = s3Client.putObject(request -> request
                    .key(fileUploadPayload.getS3Key())
                    .contentLength(fileUploadPayload.getChunkSize())
                    .bucket(bucket),
                RequestBody.fromBytes(fileUploadPayload.getBody().readAllBytes()));
        return new S3Item(fileUploadPayload.getS3Key(), response.eTag(),
response.versionId());
    }

    @Override
    public List<S3Item> uploadObjects(List<S3FileChunkPayload> fileUploadPayloads,
String bucket) {
        return fileUploadPayloads.stream()
                .map(fileUploadPayload -> this.uploadObject(fileUploadPayload, bucket))
                .toList();
    }

    @Override
    public S3Item deleteObject(String key, String bucket) {
        DeleteObjectResponse response = s3Client.deleteObject(request ->
request.key(key).bucket(bucket));
```

```
        return new S3Item(key, null, response.versionId());
    }

    @Override
    public List<S3Item> deleteObjects(List<String> keys, String bucket) {
        return keys.stream()
                .map(key -> this.deleteObject(key, bucket))
                .toList();
    }

}
```