ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ПРОГРАМНО-АПАРАТНИЙ КОМПЛЕКС СИНХРОНІЗАЦІЇ ДАНИХ У ГІБРИДНІЙ ЛОКАЛЬНО-ХМАРНІЙ ІНФРАСТРУКТУРІ»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми «Комп'ютерна інженерія»
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ *Дмитро ФЕДОРЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач вищої освіти гр. КІД-41
Дмитро ФЕДОРЕНКО
Ім'я, ПРІЗВИЩЕ

Керівник: Ярослав ТОРОШАНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Федоренко Дмитро Олександрович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Програмно-апаратний комплекс синхронізації даних у гібридній локально-хмарній інфраструктурі»
керівник кваліфікаційної роботи: Ярослав ТОРОШАНКО, кандидат технічних наук, доцент, старший науковий співробітник ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026р. №51
2. Строк подання кваліфікаційної роботи«14» травня 2026р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, документація сервісів Amazon Web Services та принципів роботи об'єктного сховища S3, механізми керування доступом (IAM, політики безпеки), інструменти синхронізації та резервного копіювання (rclone, AWS CLI, rsync), методи інкрементної реплікації даних та автоматизації (bash-скрипти, cron), підходи до забезпечення безпеки передачі даних і логування процесів, методи тестування, валідації цілісності та оцінювання ефективності гібридних систем зберігання.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Теоретичні засади управління даними у гібридних хмарних системах
    2. Практична розробка комплексу синхронізації на базі Ubuntu Server та AWS
    3. Оптимізація та економічне обґрунтування гібридного рішення
5. Перелік ілюстративного матеріалу: *презентація*
    1. Мета, об'єкт, предмет дослідження
    2. Актуальність роботи
    3. Загальна архітектура рішення
    4. Апаратно-програмне середовище
    5. Конфігурація системи
    6. Алгоритм роботи системи
    7. Тестування системи
    8. Методи оптимізації

9. Порівняння з SaaS
10. Економічна ефективність
11. Висновки
12. Публікації та апробація роботи

6. Дата видачі завдання «20» лютого 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	22.02-20.03.2026	Виконано
2	Обробка матеріалу	20.03-30.03.2026	Виконано
3	Розробка теоретичної частини	31.03-10.04.2026	Виконано
4	Практична розробка комплексу синхронізації на базі Ubuntu Server та AWS	11.04-20.04.2026	Виконано
5	Висновки за результатами аналізу	21.04-26.04.2026	Виконано
6	Оформлення пояснювальної записки	26.04-28.04.2026	Виконано
7	Розробка презентації	29.04-01.05.2026	Виконано
8	Передзахист	08.05-11.05.2026	Виконано
9	Здача в деканат	08.05-14.05.2026	Виконано

Здобувач вищої освіти _____ Дмитро ФЕДОРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Ярослав ТОРОШАНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 77 стор., 37 рис., 6 табл., 31 джерел.

Мета роботи – оптимізація процесу синхронізації та резервного копіювання даних у гібридній локально-хмарній інфраструктурі за рахунок використання інкрементних методів передачі, автоматизації процесів та об'єктних моделей зберігання.

Об'єкт дослідження – процеси синхронізації та реплікації даних у гібридних локально-хмарних середовищах.

Предмет дослідження – засоби та технології побудови, налаштування й оптимізації систем синхронізації даних на базі локальних серверів і хмарних сервісів об'єктного зберігання.

Короткий зміст роботи: У роботі розглянуто шляхи до побудови гібридних систем зберігання даних, архітектуру локальних та хмарних сховищ, принципи об'єктного зберігання (Amazon S3) та механізми забезпечення безпеки. Проаналізовано функціональні обмеження інструментів синхронізації (rclone, AWS CLI), зокрема щодо залежності від провайдера та ефективності оптимізації.

Практична частина включає розробку програмно-апаратного комплексу на базі Ubuntu Server, інтегрованого з хмарним середовищем AWS (S3) через rclone. Реалізовано алгоритми локального дублювання (rsync), асинхронної передачі даних та автоматизації за допомогою bash-скриптів. Проведене тестування підтвердило цілісність даних та працездатність системи, а аналіз ефективності та економічне обґрунтування довели доцільність впровадження запропонованої гібридної моделі.

КЛЮЧОВІ СЛОВА: ГІБРИДНА ІНФРАСТРУКТУРА, СИНХРОНІЗАЦІЯ ДАНИХ, РЕПЛІКАЦІЯ, AMAZON S3, RCLONE, AWS CLI, RSYNC, ІНКРЕМЕНТНА ПЕРЕДАЧА, РЕЗЕРВНЕ КОПІЮВАННЯ, АВТОМАТИЗАЦІЯ, ХМАРНІ СЕРВІСИ, ОПТИМІЗАЦІЯ, ЛОГІНГ.

ЗМІСТ

ВСТУП 8
1. ТЕОРЕТИЧНІ ЗАСАДИ УПРАВЛІННЯ ДАНИМИ У ГІБРИДНИХ ХМАРНИХ СИСТЕМАХ 9
1.1 Концепція гібридної інфраструктури зберігання даних 9
1.2 Технології об'єктного зберігання даних 16
1.3 Інструментарій синхронізації та резервного копіювання 21
1.4 Безпека передачі даних у гетерогенних мережах 27
1.5 Нормативні вимоги до зберігання інформації 31
2. ПРАКТИЧНА РОЗРОБКА КОМПЛЕКСУ СИНХРОНІЗАЦІЇ НА БАЗІ UBUNTU SERVER ТА AWS. 37
2.1 Архітектура програмно-апаратного комплексу 37
2.2 Етапи розгортання та конфігурації системи 45
2.3 Алгоритми роботи системи реплікації 53
2.4 Тестування працездатності комплексу 59
3. ОПТИМІЗАЦІЯ ТА ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ГІБРИДНОГО РІШЕННЯ 67
3.1 Методи оптимізації процесу синхронізації 67
3.2 Порівняльний аналіз із комерційними аналогами 73
3.3 Оцінка вартості та перспективи розвитку 78
ВИСНОВКИ 84
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ 85
Додаток А. Лістинг конфігураційного файлу політики доступу AWS IAM (S3SyncPolicy.json) 87
Додаток Б. Вихідний код bash-скрипта автоматизованої двоетапної синхронізації даних (sync_script.sh) 88
Додаток В. Конфігурація системи ротації та архівування логів (logrotate) 89
Додаток Г. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація) 90

ВСТУП

Інформаційні системи потребують ефективного зберігання й обробки даних через їх стрімке зростання. Традиційні локальні рішення не забезпечують достатньої масштабованості, а хмарні моделі обмежують контроль над ресурсами. Тому актуальною стає гібридна інфраструктура, що поєднує переваги обох середовищ, забезпечуючи баланс між продуктивністю, надійністю та гнучкістю управління.

Гібридні системи розподіляють дані між рівнями: локальні сховища забезпечують оперативний доступ, а хмарні сервіси — масштабоване резервування та довготривале зберігання. Важливим аспектом є організація об’єктного зберігання (наприклад, Amazon S3) з механізмами контролю доступу та безпеки, що відрізняє такі рішення від традиційних файлових систем і дозволяє будувати відмовостійкі розподілені архітектури.

Для ефективної взаємодії компонентів потрібні спеціалізовані інструменти автоматизації, зокрема CLI-утиліти для інкрементальної передачі даних та API-інтеграції. Особлива увага приділяється безпеці: захищеному транспортуванню, шифруванню та управлінню обліковими даними. Також важливо дотримуватися нормативних вимог щодо життєвого циклу даних, резервного копіювання та ведення журналів операцій.

Практична частина передбачає розробку комплексу на базі Ubuntu Server та AWS. Реалізація охоплює налаштування взаємодії з об’єктним сховищем, конфігурацію AWS CLI та rclone, а також створення алгоритмів реплікації (локальне дублювання та асинхронна передача у хмару). Тестування системи через моделювання змін у файловій структурі дозволить перевірити цілісність даних та оцінити ефективність рішення в умовах, наближених до реальної експлуатації.

1. ТЕОРЕТИЧНІ ЗАСАДИ УПРАВЛІННЯ ДАНИМИ У ГІБРИДНИХ ХМАРНИХ СИСТЕМАХ

1.1 Концепція гібридної інфраструктури зберігання даних

Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів даних, що формуються у процесі діяльності підприємств, функціонування інформаційних систем та взаємодії користувачів із цифровими сервісами, при цьому зростання обсягів інформації супроводжується необхідністю забезпечення її ефективного, безпечного та масштабованого зберігання. Традиційні підходи, що базуються виключно на використанні локальних серверів, дедалі частіше не відповідають сучасним вимогам, оскільки мають обмежену масштабованість, потребують значних витрат на обслуговування та ускладнюють забезпечення відмовостійкості, що в умовах безперервної роботи інформаційних систем стає критичним фактором.

Водночас повністю хмарні рішення, незважаючи на їх гнучкість та масштабованість, також мають низку обмежень, серед яких залежність від:

1. стабільності мережевого з’єднання;
2. потенційні ризики, пов’язані з безпекою даних;
3. обмежений рівень контролю над інфраструктурою.

За таких умов доцільним стає використання комбінованого підходу, який поєднує переваги локальних і хмарних технологій у межах єдиної системи, що реалізується через концепцію гібридної інфраструктури зберігання даних [1].

Гібридна інфраструктура передбачає інтеграцію локальних обчислювальних ресурсів із хмарними сервісами у вигляді узгодженого середовища, в якому частина даних або сервісів розміщується на локальних серверах (on-premise), тоді як інша частина — у хмарному середовищі, наприклад у бакетах Amazon S3, при цьому взаємодія між цими компонентами забезпечується за допомогою механізмів

синхронізації, реплікації та резервного копіювання. Такий підхід дозволяє забезпечити швидкий доступ до критично важливих даних на локальному рівні, одночасно використовуючи масштабованість, гнучкість та надійність хмарної інфраструктури, що формує ефективне рішення для сучасних інформаційних систем [1].

У контексті розробки програмно-апаратного комплексу синхронізації даних гібридна інфраструктура виступає базовою моделлю, яка визначає принципи взаємодії між локальним сервером та хмарним сховищем, забезпечуючи узгодженість, доступність і цілісність інформації.

Архітектурні особливості локальних та хмарних сховищ. Локальні системи зберігання даних зазвичай реалізуються у вигляді:

— файлових серверів;

— мережевих сховищ (NAS);

— систем із прямим підключенням (DAS).

При цьому їх основною характеристикою є забезпечення мінімальних затримок доступу до інформації та повний контроль над апаратною інфраструктурою. Дані у таких системах організовуються у вигляді ієрархічної файлової структури, яка є типовою для більшості операційних систем, що спрощує роботу з ними та інтеграцію у вже існуючі програмні середовища (див. рис. 1.1) [2].

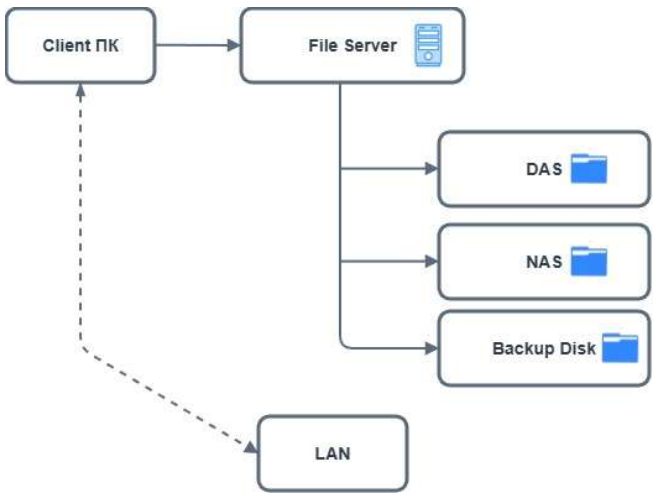


Рисунок 1.1 – Архітектура локального файлового сховища

Разом із цим локальні сховища мають ряд обмежень, які стають особливо помітними при збільшенні обсягів даних, оскільки масштабування таких систем потребує фізичного оновлення обладнання, розширення дискового простору та обслуговування апаратних компонентів, що супроводжується значними фінансовими витратами. Додатково забезпечення резервування та відмовостійкості вимагає впровадження складних конфігурацій, використання додаткових носіїв та організації дублювання даних, що ускладнює адміністрування системи [2].

На відміну від локальних рішень, хмарні сховища, зокрема об’єктні системи зберігання, такі як Amazon S3, використовують інший підхід до організації даних, у межах якого інформація зберігається у вигляді об’єктів, кожен з яких має унікальний ідентифікатор та набір метаданих. Така модель дозволяє реалізувати високу масштабованість, оскільки дані автоматично розподіляються між великою кількістю фізичних серверів, що забезпечує не лише ефективне використання ресурсів, але й підвищену відмовостійкість (див. рис. 1.2).

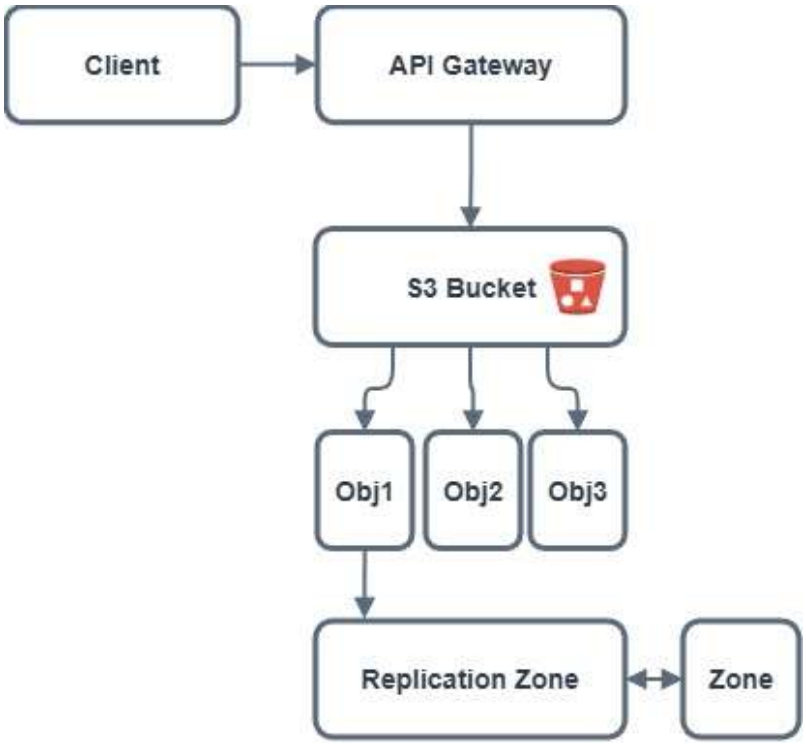


Рисунок 1.2 – Архітектура хмарного об’єктного сховища

Хмарні сервіси додатково надають вбудовані механізми резервування, географічної реплікації та балансування навантаження, що дозволяє досягати високого рівня доступності даних без необхідності безпосереднього управління інфраструктурою. Разом із цим доступ до хмарних ресурсів залежить від мережевого з'єднання, а обробка великих обсягів інформації може супроводжуватися затримками, що необхідно враховувати при проєктуванні систем [2].

Архітектурні відмінності між локальними та хмарними сховищами полягають не лише у фізичному розміщенні даних, але й у принципах їх організації, доступу та масштабування, що обумовлює необхідність використання спеціалізованих інструментів для синхронізації та інтеграції цих середовищ у межах гібридної інфраструктури.

Переваги та недоліки гібридних підходів. Гібридна інфраструктура поєднує сильні сторони локальних і хмарних рішень, що робить її універсальним підходом для широкого спектра прикладних задач, пов'язаних зі зберіганням та обробкою даних, при цьому одним із головних аспектів такої моделі є можливість гнучкого розподілу інформації залежно від її призначення, важливості та частоти використання, оскільки дані, що не потребують постійного доступу або мають архівний характер, можуть без обмежень зберігатися у хмарному середовищі, тоді як критично важлива інформація розміщується локально для забезпечення швидкого доступу [3].

Підвищення надійності системи досягається за рахунок дублювання даних у різних середовищах, що дозволяє відновити інформацію у разі виходу з ладу локального обладнання шляхом отримання доступу до хмарного сховища, завдяки чому значно знижуються ризики втрати даних і забезпечується безперервність роботи інформаційної системи. Такий підхід природно узгоджується з сучасними стратегіями резервного копіювання, зокрема моделлю 3-2-1, яка передбачає наявність кількох копій даних на різних носіях із розміщенням щонайменше однієї копії поза межами основної інфраструктури, що у гібридному середовищі реалізується за рахунок поєднання локальних ресурсів і хмарних сервісів [3].

Оптимізація витрат також є важливою перевагою, оскільки відсутня необхідність інвестування у надлишкові локальні ресурси великого обсягу, які можуть використовуватися не повністю, натомість частина даних може зберігатися у хмарі за моделлю оплати відповідно до фактичного використання, що дозволяє ефективно розподіляти фінансові ресурси та уникати зайвих витрат на придбання і обслуговування обладнання.

Разом із цим гібридний підхід характеризується підвищеною складністю реалізації та адміністрування, оскільки необхідно забезпечити коректну взаємодію між локальними та хмарними компонентами, включаючи:

— налаштування мережевих з’єднань;

— механізмів автентифікації;

— контролю доступу;

— процесів синхронізації даних.

Все це потребує додаткових зусиль і технічної підготовки. Забезпечення узгодженості даних стає окремим викликом, оскільки використання кількох джерел зберігання вимагає контролю версій, обробки можливих конфліктів та підтримки актуальності інформації, що ускладнює загальну логіку функціонування системи [3].

Залежність від мережевого з’єднання залишається ще одним важливим фактором, оскільки нестабільність каналу передачі даних може призводити до затримок або переривання процесів синхронізації, у результаті чого локальні дані можуть тимчасово відрізнятися від тих, що зберігаються у хмарному середовищі, що особливо критично для систем, які потребують роботи в режимі, близькому до реального часу. У таких умовах виникає необхідність використання додаткових механізмів контролю передачі, буферизації та повторної синхронізації (див. табл. 1.1) [3].

Таблиця 1.1 – Порівняння підходів до зберігання даних

Критерій	Локальне зберігання	Хмарне зберігання	Гібридне зберігання
Масштабованість	Обмежена (залежить від обладнання)	Висока (динамічне розширення)	Висока (комбінована)
Доступність	Висока в локальній мережі	Залежить від Інтернету	Висока (локально та віддалено)
Контроль	Повний контроль	Обмежений (через провайдера)	Частковий (баланс контролю)
Вартість	Висока при масштабуванні	Оплата за використання	Оптимізована
Продуктивність	Висока (низькі затримки)	Середня (мережеві затримки)	Висока (локально та кешування)
Відмовостійкість	Потребує додаткових рішень	Вбудована	Висока (реплікація даних)

Гібридна інфраструктура у підсумку формує потужне та гнучке рішення для організації зберігання даних, проте її ефективність безпосередньо залежить від якості проєктування, правильного вибору інструментів та налаштування механізмів управління даними, що дозволяє досягти балансу між продуктивністю, надійністю та витратами.

Вимоги до надійності корпоративних даних. У сучасних інформаційних системах дані розглядаються як один із найцінніших ресурсів, оскільки їх неможливо повністю відтворити у первісному вигляді після втрати, адже кожен набір інформації є унікальним за своєю структурою, змістом та контекстом використання, у зв’язку з чим забезпечення їх надійності стає критично важливим завданням.

Надійність даних охоплює декілька взаємопов’язаних аспектів, серед яких:

— доступність;

— цілісність;

— конфіденційність;

— відмовостійкість.

При цьому доступність передбачає можливість отримання інформації у будь-який необхідний момент часу, що у гібридних системах досягається за рахунок дублювання даних у різних середовищах, зокрема на локальних серверах і у хмарних сховищах, що дозволяє мінімізувати час простою у разі відмови одного з компонентів [4].

Цілісність даних означає їх незмінність та коректність під час зберігання і передачі, що забезпечується використанням механізмів перевірки, таких як контрольні суми, журналювання змін та версіювання, завдяки чому стає можливим виявлення помилок або спотворень інформації, особливо у процесі синхронізації, де кожен файл повинен бути переданий без втрат або несанкціонованих змін.

Конфіденційність інформації досягається шляхом застосування сучасних методів шифрування та обмеження доступу до даних лише для авторизованих користувачів, що у хмарних середовищах реалізується через політики доступу та механізми управління правами, які дозволяють чітко визначити рівень доступу для кожного користувача або сервісу.

Відмовостійкість системи забезпечується за рахунок використання резервного копіювання та надлишкових компонентів, що дозволяє зберегти працездатність навіть у разі виникнення збоїв, при цьому одним із найбільш поширених підходів є стратегія 3-2-1, яка передбачає зберігання трьох копій даних на двох різних типах носіїв із розміщенням однієї копії поза межами основної інфраструктури, що у гібридному середовищі реалізується шляхом поєднання локального зберігання та хмарних сервісів (див. рис. 1.3) [4].

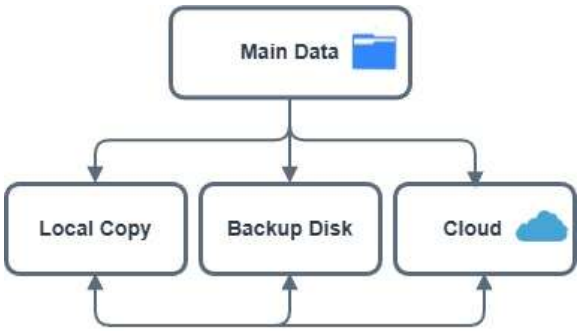


Рисунок 1.3 – Модель забезпечення відмовостійкості даних

Додатковим аспектом забезпечення надійності є аудит і моніторинг операцій із даними, що дозволяє відстежувати всі зміни, виявляти потенційні загрози та своєчасно реагувати на інциденти, забезпечуючи контроль над процесами обробки інформації. У межах програмно-апаратного комплексу синхронізації такі вимоги реалізуються через поєднання локального резервування, використання інструментів автоматизації та хмарної синхронізації, що дозволяє досягти високого рівня надійності даних при відносно оптимальних витратах ресурсів.

1.2 Технології об'єктного зберігання даних

Зростання обсягів цифрової інформації, що генерується у процесі функціонування інформаційних систем, супроводжується необхідністю використання масштабованих і гнучких рішень для її зберігання, у зв’язку з чим традиційні файлові системи поступово доповнюються або комбінуються з новими підходами, серед яких особливе місце займає об’єктне зберігання даних, яке передбачає представлення інформації у вигляді окремих об’єктів, кожен з яких містить не лише самі дані, але й метадані та унікальний ідентифікатор. Така структура дозволяє ефективно працювати з великими обсягами інформації, забезпечуючи високу масштабованість, гнучкість та надійність системи зберігання, оскільки відсутність жорсткої ієрархічної структури значно спрощує розподіл даних між фізичними вузлами.

На відміну від традиційних файлових систем, у яких інформація організована у вигляді каталогів і підкаталогів, об’єктні сховища використовують плоску структуру, в межах якої всі об’єкти розміщуються у логічних контейнерах, що в середовищі хмарних сервісів називаються бакетами, при цьому доступ до об’єктів здійснюється через унікальні ключі, що забезпечує швидкий пошук і взаємодію з даними незалежно від їх фізичного розташування. Такий підхід набув широкого застосування у хмарних середовищах, де необхідно обробляти великі обсяги

неструктурованих даних, включаючи журнали подій, резервні копії, мультимедійні файли та інші типи інформації, що не мають чітко визначеної структури.

Популярність об’єктного зберігання значною мірою обумовлена можливістю горизонтального масштабування, коли система автоматично розподіляє дані між великою кількістю серверів, що дозволяє збільшувати обсяг сховища без суттєвих змін у логіці роботи додатків. У цьому контексті одним із найбільш поширених рішень є сервіс Amazon S3, який став де-факто стандартом у сфері хмарного зберігання даних і широко використовується для реалізації як корпоративних, так і індивідуальних інформаційних систем.

Принципи роботи Amazon S3. Amazon S3 (Simple Storage Service) представляє собою об’єктне сховище, яке забезпечує можливість зберігання та отримання довільного обсягу даних у будь-який момент часу через мережу Інтернет, при цьому основною одиницею зберігання є об’єкт, що складається з трьох компонентів:

— власне даних;

— метаданих;

— унікального ключа.

Дані зберігаються у вигляді бінарного вмісту, тоді як метадані містять службову інформацію, таку як тип файлу, дата створення, параметри доступу або додаткові користувацькі атрибути, що дозволяє ефективно управляти об’єктами та організовувати їх обробку (див. рис. 1.4) [5].

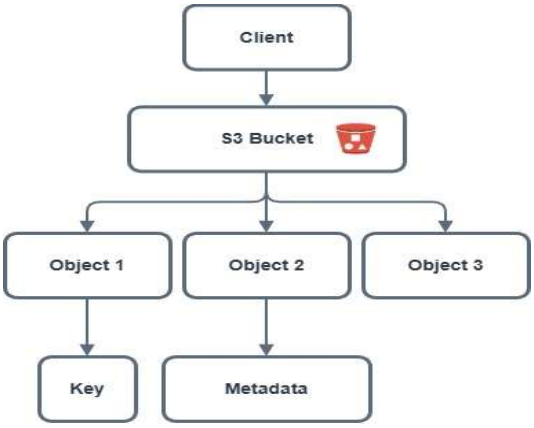


Рисунок 1.4 – Принцип роботи Amazon S3 (bucket-object модель)

Унікальний ключ виступає ідентифікатором об’єкта в межах бакета та використовується для доступу до нього, що дозволяє виконувати операції над даними незалежно від їх фізичного розташування у хмарній інфраструктурі. Організація даних у S3 здійснюється через бакети, які є логічними контейнерами для об’єктів, при цьому кожен бакет має унікальне ім’я в межах всієї платформи та прив’язаний до певного географічного регіону, що дозволяє оптимізувати доступ до даних і зменшити затримки при їх передачі [5].

Доступ до об’єктів у межах Amazon S3 реалізується через REST API або за допомогою спеціалізованих інструментів, таких як AWS CLI чи сторонні утиліти, наприклад rclone, що дозволяє інтегрувати хмарне сховище з локальними системами. Основні операції включають завантаження даних, їх отримання, видалення та синхронізацію, що забезпечує повний цикл роботи з інформацією.

Однією з ключових характеристик S3 є автоматичне масштабування, яке дозволяє системі розподіляти дані між великою кількістю фізичних серверів без участі користувача, забезпечуючи високу доступність і відмовостійкість. Додатково підтримується механізм версіювання, що дозволяє зберігати попередні версії об’єктів і відновлювати їх у разі необхідності, що є важливим для систем резервного копіювання та синхронізації даних [5].

Сучасні реалізації сервісу забезпечують сильну узгодженість, яка означає, що після запису об’єкта він одразу стає доступним для читання, що спрощує розробку систем, які працюють із даними у реальному часі або близькому до нього режимі. У сукупності ці властивості формують високонадійне, масштабоване та гнучке середовище зберігання, придатне для реалізації гібридних сценаріїв синхронізації даних між локальними та хмарними компонентами.

Механізми контролю доступу в хмарних бакетах. Забезпечення безпеки даних у хмарних середовищах передбачає використання комплексних підходів до контролю доступу, що дозволяють гнучко керувати правами користувачів і сервісів, у зв’язку з чим у середовищі Amazon S3 реалізовано багаторівневу систему управління доступом, яка охоплює механізми ідентифікації, авторизації та

обмеження прав на виконання операцій з об’єктами. Основою такої системи виступає сервіс керування ідентифікацією та доступом, що дозволяє створювати:

— користувачів;

— ролі;

— політики.

Всі вони визначають дозволені дії щодо конкретних ресурсів, при цьому доступ може бути обмежений як на рівні окремих бакетів, так і на рівні окремих об’єктів [6].

Політики доступу описуються у формалізованому вигляді та містять правила, що визначають дозволи або заборони на виконання певних операцій над даними, таких як:

— читання;

— запис;

— видалення.

Все це дозволяє реалізувати принцип мінімальних привілеїв, відповідно до якого кожному користувачу або сервісу надаються лише ті права, які необхідні для виконання конкретних завдань. Додатковим рівнем контролю виступають політики бакетів, які застосовуються безпосередньо до контейнерів зберігання та визначають умови доступу для всіх суб’єктів взаємодії, включаючи можливість обмеження доступу за IP-адресами, протоколами або іншими параметрами середовища (див. рис. 1.5).

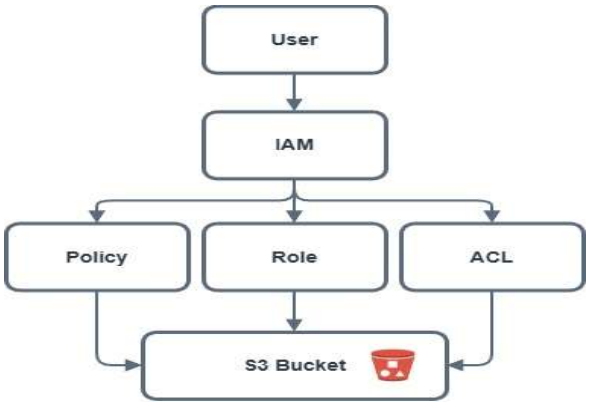


Рисунок 1.5 – Механізми контролю доступу в S3 (IAM, ACL, Policy)

У межах системи контролю доступу також використовуються списки контролю доступу, які дозволяють задавати права на рівні окремих об’єктів, проте сучасні практики передбачають поступову відмову від їх використання на користь більш гнучких і централізованих механізмів, заснованих на політиках доступу, що забезпечують кращу керованість і масштабованість. Додатковим заходом безпеки є блокування публічного доступу, яке дозволяє запобігти випадковому відкриттю даних у мережі Інтернет, що є особливо важливим для корпоративних систем, де витік інформації може призвести до серйозних наслідків [6].

Захист даних доповнюється використанням механізмів шифрування як на стороні сервера, так і на стороні клієнта, що забезпечує додатковий рівень безпеки під час зберігання та передачі інформації, у результаті чого багаторівнева модель контролю доступу дозволяє адаптувати параметри безпеки відповідно до вимог конкретної організації та забезпечити належний рівень захисту даних у хмарному середовищі.

Порівняння S3 з традиційними файловими системами. Порівняння об’єктного сховища Amazon S3 з традиційними файловими системами дозволяє більш чітко визначити їхні особливості, переваги та обмеження у контексті використання в гібридних інфраструктурах, оскільки класичні файлові системи, такі як ті, що використовуються в операційних системах, організовують дані у вигляді ієрархічної структури каталогів і файлів, забезпечуючи швидкий доступ до інформації, підтримку редагування та тісну інтеграцію з програмним середовищем [7].

Такий підхід є ефективним для локального використання та роботи з відносно невеликими обсягами даних, проте при масштабуванні виникають обмеження, пов’язані з апаратними ресурсами та складністю адміністрування. На відміну від цього, Amazon S3 використовує об’єктну модель, яка не передбачає класичної файлової ієрархії, хоча логічна структура може бути імітована за допомогою використання префіксів у ключах об’єктів, що створює враження каталогів, тоді як фактично всі дані зберігаються у плоскій структурі.

Однією з головних переваг S3 є можливість зберігання практично необмеженого обсягу даних без необхідності модернізації локальної інфраструктури, оскільки масштабування здійснюється на рівні хмарної платформи, що значно спрощує управління ресурсами. Доступ до даних у традиційних файлових системах здійснюється безпосередньо через операційну систему, тоді як у випадку S3 взаємодія відбувається через мережеві запити, що забезпечує гнучкість, але водночас може призводити до затримок, пов’язаних із передачею даних через мережу [7].

У контексті синхронізації даних використання об’єктного сховища вимагає застосування спеціалізованих інструментів, які враховують особливості об’єктної моделі, при цьому такі утиліти дозволяють виконувати порівняння локальних файлів і віддалених об’єктів та передавати лише змінені дані, що підвищує ефективність обміну інформацією. Рівень надійності у S3 забезпечується за рахунок автоматичного дублювання даних і розподілу їх між різними фізичними вузлами, тоді як у локальних системах резервування потребує окремого налаштування і додаткових ресурсів [6-7].

Разом із цим локальні файлові системи зберігають перевагу у швидкості доступу, особливо при роботі з великими обсягами даних у межах однієї мережі, тоді як використання хмарних сервісів завжди пов’язане з впливом характеристик мережевого з’єднання, що визначає затримки та пропускну здатність. У результаті обидва підходи мають свої сильні та слабкі сторони, що обумовлює доцільність їх комбінування у межах гібридної інфраструктури для досягнення оптимального балансу між продуктивністю, масштабованістю та надійністю.

1.3 Інструментарій синхронізації та резервного копіювання

Ефективна організація зберігання даних у гібридних інфраструктурах передбачає використання спеціалізованого інструментарію, який забезпечує

синхронізацію та резервне копіювання інформації між локальними і хмарними компонентами, у зв’язку з чим застосування відповідних програмних засобів стає необхідною умовою для підтримання цілісності даних, їх актуальності та доступності. Синхронізація дозволяє підтримувати узгоджений стан даних у різних середовищах, тоді як резервне копіювання спрямоване на збереження їхніх копій для подальшого відновлення у разі виникнення збоїв або втрати інформації, що у сукупності формує основу надійної системи управління даними.

Основними вимогами до інструментів такого типу є підтримка інкрементальної передачі даних, що дозволяє передавати лише змінені частини інформації, можливість роботи з різними типами сховищ, включаючи локальні файлові системи та хмарні сервіси, стійкість до нестабільного мережевого з’єднання, а також гнучкість у налаштуванні параметрів синхронізації. У контексті гібридних систем особливого значення набуває здатність інтеграції з об’єктними сховищами, зокрема такими як Amazon S3, оскільки саме вони забезпечують масштабовану та відмовостійку частину інфраструктури, що використовується для довготривалого зберігання даних.

У межах реалізації програмно-апаратного комплексу синхронізації доцільно розглядати кілька підходів до організації передачі даних, серед яких виділяється:

1. використання універсальних інструментів синхронізації;
2. застосування офіційних утиліт для роботи з хмарними сервісами;
3. аналіз альтернативних рішень.

Все це дозволяє реалізувати реплікацію даних залежно від вимог до продуктивності, надійності та складності системи.є

Аналіз можливостей утиліти rclone. Серед доступних інструментів синхронізації особливу увагу привертає утиліта rclone, яка представляє собою універсальне рішення для роботи з даними між локальними файловими системами та хмарними сховищами, забезпечуючи підтримку великої кількості провайдерів, включаючи Amazon S3, Google Drive, Microsoft Azure та інші сервіси, що робить її зручним інструментом для побудови гібридних систем зберігання даних. Універсальність такого підходу полягає у можливості використання єдиного

інтерфейсу для взаємодії з різними типами сховищ, що значно спрощує процес інтеграції та адміністрування.

Налаштування rclone здійснюється через конфігураційний файл, у якому визначаються параметри підключення до віддалених ресурсів, включаючи:

— облікові дані;

— тип сховища;

— інші параметри доступу.

Все це дозволяє адаптувати інструмент під конкретні вимоги середовища. Однією з основних функцій є команда синхронізації, яка забезпечує односторонню передачу даних між джерелом і цільовим сховищем, при цьому виконуючи порівняння файлів за розміром, датою зміни та контрольними сумами, що дозволяє передавати лише змінені дані та значно зменшувати обсяг мережевого трафіку (див. рис. 1.6) [8].

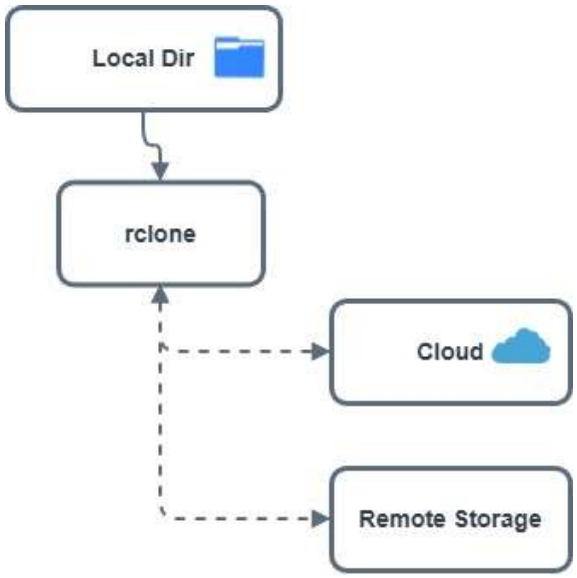


Рисунок 1.6 – Принцип роботи утиліти rclone

Додаткові можливості включають підтримку багатопотокової передачі, що дозволяє підвищити швидкість обміну даними, а також механізми обмеження пропускної здатності, які є корисними при роботі у середовищах з обмеженими мережевими ресурсами, при цьому наявність розширених засобів логування

забезпечує можливість детального аналізу виконання операцій та спрощує процес діагностики помилок. Використання rclone у межах гібридної інфраструктури дозволяє реалізувати ефективну систему синхронізації між локальними сховищами та хмарним середовищем, забезпечуючи високий рівень гнучкості та надійності процесів передачі даних.

Використання AWS CLI для автоматизації. AWS CLI (Command Line Interface) являє собою офіційний інструмент компанії Amazon для взаємодії з сервісами AWS через командний рядок, що забезпечує повний доступ до функціональних можливостей платформи, включаючи управління об’єктним сховищем Amazon S3, при цьому використання такого підходу дозволяє виконувати широкий спектр операцій без необхідності застосування графічних інтерфейсів. Основною перевагою даного інструменту є можливість інтеграції у сценарії автоматизації, що дозволяє організувати безперервні процеси синхронізації та управління даними у гібридному середовищі.

Серед доступних команд особливе значення має функція синхронізації, яка забезпечує передачу даних між локальними директоріями та S3-бакетами, виконуючи попереднє порівняння вмісту джерела та призначення з подальшою передачею лише відмінностей, що дозволяє зменшити обсяг мережевого трафіку та оптимізувати час виконання операцій. Такий механізм відповідає принципу інкрементальної передачі, що є критично важливим при роботі з великими обсягами даних або в умовах обмеженої пропускної здатності мережі [9].

Інтеграція AWS CLI з іншими сервісами платформи дозволяє формувати комплексні сценарії автоматизації, у межах яких синхронізація даних може поєднуватися з іншими операціями, такими як:

— створення бакетів;

— управління об’єктами;

— налаштування політик доступу;

— моніторинг стану ресурсів.

При цьому виконання таких сценаріїв може бути організоване за допомогою планувальників завдань, наприклад "cron", що забезпечує регулярне оновлення даних без необхідності ручного втручання (див. рис. 1.7) [9].

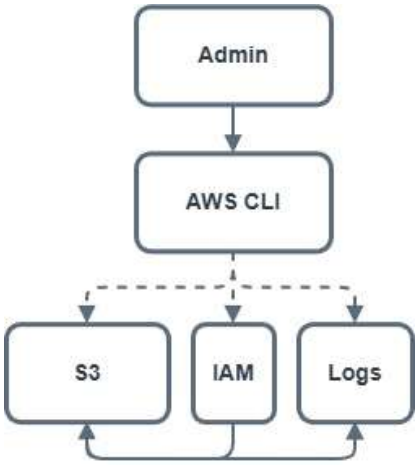


Рисунок 1.7 – Взаємодія AWS CLI з сервісами AWS

Використання механізмів автентифікації та авторизації через IAM дозволяє забезпечити безпечний доступ до ресурсів та контроль виконання операцій, оскільки кожна дія, що виконується через AWS CLI, підпорядковується встановленим політикам доступу, що відповідає вимогам до безпеки у сучасних інформаційних системах. У результаті застосування AWS CLI забезпечує високий рівень гнучкості, автоматизації та сумісності з екосистемою AWS, що робить його доцільним як у ролі самостійного інструменту синхронізації, так і у поєднанні з іншими утилітами в межах комплексних рішень.

Альтернативні рішення для реплікації даних. Окрім використання rclone та AWS CLI існує низка альтернативних інструментів і підходів, що можуть застосовуватися для організації синхронізації та резервного копіювання у гібридних системах, при цьому вибір конкретного рішення залежить від вимог до продуктивності, рівня автоматизації, безпеки та складності інфраструктури. Одним із найбільш поширених інструментів є rsync, який використовується для синхронізації файлів між локальними системами або через мережу та реалізує

алгоритм інкрементальної передачі, що дозволяє передавати лише змінені частини файлів, значно зменшуючи обсяг переданих даних [10].

Використання rsync є особливо ефективним для організації локального резервного копіювання, проте відсутність вбудованої підтримки об’єктних сховищ обумовлює необхідність застосування додаткових інструментів або проміжних рішень для інтеграції з хмарними сервісами, що ускладнює загальну архітектуру системи. Іншим підходом є використання спеціалізованих сервісів, які забезпечують інтеграцію локальної інфраструктури з хмарними сховищами, надаючи інтерфейс, подібний до традиційної файлової системи, що спрощує взаємодію з даними, але може вимагати додаткових ресурсів та супроводжуватися збільшенням витрат.

Комерційні рішення, такі як корпоративні хмарні платформи для зберігання даних, пропонують високий рівень автоматизації та зручні інтерфейси, проте їх використання часто обмежується меншою гнучкістю налаштування та зниженим рівнем контролю над даними у порівнянні з індивідуально налаштованими системами. Окрему категорію складають системи резервного копіювання, які підтримують роботу з хмарними сховищами та забезпечують додаткові функції, такі як шифрування, дедуплікація та відновлення даних, що дозволяє підвищити рівень захищеності та ефективності зберігання інформації.

Комбінований підхід, у межах якого rsync використовується для локального дублювання даних, а rclone або AWS CLI застосовуються для синхронізації з хмарним середовищем, дозволяє поєднати переваги локальної швидкодії з гнучкістю та масштабованістю хмарних сервісів, що забезпечує збалансоване рішення для побудови гібридної інфраструктури. Вибір конкретного набору інструментів визначається вимогами до обсягу даних, частоти їх оновлення, рівня безпеки та доступності, при цьому використання сучасних засобів синхронізації дозволяє мінімізувати ризики втрати інформації та забезпечити стабільну роботу системи у різних умовах експлуатації [6-10].

1.4 Безпека передачі даних у гетерогенних мережах

В умовах побудови гібридних інформаційних систем питання безпеки передачі даних набуває особливого значення, оскільки обмін інформацією відбувається між різнорідними середовищами, включаючи локальні сервери, корпоративні мережі та хмарні платформи, що формує складну гетерогенну інфраструктуру з підвищеним рівнем ризиків. Така неоднорідність мереж створює додаткові загрози, пов'язані з можливістю перехоплення, підміни або несанкціонованого доступу до інформації, особливо у випадках, коли передача даних здійснюється через відкриті канали зв'язку, зокрема через мережу Інтернет.

Передача даних у подібних умовах супроводжується потенційними загрозами, серед яких особливе місце займають [31]:

— атаки типу «людина посередині»;
— прослуховування трафіку;
— компрометація облікових даних.

Все це може призвести до втрати конфіденційності або спотворення інформації, у зв'язку з чим виникає необхідність застосування комплексних механізмів захисту, спрямованих на забезпечення конфіденційності, цілісності та автентичності даних. У межах гібридної інфраструктури, де синхронізація здійснюється між локальними вузлами та хмарними сервісами, зокрема об'єктними сховищами, захист має реалізовуватися на декількох рівнях, включаючи транспортний рівень, прикладний рівень та рівень управління доступом.

Реалізація безпечної передачі даних передбачає використання [31]:

— захищених протоколів зв'язку;
— механізмів шифрування інформації;
— ефективного управління ключами доступу.

Все це у сукупності дозволяє мінімізувати ризики несанкціонованого доступу та забезпечити стабільну роботу системи у різних умовах експлуатації. У результаті застосування таких підходів формується надійне середовище для передачі даних

між компонентами гібридної інфраструктури незалежно від їх фізичного розташування.

Протоколи захищеного транспортування (TLS/SSL). Одним із базових, але водночас критично важливих механізмів забезпечення безпеки передачі даних є використання протоколів захищеного транспортування, серед яких найбільш поширеними є TLS (Transport Layer Security) та його попередник SSL (Secure Sockets Layer), що забезпечують встановлення захищеного каналу зв'язку між клієнтом і сервером. Принцип роботи таких протоколів базується на застосуванні криптографічних алгоритмів, які дозволяють шифрувати дані під час передачі та запобігати їх перехопленню третіми сторонами [11].

Початковим етапом встановлення захищеного з'єднання є процедура узгодження параметрів взаємодії, у межах якої сторони обмінюються ключами, визначають алгоритми шифрування та здійснюють автентифікацію, що дозволяє забезпечити безпечний канал передачі інформації. Важливим елементом даного процесу є використання цифрових сертифікатів, які підтверджують автентичність сервера та дозволяють уникнути підключення до підставних ресурсів, що є критично важливим для запобігання атакам, спрямованим на перехоплення даних [11].

У процесі взаємодії з хмарними сервісами всі запити здійснюються через захищений протокол HTTPS, який використовує TLS для шифрування інформації, що забезпечує захист даних під час їх передачі між локальними компонентами системи та віддаленими ресурсами. Ефективність такого підходу безпосередньо залежить від коректності налаштування параметрів безпеки, оскільки використання застарілих версій протоколів або слабких криптографічних алгоритмів може призводити до виникнення вразливостей та створювати додаткові точки входу для потенційних атак (див. рис. 1.8) [11].

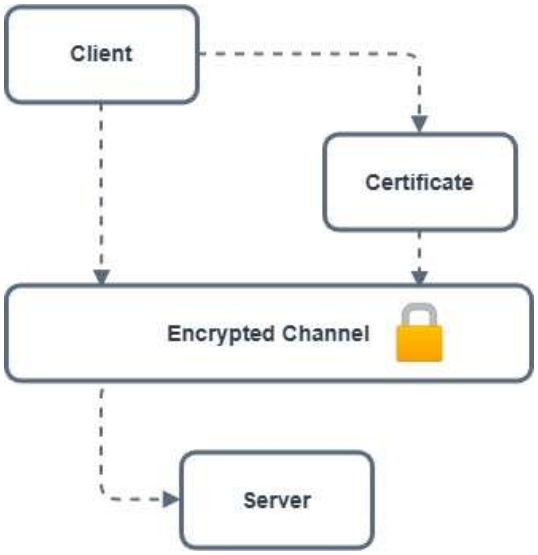


Рисунок 1.8 – Принцип роботи протоколів TLS/SSL

Застосування сучасних реалізацій TLS дозволяє досягти високого рівня захищеності даних, що робить його необхідною умовою для організації безпечної передачі інформації у гібридних мережах, особливо у випадках інтеграції локальних систем із хмарними платформами, де питання захисту даних набуває підвищеної актуальності.

Шифрування даних на стороні клієнта. Поряд із забезпеченням захисту даних під час передачі важливим аспектом є гарантування їх безпеки безпосередньо у процесі зберігання, оскільки інформація повинна залишатися захищеною не лише під час переміщення між вузлами, але й після розміщення у хмарному середовищі, у зв’язку з чим застосовується підхід клієнтського шифрування, що передбачає попереднє перетворення даних у зашифрований вигляд ще на локальному сервері перед їх відправленням у віддалене сховище. Такий механізм дозволяє забезпечити додатковий рівень конфіденційності, оскільки у хмару передається вже зашифрований вміст, що унеможливлює доступ до реальної інформації навіть у разі компрометації самого сховища або несанкціонованого доступу до нього [12].

Реалізація клієнтського шифрування базується на використанні криптографічних алгоритмів, які можуть бути як симетричними, так і асиметричними, при цьому у більшості практичних сценаріїв застосовуються

симетричні алгоритми, зокрема AES, що дозволяє досягти високої швидкості обробки даних при збереженні належного рівня захисту. У межах інструментів синхронізації, що використовуються у гібридних системах, передбачено підтримку вбудованих механізмів шифрування, які дозволяють створювати зашифровані віддалені сховища, де не лише вміст файлів, але й їх імена та структура можуть бути представлені у захищеному вигляді [12].

Такий підхід підвищує рівень конфіденційності інформації, проте супроводжується певними обмеженнями, пов’язаними зі складністю управління ключами шифрування, а також зі зниженням можливостей виконання деяких операцій безпосередньо у хмарному середовищі, включаючи пошук або обробку даних, оскільки їхній вміст залишається недоступним для аналізу без попереднього дешифрування. У контексті гібридних систем синхронізації використання клієнтського шифрування є доцільним для захисту конфіденційних даних, особливо у випадках, коли доступ до хмарного середовища здійснюється з різних мереж або через відкриті канали зв’язку, що підвищує ризики несанкціонованого доступу.

Управління ключами та токенами доступу. Ефективне управління ключами та обліковими даними є одним із визначальних аспектів забезпечення безпеки у хмарних системах, оскільки неналежне зберігання або використання ключів доступу може призвести до компрометації всієї інфраструктури та створення потенційних точок входу для зловмисників. У середовищі хмарних сервісів доступ до ресурсів здійснюється за допомогою облікових даних, які використовуються для автентифікації запитів і повинні зберігатися у захищеному вигляді з обмеженим доступом [13].

Застосування механізмів централізованого управління доступом дозволяє створювати користувачів, ролі та політики, які визначають права доступу до ресурсів, що забезпечує можливість реалізації принципу мінімальних привілеїв, відповідно до якого кожному суб’єкту надаються лише ті дозволи, які необхідні для виконання конкретних операцій. Додатковим заходом підвищення безпеки є використання тимчасових облікових даних, що мають обмежений термін дії, що

дозволяє зменшити ризики у разі їх витоку та спрощує процес управління доступом.

Важливою складовою є правильна організація зберігання ключів, що передбачає відмову від їх жорсткого кодування у програмному коді та використання захищених конфігураційних файлів або змінних середовища, які мають обмежений доступ. Додатково необхідно здійснювати регулярну ротацію ключів, що передбачає їх періодичне оновлення для зменшення ризику компрометації, а також використовувати механізми журналювання та моніторингу доступу, що дозволяє виявляти підозрілу активність і своєчасно реагувати на потенційні інциденти [13].

Поєднання зазначених підходів дозволяє сформувати безпечне середовище для синхронізації даних між локальними компонентами та хмарними ресурсами, забезпечуючи контроль доступу до інформації та мінімізуючи ризики, пов’язані з несанкціонованим використанням облікових даних у межах гібридної інфраструктури.

1.5 Нормативні вимоги до зберігання інформації

Нормативні вимоги до зберігання інформації у сучасних інформаційних системах формуються не лише на основі технічних характеристик інфраструктури, а й під впливом організаційних стандартів, внутрішніх політик компаній та міжнародних регуляторних підходів, що пов’язано з тим, що дані дедалі частіше містять конфіденційну, персональну або критично важливу інформацію, втрата чи компрометація якої може призвести до значних фінансових втрат, порушення бізнес-процесів і суттєвих репутаційних ризиків, тому питання їх правильного зберігання, обробки та контролю набуває системного характеру і потребує комплексного підходу, який охоплює всі етапи взаємодії з інформацією.

У межах гібридної інфраструктури, де дані одночасно розміщуються у локальних середовищах і хмарних платформах, складність дотримання нормативних вимог зростає, оскільки виникає необхідність узгодження політик безпеки між різними типами середовищ, забезпечення єдиного рівня контролю доступу, а також підтримання цілісності та узгодженості інформації під час її переміщення між компонентами системи, при цьому дотримання встановлених норм дозволяє підвищити загальний рівень надійності інфраструктури та мінімізувати ризики втрати або несанкціонованого доступу до даних.

Політики життєвого циклу даних. Життєвий цикл даних охоплює повний період їх існування, починаючи від моменту створення і завершуючи видаленням або архівуванням, причому кожен із цих етапів супроводжується певними правилами та обмеженнями, які визначають порядок обробки інформації та її зберігання, і ефективне управління цим циклом дозволяє оптимізувати використання ресурсів, підвищити рівень безпеки та забезпечити відповідність нормативним вимогам, оскільки на етапі створення важливо забезпечити коректність і цілісність даних шляхом перевірки вхідної інформації, фіксації метаданих і контролю формату, що особливо актуально для систем, де дані надалі підлягають синхронізації між локальним сервером і хмарним сховищем [14].

Подальший етап зберігання передбачає визначення оптимального місця розміщення інформації з урахуванням її важливості, частоти доступу та вимог до швидкодії, при цьому у гібридних системах частина даних може залишатися на локальних носіях для забезпечення швидкого доступу, тоді як інша частина переноситься до хмарних сервісів, де використовуються різні класи зберігання, що дозволяє оптимізувати витрати за рахунок вибору відповідного рівня доступності та продуктивності, а етап використання включає контроль доступу до інформації, автентифікацію користувачів, авторизацію їх дій і ведення журналів операцій, що дозволяє відстежувати всі взаємодії з даними та своєчасно реагувати на потенційні інциденти безпеки (див. рис. 1.9) [14].

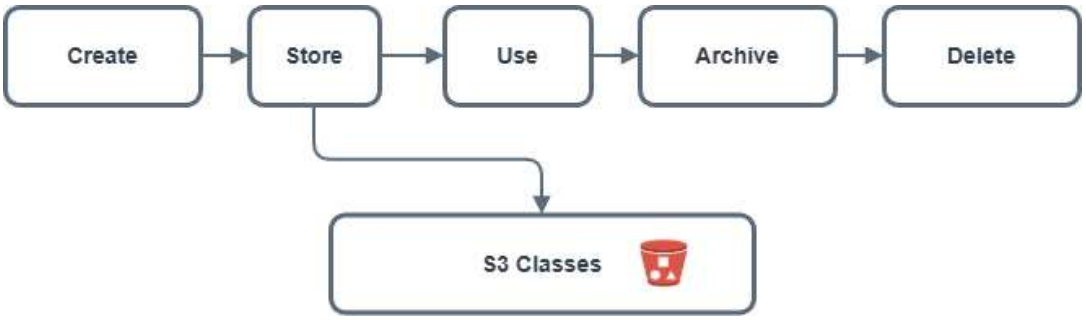


Рисунок 1.9 – Політики життєвого циклу об’єктів у S3

Архівування інформації передбачає переміщення даних, які рідко використовуються, до менш продуктивних, але економічно вигідніших сховищ, що дає можливість зменшити витрати на інфраструктуру, при цьому у хмарних середовищах такі процеси можуть автоматизуватися за допомогою політик життєвого циклу, які визначають умови переходу об’єктів між різними класами зберігання залежно від часу їх неактивності, а завершальний етап видалення повинен здійснюватися відповідно до встановлених правил і нормативів, що дозволяє уникнути накопичення застарілих або непотрібних даних і водночас гарантує, що важлива інформація не буде втрачена передчасно.

Відмовостійкість та стратегії 3-2-1. Відмовостійкість систем зберігання даних визначається здатністю інфраструктури зберігати працездатність і забезпечувати доступ до інформації навіть у випадку виникнення збоїв, відмов обладнання або зовнішніх впливів, причому у контексті гібридних середовищ така властивість набуває особливого значення через розподілений характер архітектури, де дані одночасно розміщуються у локальних та хмарних компонентах, що створює додаткові вимоги до [15]:

— узгодженості;
— резервування;
— механізмів відновлення.

Доступність інформації у таких умовах повинна підтримуватися незалежно від стану окремих елементів системи.

Одним із найбільш поширених і практично перевірених підходів до забезпечення надійності зберігання даних є стратегія резервного копіювання 3-2-1, яка передбачає наявність щонайменше трьох копій інформації, використання двох різних типів носіїв і розміщення однієї копії поза межами основної інфраструктури, що дозволяє знизити ризики втрати даних унаслідок апаратних несправностей, помилок користувачів або цілеспрямованих атак, при цьому у гібридній моделі така стратегія природно інтегрується у загальну архітектуру, де основна копія може зберігатися на локальному сервері, додаткова копія — на резервному локальному носії або окремому сервері, а віддалена копія — у хмарному сховищі, що створює багаторівневу систему захисту (див. рис. 1.10) [15].

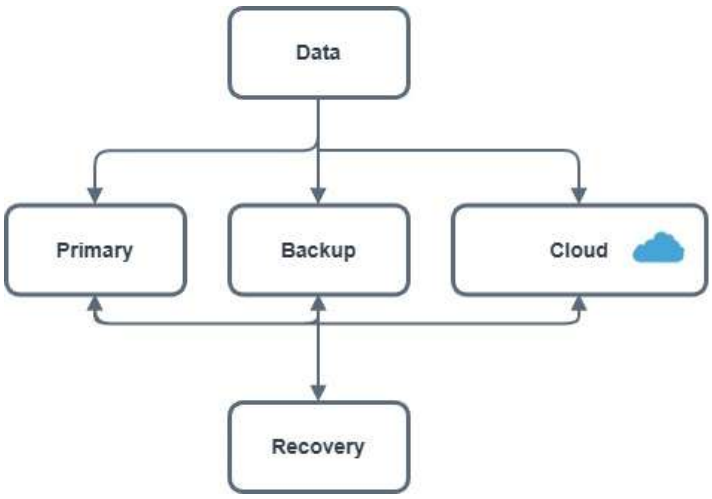


Рисунок 1.10 – Стратегія резервного копіювання 3-2-1

Реалізація такого підходу дозволяє забезпечити можливість відновлення інформації у різних сценаріях відмов, включаючи часткове пошкодження локальної інфраструктури або повну втрату доступу до неї, оскільки наявність незалежних копій гарантує, що дані залишаються доступними у принаймні одному з середовищ, а використання хмарних сервісів як віддаленого рівня резервування підвищує стійкість системи до катастрофічних подій, пов’язаних із фізичним знищенням обладнання або тривалими перебоями в роботі локальних ресурсів [14].

Ефективність резервного копіювання значною мірою залежить від регулярності створення копій і ступеня автоматизації цього процесу, оскільки

ручне виконання таких операцій підвищує ймовірність помилок і знижує актуальність резервних даних, тоді як використання сценаріїв автоматизації та планувальників завдань дозволяє забезпечити систематичне оновлення копій відповідно до заданих інтервалів часу або подій, пов’язаних зі зміною даних, що у свою чергу гарантує мінімальні втрати інформації у разі інцидентів.

Додатковим аспектом є необхідність періодичного тестування процедур відновлення, оскільки наявність резервних копій не гарантує їх працездатність без перевірки можливості фактичного відновлення, а проведення таких тестів дозволяє [14-15]:

1. виявити потенційні проблеми у конфігурації системи;
2. перевірити цілісність збережених даних;
3. оцінити час для повернення системи до робочого стану.

Все це є критично важливим для планування безперервності бізнес-процесів.

Аудит та логування операцій з даними. Аудит та логування операцій із даними формують окремий напрям забезпечення надійності та безпеки інформаційних систем, оскільки дозволяють здійснювати детальний контроль над усіма діями, що виконуються у межах інфраструктури, і забезпечують можливість відстеження як користувацьких, так і системних процесів, при цьому логування передбачає фіксацію подій, пов’язаних із доступом до даних, їх зміною, передачею та видаленням, із зазначенням ідентифікаторів користувачів, часу виконання операцій і результатів цих дій.

У гібридних системах журнали подій формуються як на локальному рівні, так і у хмарному середовищі, що створює розподілену структуру даних про активність, і об’єднання цих журналів у єдину систему моніторингу дозволяє отримати цілісне уявлення про функціонування всієї інфраструктури, а також оперативно виявляти відхилення від нормальної поведінки системи, що може свідчити про помилки конфігурації або спроби несанкціонованого доступу [16].

Використання інструментів синхронізації та взаємодії з хмарними сервісами супроводжується можливістю ведення детальних журналів виконання операцій, які містять інформацію про:

— передані файли;

— обсяг даних;

— тривалість процесів;

— виниклі помилки.

Все це значно спрощує діагностику проблем і дозволяє контролювати ефективність роботи системи, а хмарні платформи, у свою чергу, надають вбудовані механізми аудиту, які забезпечують відстеження всіх дій із об’єктами, включаючи операції читання, запису та видалення [16].

Аналіз журналів подій може використовуватися не лише для забезпечення безпеки, а й для оптимізації роботи системи, оскільки інформація про частоту доступу до даних, обсяг переданої інформації та характер операцій дозволяє приймати обґрунтовані рішення щодо розміщення даних, їх архівування або переміщення між різними рівнями зберігання, що сприяє підвищенню ефективності використання ресурсів [16].

Захист журналів є не менш важливим завданням, ніж захист основних даних, оскільки вони можуть містити чутливу інформацію про структуру системи, користувачів і виконувані операції, тому застосування політик контролю доступу, шифрування та резервного копіювання журналів дозволяє запобігти їх несанкціонованому використанню або втраті, а інтеграція механізмів аудиту та логування у загальну архітектуру програмно-апаратного комплексу забезпечує підвищення рівня контролю над процесами синхронізації, своєчасне виявлення помилок і стабільність функціонування всієї системи.

2. ПРАКТИЧНА РОЗРОБКА КОМПЛЕКСУ СИНХРОНІЗАЦІЇ НА БАЗІ UBUNTU SERVER ТА AWS

2.1 Архітектура програмно-апаратного комплексу

Практична реалізація комплексу синхронізації даних у гібридній локально-хмарній інфраструктурі базується на поєднанні віртуалізованого локального середовища з хмарною платформою, що забезпечує гнучкість розгортання, масштабованість та можливість моделювання реальних умов експлуатації без значних апаратних витрат, а також дозволяє організувати дублювання і автоматичну передачу даних між різними середовищами з мінімальним втручанням з боку користувача в межах такого рішення реалізовано багаторівневу модель, яка включає [30]:

— локальний вузол;

— резервний рівень зберігання;

— хмарну інфраструктуру.

При цьому локальний вузол функціонує на базі серверної операційної системи, розгорнутої у віртуальному середовищі, резервування здійснюється локально шляхом дзеркалювання файлової структури, а передача у хмару виконується за допомогою спеціалізованих інструментів взаємодії з об’єктним сховищем (див. рис. 2.1).

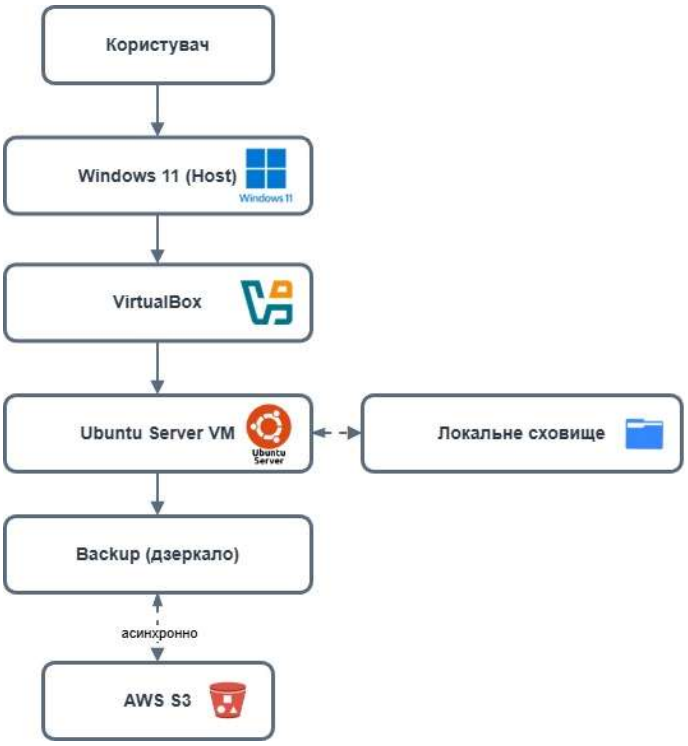


Рисунок 2.1 – Архітектура програмно-апаратного комплексу синхронізації на базі Ubuntu Server та AWS

Така організація дозволяє забезпечити розподіл навантаження між локальними та віддаленими ресурсами, підвищити відмовостійкість системи та реалізувати принципи гібридного зберігання даних, при яких актуальні файли доступні локально, резервні копії зберігаються на тому ж вузлі, а додатковий рівень захисту забезпечується за рахунок хмарного середовища, у результаті чого формується цілісна інфраструктура, здатна функціонувати навіть у разі часткової відмови окремих компонентів.

З точки зору логічної структури архітектура може бути представлена як послідовність взаємодії між рівнями [30]:

1. джерело даних у робочій директорії;
2. локальне дзеркало для резервування;
3. віддалене сховище для довготривалого зберігання.

При цьому кожен рівень виконує власну функцію, але водночас інтегрований у загальний процес синхронізації, що дозволяє досягти узгодженості даних у всіх точках зберігання.

Апаратна топологія локального вузла. Реалізація локального вузла системи здійснюється на базі персонального комп'ютера під управлінням операційної системи Windows 11, яка виступає хостовою платформою для розгортання віртуалізованого середовища, при цьому для створення ізольованого серверного середовища використовується гіпервізор Oracle VM VirtualBox, що забезпечує можливість запуску віртуальних машин без необхідності використання окремого фізичного серверного обладнання.

У межах цієї віртуальної інфраструктури розгортається серверна операційна система Ubuntu Server 22.04 LTS, вибір якої обумовлений її стабільністю, низькими вимогами до ресурсів та широкою підтримкою в середовищі адміністрування, відсутність графічного інтерфейсу дозволяє мінімізувати споживання ресурсів і спрямувати їх на виконання прикладних задач, пов'язаних із обробкою та синхронізацією даних (див. рис. 2.2) [17].

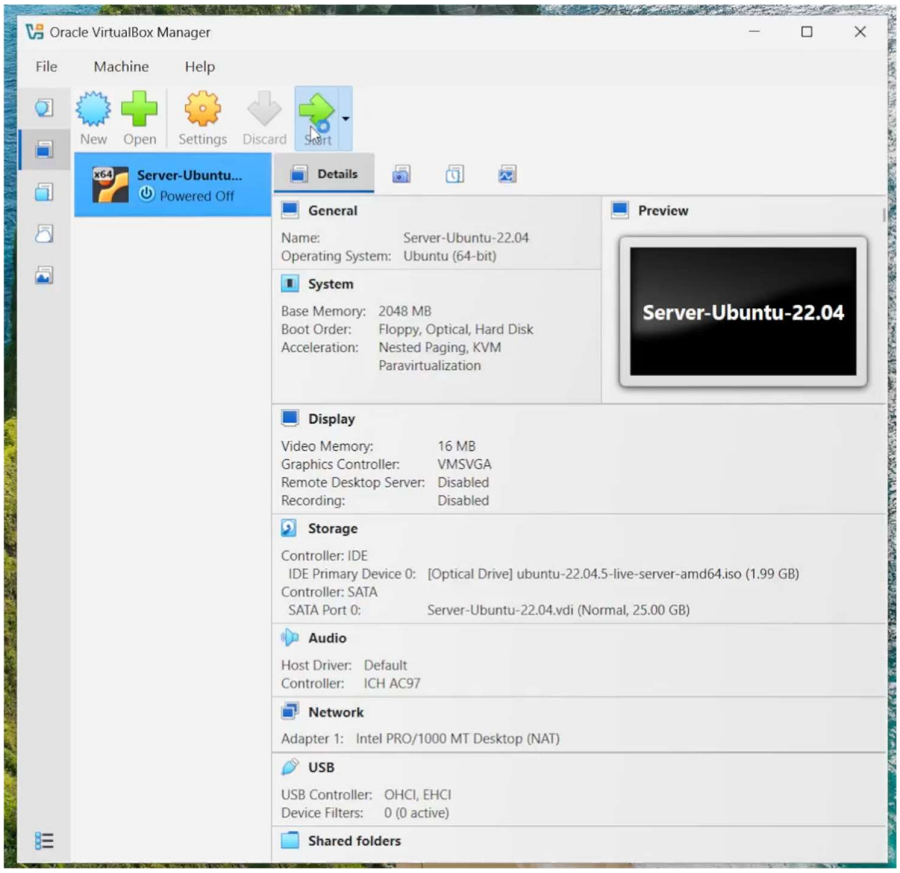


Рисунок 2.2 – Віртуалізоване середовище на базі Ubuntu Server

Конфігурація віртуального середовища включає (див. табл. 2.1):

— виділення оперативної пам’яті в обсязі від 2 до 4 ГБ;

— використання двох віртуальних ядер процесора;

— дискового простору в межах 30–50 ГБ.

Цього достатньо для зберігання робочих даних і створення резервних копій у межах тестового або навчального середовища, така конфігурація дозволяє відтворити типові умови функціонування серверної системи без перевантаження апаратних ресурсів хостової машини.

Таблиця 2.1 – Характеристики апаратного та програмного середовища

Параметр	Значення
Хостова ОС	Windows 11
Гіпервізор	Oracle VM VirtualBox
Гостьова ОС	Ubuntu Server 22.04 LTS
CPU	2 віртуальні ядра
RAM	2 ГБ
Диск	30ГБ
Мережа	Bridge (окрема IP-адреса)
Основне ПЗ	AWS CLI, rclone

Мережева взаємодія налаштована у режимі Bridge, що дозволяє віртуальній машині використовувати мережевий інтерфейс хостової системи та отримувати власну IP-адресу в локальній мережі, наприклад 192.168.0.126, у результаті чого вона функціонує як повноцінний мережевий вузол, доступний для інших пристроїв, така конфігурація забезпечує можливість підключення до сервера через SSH, передачі файлів та віддаленого адміністрування.

З точки зору структурної організації апаратна топологія може бути представлена як ієрархічна модель, у якій фізичний комп’ютер із встановленою операційною системою виступає базовим рівнем, поверх якого функціонує гіпервізор, що створює ізольоване середовище для віртуальної машини, всередині якої розгортається серверна операційна система та прикладні компоненти, включаючи файлові директорії, служби синхронізації та інструменти взаємодії з хмарою.

Схема взаємодії між клієнтом, локальним сервером та AWS S3. Архітектура комплексу синхронізації побудована за принципом гібридної моделі, у якій дані одночасно зберігаються у локальному середовищі та у хмарному сховищі, що дозволяє поєднати переваги швидкого локального доступу з підвищеною надійністю віддаленого зберігання, при цьому логічна структура системи передбачає наявність робочої директорії, локального резервного сховища та віддаленого об’єктного сховища на базі Amazon Web Services, зокрема сервісу Amazon S3, який забезпечує довготривале зберігання даних і підвищення відмовостійкості всієї інфраструктури.

У межах такої моделі робоча директорія, наприклад "/data/main", виступає основним середовищем взаємодії користувача або прикладної системи з файлами, де відбувається створення, редагування та видалення даних, після внесення змін запускається механізм синхронізації, який забезпечує копіювання модифікованих файлів у локальне резервне сховище "/data/local_backup", що функціонує як дзеркало основної директорії, у результаті чого формується проміжний рівень зберігання, здатний швидко відновити дані у разі локальних збоїв.

Подальший етап передбачає передачу підготовлених даних до хмарного середовища, де використовується спеціалізований інструмент синхронізації, який взаємодіє з API об’єктного сховища, передача виконується не безпосередньо з робочої директорії, а саме з локального резервного рівня, що дозволяє уникнути дублювання обчислень і мінімізувати обсяг переданого трафіку, оскільки попередня обробка вже відфільтровує незмінені файли.

Важливою особливістю функціонування такої схеми є асинхронний характер обміну даними, при якому операції синхронізації не блокують роботу користувача з файловою системою, локальна копія виступає буфером між клієнтським рівнем і хмарним середовищем, що дозволяє забезпечити стабільність системи навіть за умов нестабільного або відсутнього мережевого з’єднання, при цьому у випадку тимчасової недоступності хмарного сховища всі зміни накопичуються на локальному рівні і передаються до віддаленого середовища після відновлення з’єднання [18].

У разі виникнення аварійної ситуації, пов’язаної з втратою доступу до локального середовища, відновлення може бути виконане з хмарного сховища, що забезпечує додатковий рівень захисту інформації, у протилежному випадку, при відмові мережевого з’єднання, система продовжує функціонувати локально без втрати працездатності, що формує гнучку та стійку до збоїв архітектуру.

Логіка взаємодії між компонентами може бути представлена як послідовний процес обробки даних, у якому кожен рівень виконує власну функцію, але водночас інтегрований у загальну систему синхронізації, забезпечуючи узгодженість стану даних у всіх точках зберігання (див. рис. 2.3).

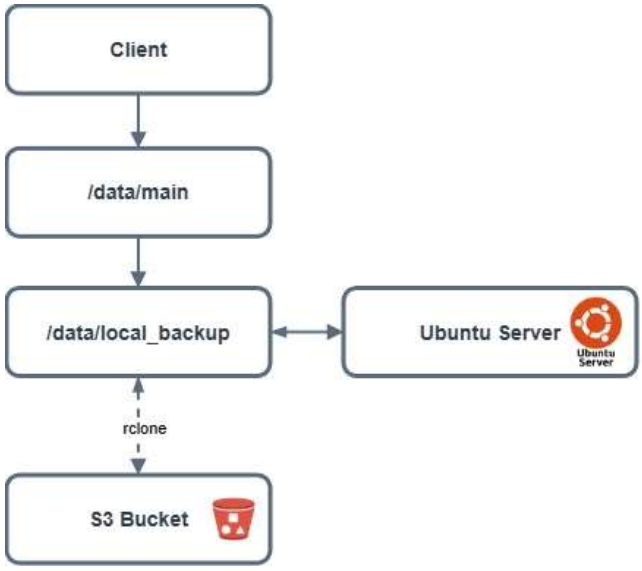


Рисунок 2.3 – Схема взаємодії клієнта, локального сервера та AWS S3

Вибір та обґрунтування програмного стека. Для реалізації функціоналу синхронізації та резервного копіювання було обрано набір інструментів з відкритим вихідним кодом, що забезпечують необхідний рівень гнучкості, надійності та адаптивності до різних сценаріїв використання, причому така сукупність компонентів дозволяє формувати цілісну систему без прив’язки до конкретного програмного або хмарного постачальника, а також забезпечує можливість подальшого масштабування та модифікації архітектури без суттєвих змін у її базовій логіці, при цьому ключовими складовими програмного стека

виступають утиліти для локальної синхронізації, взаємодії з хмарними сервісами та автоматизації виконання завдань, що у сукупності формують єдиний процес обробки та передачі даних.

Основним інструментом для локальної синхронізації та дзеркалювання даних виступає rsync, який забезпечує ефективну передачу інформації за рахунок використання алгоритмів інкрементального копіювання, де обробляються лише змінені частини файлів, що значно зменшує обсяг переданих даних та знижує навантаження на систему зберігання і обчислювальні ресурси, при цьому додатково реалізується механізм перевірки цілісності за допомогою контрольних сум, що дозволяє уникнути пошкодження або втрати інформації у процесі синхронізації, а логіка роботи може бути представлена як послідовний процес, у якому виконується аналіз структури файлів, порівняння їхніх станів, визначення відмінностей, передача лише необхідних змін та оновлення резервного сховища, що формує ефективний і ресурсозберігаючий підхід до локального дублювання даних.

Для взаємодії з хмарним середовищем використовується інструмент rclone, який підтримує широкий спектр хмарних провайдерів та реалізує універсальний механізм роботи через API, що дозволяє організувати передачу даних у середовище Amazon S3 без жорсткої прив’язки до конкретної платформи, а також забезпечує можливість інтеграції з іншими сервісами у разі зміни інфраструктури, причому до основних переваг належить підтримка S3-сумісних інтерфейсів, можливість шифрування даних перед передачею, гнучке налаштування параметрів синхронізації, включаючи обмеження швидкості, кількість потоків та політики оновлення, а також детальне логування всіх операцій, що дозволяє контролювати процес передачі та аналізувати можливі помилки, при цьому алгоритм роботи включає визначення змін у локальному резервному сховищі, формування списку об’єктів для передачі, виконання передачі через API хмарного сервісу та фіксацію результатів у журналах, що забезпечує прозорість і контрольованість процесу синхронізації [19].

Автоматизація процесів реалізується за допомогою планувальника задач cron, який дозволяє виконувати скрипти у визначені часові інтервали без участі користувача, що забезпечує регулярність оновлення даних та підтримання їх актуальності, при цьому сценарій автоматизації передбачає запуск скрипта синхронізації у нічний час, наприклад о 03:00, що дозволяє мінімізувати вплив на продуктивність системи у періоди активного використання ресурсів, а також забезпечує стабільність роботи сервісів та зниження навантаження на мережу, причому взаємодія компонентів у цьому випадку може бути описана як послідовний процес, у якому планувальник ініціює виконання скрипта, скрипт запускає локальну синхронізацію через rsync, після чого виконується передача даних у хмарне середовище за допомогою rclone, а результати виконання фіксуються у логах, що дозволяє відстежувати стан системи та виявляти можливі відхилення у роботі (див. рис. 2.4).

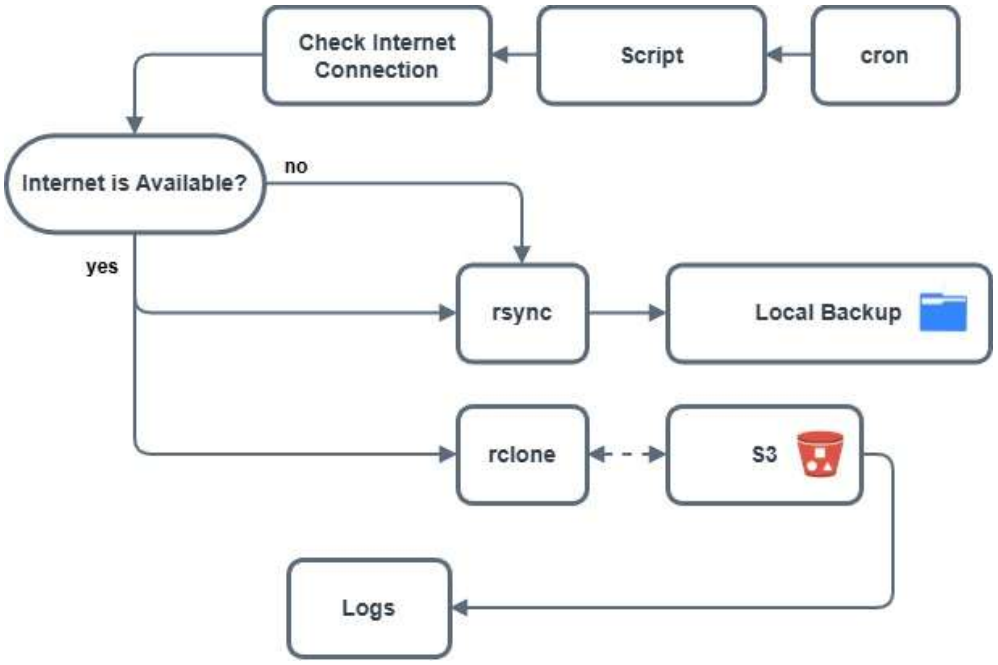


Рисунок 2.4 – Структура програмного стека системи

Сформований програмний стек забезпечує ефективне поєднання швидкості локальної обробки даних із надійністю хмарного зберігання, при цьому використання відкритих інструментів дозволяє досягти незалежності від

конкретного постачальника послуг, спростити процес налаштування та адаптації системи, а також забезпечити можливість розширення функціоналу без значних змін у загальній архітектурі, що у підсумку формує збалансоване рішення, придатне для використання у різних умовах експлуатації, включаючи як тестові середовища, так і практичні сценарії застосування у невеликих та середніх інформаційних системах.

2.2 Етапи розгортання та конфігурації системи

Практична реалізація комплексу синхронізації у гібридній локально-хмарній інфраструктурі передбачає послідовне виконання низки взаємопов’язаних етапів, що охоплюють підготовку локального серверного середовища, налаштування хмарної інфраструктури та інтеграцію інструментів для забезпечення обміну даними між цими середовищами, причому кожен із зазначених етапів формує окремий рівень системи та безпосередньо впливає на її стабільність, продуктивність і надійність, а загальна логіка розгортання визначається необхідністю створення узгодженого середовища, у якому локальні ресурси та хмарні сервіси взаємодіють як єдина система з мінімальними затримками та контрольованими механізмами обробки даних, при цьому умовний поділ процесу на підготовку операційної системи, конфігурацію хмарного провайдера та налаштування засобів доступу і синхронізації дозволяє структуровано реалізувати кожен компонент і забезпечити їхню коректну взаємодію на етапі експлуатації.

Підготовка базової ОС та апаратного сховища. Перший етап реалізації системи полягає у розгортанні серверного середовища на базі операційної системи Ubuntu Server 22.04 LTS, інсталяція якої виконується у віртуальній машині, створеній у середовищі VirtualBox із попередньо визначеними апаратними параметрами, що відповідають вимогам стабільної роботи серверних служб, при цьому обирається мінімальна конфігурація без графічного інтерфейсу, що дозволяє

зменшити споживання ресурсів і зосередити обчислювальну потужність на виконанні прикладних задач, а налаштування мережевого інтерфейсу забезпечує стабільне підключення до локальної мережі та доступ до зовнішніх ресурсів, що є необхідною умовою для подальшої інтеграції з хмарними сервісами, після завершення інсталяції виконується оновлення системних пакетів і встановлення необхідного програмного забезпечення, до якого належать rsync для локальної синхронізації, rclone для взаємодії з хмарними сховищами, awscli для роботи із сервісами Amazon Web Services через командний рядок, а також openssh-server для забезпечення віддаленого доступу до системи, причому встановлення здійснюється за допомогою стандартного менеджера пакетів apt, що гарантує сумісність компонентів із системою та спрощує процес подальшого оновлення (див. рис. 2.5).

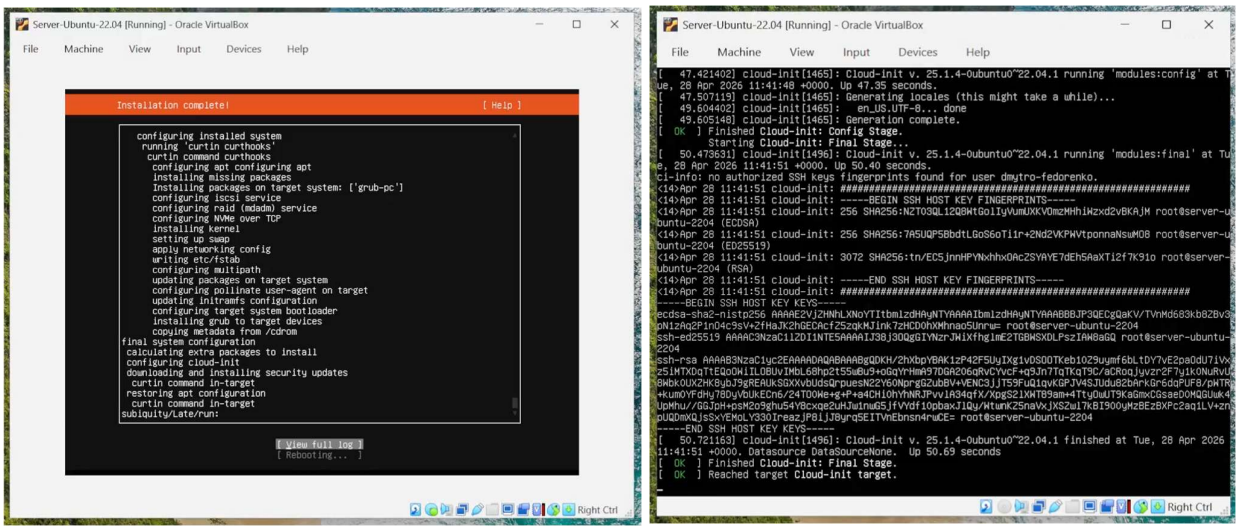


Рисунок 2.5 – Встановлення та налаштування Ubuntu Server

Окрему увагу приділено організації файлової структури, де створюються дві основні директорії, а саме (див. рис. 2.6):

1. "/data/main" як робоче середовище для зберігання активних даних.
2. "/data/local_backup" як локальне резервне сховище, що виконує функцію дзеркала.



Рисунок 2.6 – Структура директорій локального сховища

Така структура дозволяє логічно розмежувати дані та їх копії, спрощуючи процес синхронізації, дедуплікації та відновлення інформації, додатково налаштовується служба SSH, яка забезпечує можливість підключення до сервера з хостової машини під управлінням Windows 11, що дозволяє передавати конфігураційні файли, скрипти та виконувати адміністрування системи віддалено, при цьому процес підготовки формує послідовний ланцюг дій, що включає (див. рис. 2.7):

1. Інсталяцію операційної системи.
2. Оновлення пакетів.
3. Встановлення необхідного програмного забезпечення.
4. Створення файлової структури.
5. Налаштування віддаленого доступу.

Отриманий результат виступає стабільною основою для подальшої інтеграції локального вузла з хмарною інфраструктурою.

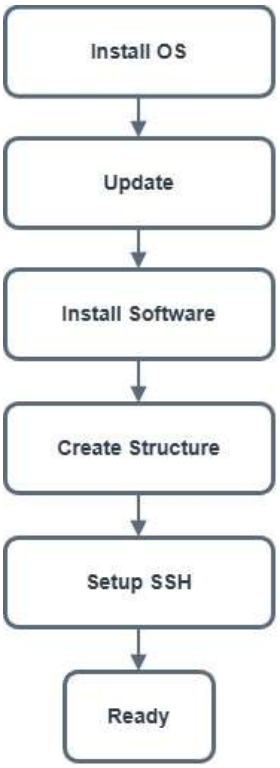


Рисунок 2.7 – Загальна послідовність розгортання системи

Схема взаємодії між клієнтом, локальним сервером та AWS S3. Другим важливим етапом є конфігурація хмарного середовища, яке забезпечує віддалене зберігання даних та виступає одним із базових елементів гібридної архітектури, причому у межах цього етапу формується не лише сховище як таке, а й механізми контролю доступу до нього, що безпосередньо впливають на безпеку та керованість усієї системи, а реалізація здійснюється у середовищі Amazon Web Services із використанням сервісу об’єктного зберігання, де створюється S3-бакет у регіоні eu-central-1 (Франкфурт), вибір якого обумовлений географічною близькістю та дозволяє зменшити затримки при передачі даних, що позитивно впливає на швидкість синхронізації та загальну продуктивність системи, при цьому під час створення бакета застосовуються сучасні рекомендації безпеки, зокрема відключення механізму ACL через встановлення параметра Object Ownership у режимі enforced, що дозволяє централізувати управління доступом через IAM-політики та уникнути потенційних конфліктів прав доступу між різними механізмами контролю (див. рис. 2.8) [19; 31].

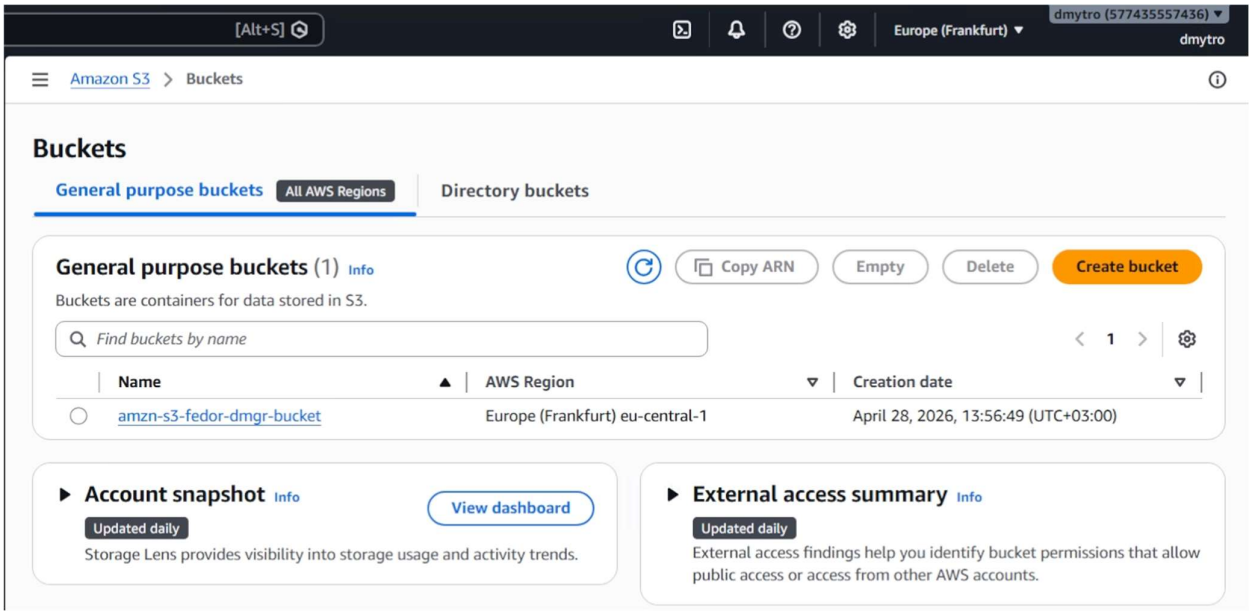


Рисунок 2.8 – Конфігурація S3 бакета

У межах організації безпечної взаємодії з хмарним середовищем створюється окремий IAM-користувач із обмеженими правами доступу, наприклад користувач типу "sync-fedor-dmgr", якому надаються лише ті дозволи, що необхідні для виконання операцій синхронізації, причому така модель відповідає принципу мінімальних привілеїв, коли кожен обліковий запис має доступ лише до конкретного набору ресурсів і дій, що знижує ризики несанкціонованого доступу та помилок конфігурації, а сама політика доступу формується у форматі JSON і включає дозволи на виконання базових операцій з об’єктами, таких як [19; 31]:

1. "s3:PutObject" для завантаження файлів.
2. "s3:GetObject" для їх отримання.
3. "s3:DeleteObject" для видалення.
4. "s3:ListBucket" для перегляду вмісту.

При цьому дія політики обмежується лише конкретним бакетом, що забезпечує ізоляцію даних і контроль над доступом до них (див. рис. 2.9).

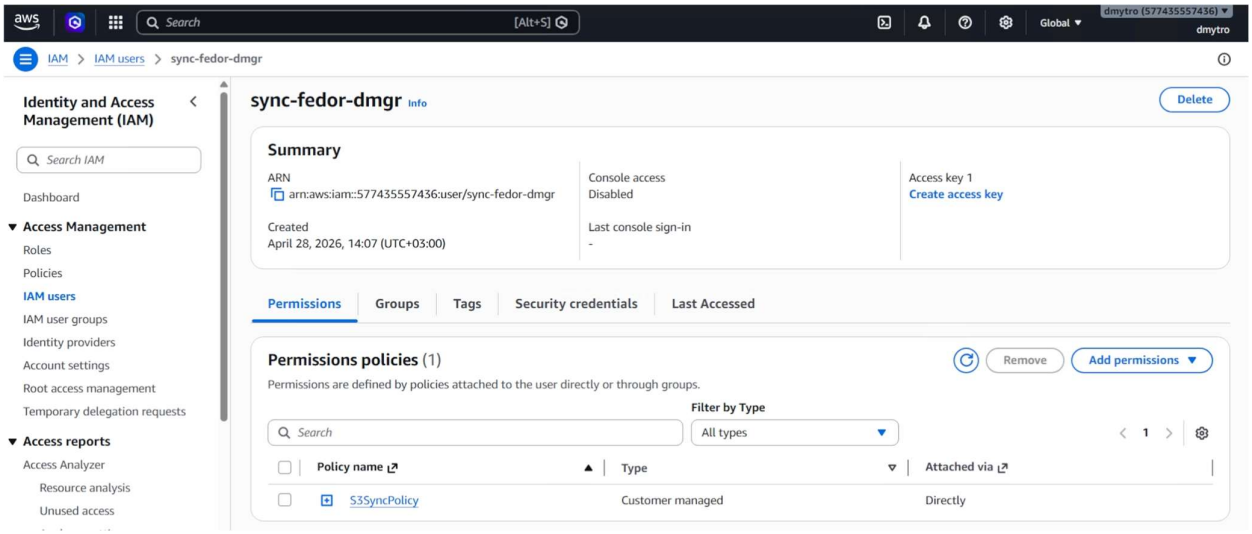


Рисунок 2.9 – Налаштування IAM користувача в AWS

Після створення користувача генеруються ключі доступу, зокрема:

— Access Key ID;

— Secret Access Key.

Вони зберігаються у вигляді файлу формату CSV та використовуються для автентифікації під час роботи з інструментами командного рядка, а їх зберігання повинно здійснюватися у захищеному середовищі з обмеженим доступом, оскільки компрометація цих даних може призвести до несанкціонованого доступу до хмарного сховища, при цьому взаємодія між локальним сервером і хмарною інфраструктурою формується як зв’язок, у якому локальна система проходить процедуру автентифікації за допомогою отриманих ключів і виконує операції над S3-бакетом у межах визначених прав, а застосування чітко сформованих політик доступу дозволяє контролювати всі дії з даними, забезпечуючи високий рівень безпеки та відповідність вимогам до захисту інформації у розподілених системах зберігання (див. рис. 2.10) [20; 31].



Рисунок 2.10 – Отримані Access Key ID та Secret Access Key
для новоствореного користувача

Вибір та обґрунтування програмного стека. Завершальний етап розгортання передбачає інтеграцію локального середовища з хмарною платформою за допомогою інструментів командного рядка, серед яких основними виступають:

— AWS CLI;

— rclone.

Саме вони забезпечують можливість виконання операцій із хмарним сховищем безпосередньо з серверного середовища, а налаштування AWS CLI виконується за допомогою команди конфігурації, у процесі якої вводяться (див. рис. 2.11):

— ключі доступу;

— регіон розташування сервісу;

— формат виводу даних.

Після чого система отримує можливість надсилати запити до сервісів Amazon Web Services через термінал, що формує базовий рівень інтеграції з хмарною інфраструктурою та забезпечує доступ до ресурсів на рівні API.

```
dmytro-fedorenko@server-ubuntu-2204:~$ aws configure
AWS Access Key ID [None]: AKIAYM4OXLY6AOBS5ZFQ
AWS Secret Access Key [None]: [redacted]
Default region name [None]: eu-central-1
Default output format [None]: json
dmytro-fedorenko@server-ubuntu-2204:~$
```

Рисунок 2.11 – Налаштування AWS CLI у терміналі

Наступним кроком виконується конфігурація rclone, у процесі якої створюється віддалене сховище типу "remote", що асоціюється із S3-бакетом і використовується як логічне представлення хмарного ресурсу у локальній системі, причому важливою особливістю є використання параметра "env_auth", який дозволяє rclone застосовувати вже налаштовані облікові дані AWS CLI, що усуває необхідність дублювання ключів доступу у конфігураційних файлах і знижує ризик виникнення помилок автентифікації, а також спрощує управління обліковими даними, оскільки всі параметри доступу централізовано зберігаються у середовищі AWS CLI, при цьому відсутність використання такого підходу часто призводить до виникнення помилок типу "403 Access Denied", пов’язаних із некоректними або застарілими ключами доступу, тому застосування "env_auth" забезпечує більш стабільну та передбачувану роботу системи (див. рис. 2.12) [21].

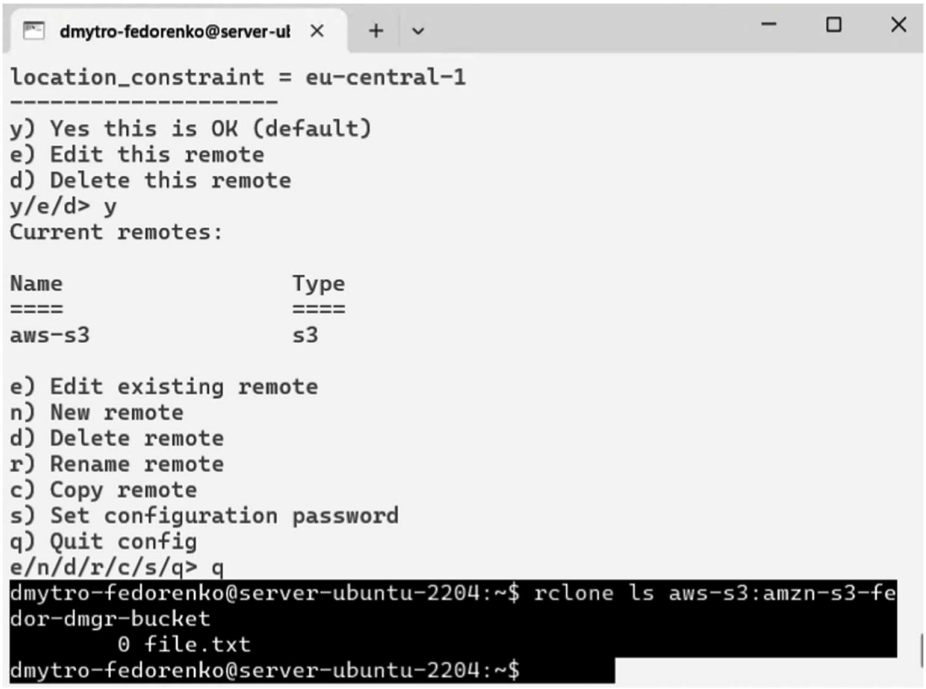


Рисунок 2.12 – Конфігурація rclone (інтерактивне меню)

Процес інтеграції може бути представлений як послідовність дій, у якій виконується [21-22]:

1. введення ключів доступу;

2. налаштування aws cli;
3. створення remote у rclone;
4. перевірка доступності хмарного ресурсу;
5. підтвердження готовності до синхронізації.

Причому після завершення конфігурації здійснюється тестове підключення до S3-бакета, що дозволяє перевірити коректність налаштувань і переконатися у можливості виконання операцій з даними, а у разі успішного результату система переходить у стан готовності до реалізації механізмів синхронізації та автоматизації, при цьому сформоване середовище забезпечує безпечну та ефективну взаємодію між локальним сервером і хмарним сховищем, що створює основу для подальшого впровадження алгоритмів реплікації даних та автоматизованих процесів у межах гібридної інфраструктури (див. табл. 2.2).

Таблиця 2.2 – Основні параметри конфігурації AWS та rclone

Параметр	Значення
Тип автентифікації	Access Key зі Secret Key
Регіон	eu-central-1
Тип сховища	Amazon S3 Bucket
Формат виводу AWS CLI	JSON
rclone remote	Тип S3
Метод доступу	env_auth (через AWS CLI)
Endpoint	AWS S3 API

2.3 Алгоритми роботи системи реплікації

Завершальний етап формування працездатної системи синхронізації передбачає побудову чітко визначеної логіки обробки даних, яка поєднує локальні механізми дублювання та віддалену передачу у хмарне середовище, при цьому інтеграція інструментів командного рядка забезпечує виконання всіх операцій безпосередньо у серверному середовищі, що дозволяє мінімізувати залежність від

графічних інтерфейсів і підвищити рівень автоматизації, а використання таких засобів як AWS CLI та rclone формує базовий рівень взаємодії з хмарною інфраструктурою через API, де налаштування AWS CLI виконується шляхом введення ключів доступу, визначення регіону та формату виводу, після чого система отримує можливість виконувати запити до сервісів Amazon Web Services через термінал, у той час як rclone виступає надбудовою для організації безпосередньої синхронізації даних між локальним і віддаленим середовищем, а сформована таким чином інтеграція створює основу для реалізації алгоритмів реплікації, що функціонують у вигляді послідовності взаємопов'язаних операцій із контролем стану та журналюванням результатів.

Створення директорій та написання bash-скрипта. Початковим етапом реалізації алгоритму виступає формування логічної структури зберігання даних, яка відповідає концепції гібридної архітектури, де робоча директорія "/data/main" використовується як основне середовище розміщення актуальних файлів, тоді як директорія "/data/local_backup" виконує функцію локального резервного сховища, що виступає проміжним буфером перед передачею у хмару, причому створення таких директорій здійснюється стандартними засобами операційної системи із подальшим налаштуванням прав доступу, що забезпечує коректну роботу сервісних процесів і запобігає конфліктам під час одночасного доступу до файлів.

Центральним елементом управління процесами синхронізації виступає bash-скрипт "sync_script.sh", який об'єднує всі етапи обробки даних у єдину логічну послідовність, при цьому у структурі скрипта визначаються змінні для шляхів до директорій, параметри логування, а також виклики відповідних утиліт для виконання синхронізації, що дозволяє стандартизувати процес і забезпечити його повторюваність, а для надання можливості виконання скрипта застосовується команда зміни прав доступу "chmod +x sync_script.sh", що переводить файл у категорію виконуваних [20].

Логіка роботи скрипта передбачає послідовне виконання операцій із фіксацією результатів у журналі, де на початковому етапі здійснюється ініціалізація параметрів, після чого виконується локальна синхронізація, перевірка

її статусу, передача даних у хмару та завершення процесу із записом відповідної інформації у лог-файл, а така структура може бути представлена у вигляді лінійної блок-схеми, де кожен етап переходить до наступного лише за умови успішного виконання попереднього, що забезпечує контрольованість процесу та можливість швидкої діагностики у разі виникнення помилок (див. рис. 2.13).

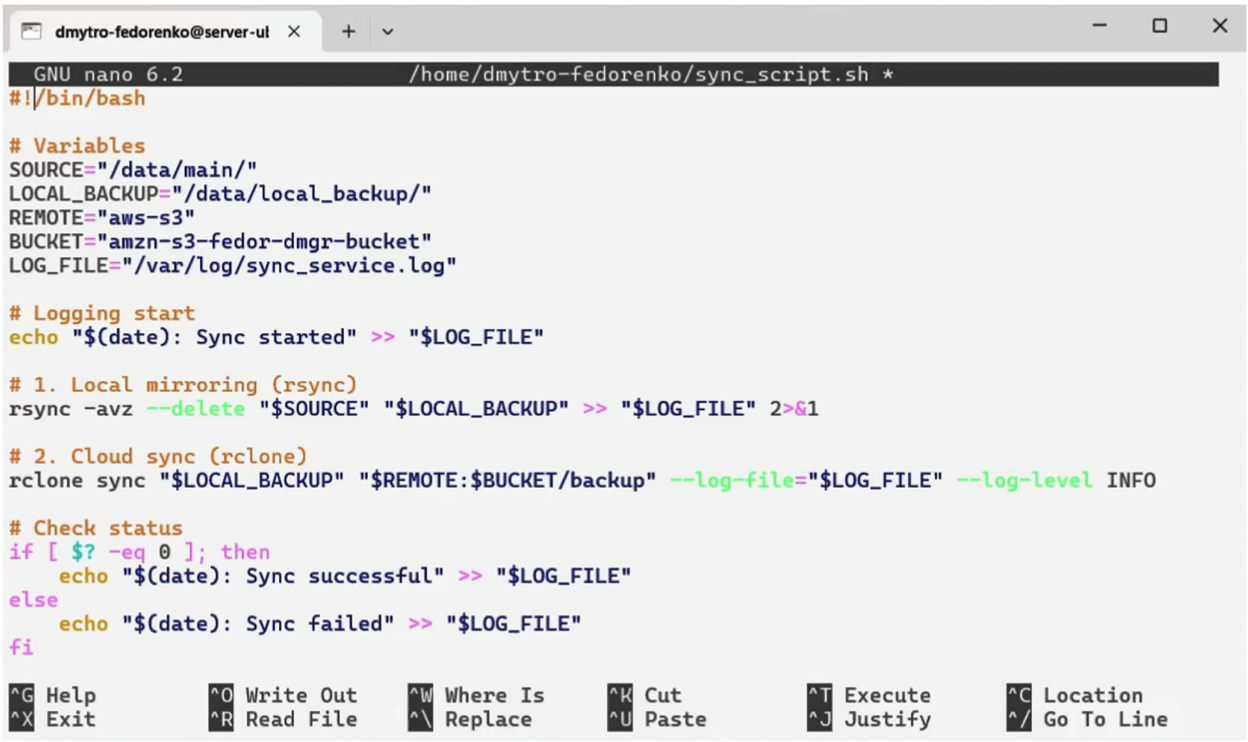


Рисунок 2.13 – Структура bash-скрипта синхронізації

Алгоритм локального дублювання даних. Першим етапом виконання скрипта є синхронізація даних між робочою директорією та локальним резервним сховищем, де використовується утиліта rsync, яка реалізує інкрементний підхід до передачі інформації з передачею лише змінених файлів або їх частин, що дозволяє значно зменшити навантаження на систему та підвищити швидкість обробки даних, при цьому базова команда синхронізації включає набір параметрів, які визначають поведінку процесу [23]:

1. параметр "-a" забезпечує збереження атрибутів файлів включаючи права доступу, часові мітки та структуру директорій;

2. параметр "-v" активує детальний режим виводу інформації про виконання операцій;

3. параметр "-z" відповідає за стиснення даних під час передачі;

4. параметр "--delete" виконує синхронізацію структури директорій шляхом видалення у резервному сховищі тих файлів, які були видалені у вихідній директорії.

Все це дозволяє підтримувати повну відповідність між джерелом і копією.

Алгоритм роботи rsync базується на порівнянні файлів за часовими мітками та розмірами, у результаті чого визначається перелік об’єктів, які потребують оновлення, після чого виконується передача лише змінених даних, що формує ефективний механізм дедуплікації на рівні передачі, при цьому у разі відсутності змін відповідні файли пропускаються без додаткових операцій, що дозволяє уникнути зайвого використання ресурсів, а сама логіка процесу може бути представлена як послідовність дій, у якій виконується (див. рис. 2.14):

1. сканування директорій;
2. порівняння файлів;
3. визначення змін;
4. передача модифікованих даних;
5. оновлення резервної копії.

Кожен наступний етап виконується лише після завершення попереднього.

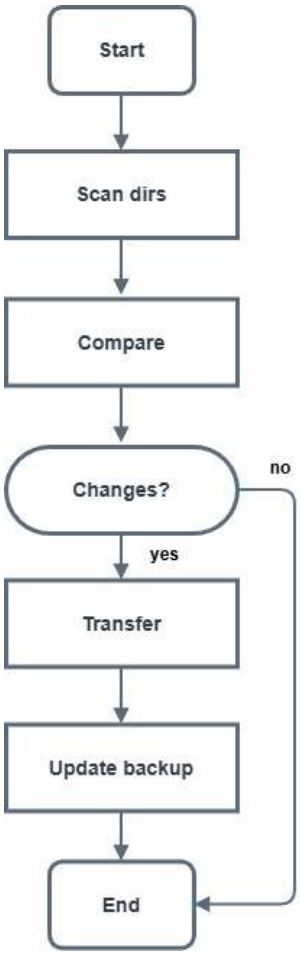


Рисунок 2.14 – Алгоритм локального резервного копіювання

У випадку наявності змін алгоритм переходить до передачі відповідних файлів із подальшим оновленням локального дзеркала, а у випадку їх відсутності процес завершується без виконання додаткових операцій, що формує оптимізований підхід до обробки даних, при якому ресурси використовуються лише за необхідності, а сформована таким чином локальна копія забезпечує наявність актуального резерву даних навіть у разі втрати мережевого з’єднання або недоступності хмарного середовища, що підвищує загальну відмовостійкість системи та забезпечує можливість швидкого відновлення інформації.

Автоматизація. Завершальним етапом реалізації алгоритмів системи реплікації є автоматизація процесу синхронізації, яка здійснюється за допомогою планувальника задач cron, що забезпечує виконання скриптів у задані часові інтервали без необхідності ручного втручання, при цьому налаштування

виконується через редагування конфігурації за допомогою команди "crontab -e", у межах якої додається відповідне правило запуску, яке визначає періодичність виконання скрипта, а саме щоденний запуск о 03:00, що відповідає періоду мінімального навантаження на систему та мережеву інфраструктуру, що дозволяє зменшити вплив процесів синхронізації на продуктивність серверного середовища та уникнути конфліктів із активною роботою користувачів.

Логіка функціонування автоматизованої системи базується на циклічному виконанні завдань, де планувальник cron ініціює запуск bash-скрипта у визначений момент часу, після чого відбувається послідовне виконання всіх етапів обробки даних, що включають [20]:

1. локальну синхронізацію за допомогою rsync;
2. передачу підготовленого резервного сховища у хмару через rclone;
3. фіксацію результатів у лог-файлі "/var/log/sync_service.log".

Після завершення виконання процес переходить у стан очікування наступного циклу, що формує безперервний механізм підтримки актуальності даних у системі (див. рис. 2.15).

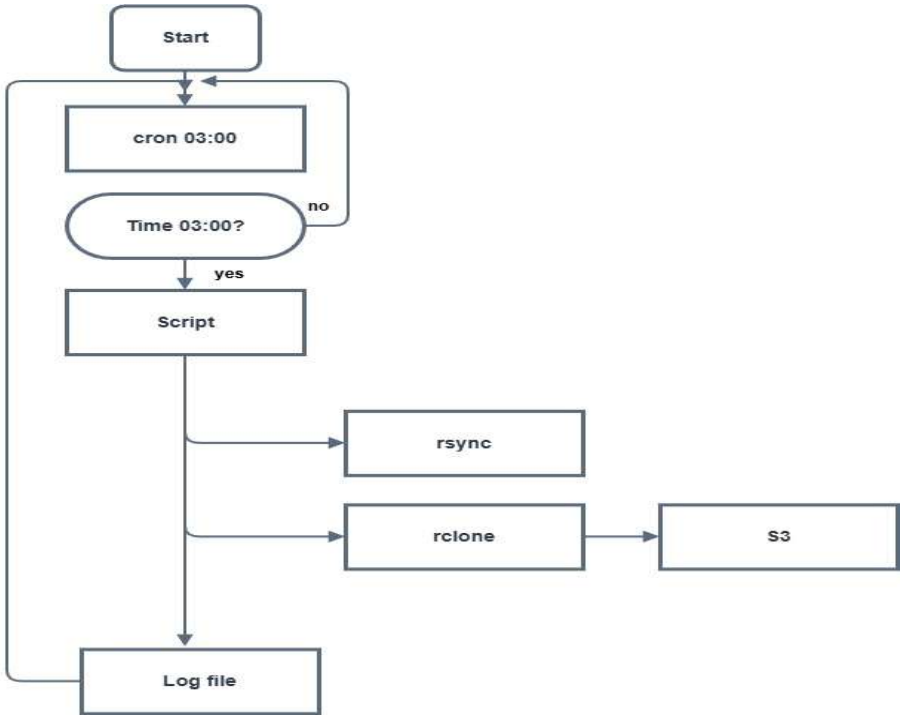


Рисунок 2.15 – Алгоритм асинхронної передачі даних у хмару

Такий підхід дозволяє забезпечити регулярне оновлення інформації без необхідності контролю з боку адміністратора, а також знижує ймовірність виникнення помилок, пов’язаних із людським фактором, при цьому кожен запуск супроводжується записом у журнал, що дозволяє відстежувати історію виконання, аналізувати тривалість операцій та виявляти можливі відхилення у роботі системи, що спрощує процес її обслуговування та діагностики.

У межах загальної архітектури автоматизація формує завершений цикл обробки даних, у якому [20-22]:

1. планувальник ініціює виконання;
2. скрипт реалізує логіку синхронізації;
3. утиліта rsync забезпечує локальне дублювання;
4. rclone виконує передачу у хмарне середовище;
5. система логування фіксує результати виконання.

Після чого відбувається перехід до наступного циклу, що забезпечує постійну актуальність даних одночасно у робочій директорії, локальному резервному сховищі та хмарному середовищі, що формує основу для відмовостійкої гібридної інфраструктури з можливістю швидкого відновлення інформації та ефективного використання ресурсів.

2.4 Тестування працездатності комплексу

Завершальним етапом практичної реалізації програмно-апаратного комплексу синхронізації даних у гібридному середовищі виступає перевірка його працездатності в умовах, максимально наближених до реальної експлуатації, при цьому тестування орієнтоване не лише на підтвердження коректності функціонування окремих компонентів, що забезпечують локальне резервне копіювання та взаємодію з хмарним сховищем, а й на комплексну оцінку стабільності алгоритмів синхронізації, цілісності даних та здатності системи

адекватно реагувати на зміни у файловій структурі, у межах такого підходу перевіряється поведінка системи при створенні, модифікації та видаленні файлів, а також фіксується швидкість обробки операцій і коректність відображення результатів у логах, що дозволяє оцінити ефективність реалізованих механізмів реплікації.

Організація тестового сценарію передбачає створення умов, у яких всі етапи синхронізації можуть бути відстежені у режимі реального часу, що досягається шляхом паралельного використання декількох термінальних сесій, де кожна з них відповідає за окрему частину процесу, а результати виконання операцій фіксуються у вигляді логів, що забезпечує наочність та контрольованість експерименту, у результаті формується цілісне уявлення про роботу системи та її поведінку у різних сценаріях навантаження.

Підготовка середовища для тесту. Підготовчий етап тестування передбачає формування контрольованого середовища, у межах якого можливо відтворити типові сценарії використання системи, при цьому основною особливістю обраної методології є застосування трьох незалежних термінальних сесій, кожна з яких виконує окрему функцію у процесі тестування, що дозволяє одночасно спостерігати за перебігом операцій, ініціювати зміни у файловій системі та запускати механізми синхронізації (див. рис. 2.16).

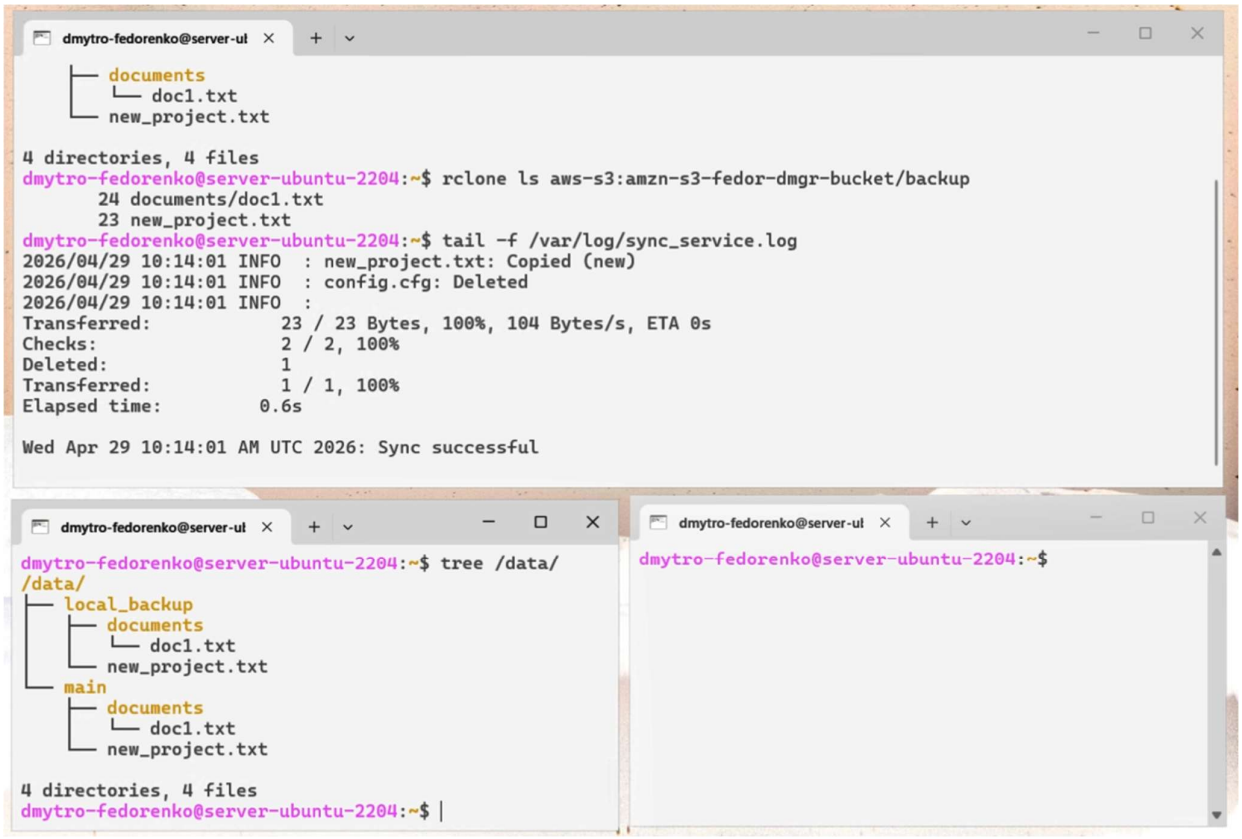


Рисунок 2.16 – Використання трьох терміналів для тестування

Перший термінал використовується для моніторингу логів системи, де за допомогою команди відстеження у реальному часі відображається інформація про виконання операцій синхронізації, включаючи повідомлення про успішне завершення, помилки та часові характеристики процесів, що дозволяє оперативно реагувати на будь-які відхилення у роботі системи та забезпечує прозорість виконання алгоритмів.

Другий термінал виконує функцію середовища генерації змін, у якому здійснюється створення нових файлів, модифікація існуючих або їх видалення у робочій директорії, що імітує реальну діяльність користувача або прикладної системи, у результаті чого формується набір змін, які повинні бути оброблені системою синхронізації, а така організація дозволяє перевірити здатність системи коректно визначати зміни та обробляти їх відповідно до заданих алгоритмів.

Третій термінал призначений для безпосереднього запуску скрипта синхронізації, що ініціює виконання всіх етапів обробки даних, включаючи

локальне дублювання та передачу у хмарне середовище, у результаті чого формується повний цикл обробки інформації, який може бути проаналізований у режимі реального часу через логування та зміну стану файлових директорій.

Перед початком тестування виконується перевірка коректності налаштування директорій "/data/main" та "/data/local_backup", доступності мережевого з’єднання, а також наявності доступу до хмарного середовища, що дозволяє забезпечити стабільність проведення експерименту, додатково здійснюється очищення попередніх тестових даних, що гарантує чистоту результатів і виключає вплив залишкової інформації на результати перевірки, а сформована таким чином структура тестування дозволяє забезпечити повний контроль над процесом виконання операцій і створює умови для об’єктивної оцінки працездатності системи.

Імітація роботи. Основна частина тестування полягає у відтворенні типових сценаріїв функціонування системи, що можуть виникати під час реальної експлуатації, при цьому імітація включає:

— створення нових файлів;

— модифікацію існуючих об’єктів;

— переміщення між директоріями;

— видалення застарілих даних.

Все це дозволяє перевірити поведінку системи у різних умовах зміни файлової структури, у другому терміналі виконуються операції над файлами, зокрема (див. рис. 2.17):

1. створюються нові документи;
2. редагуються наявні файли зі зміною їхнього вмісту та часових міток;
3. здійснюється перейменування або переміщення об’єктів між підкаталогами.

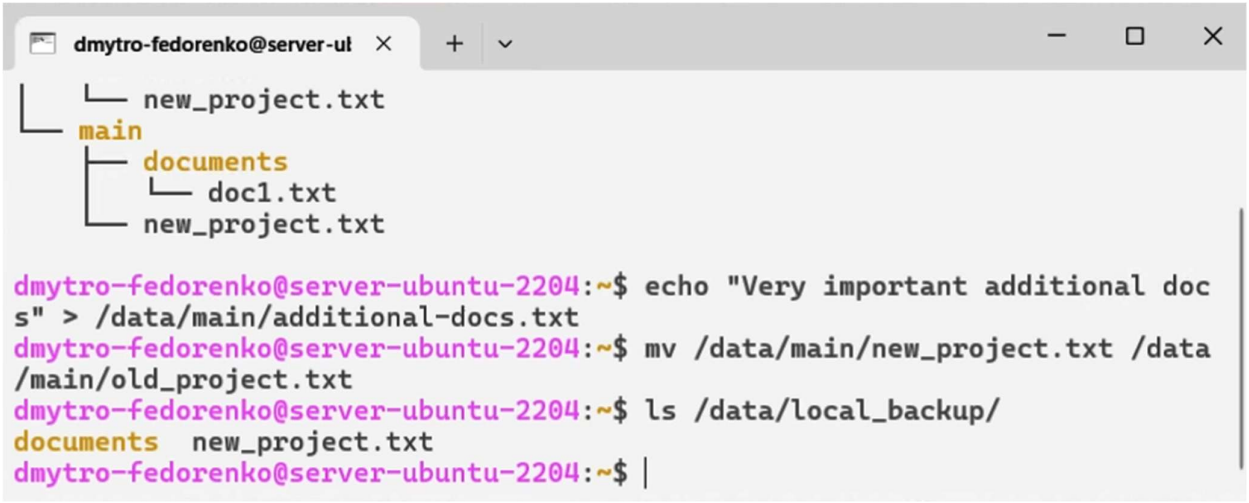


Рисунок 2.17 – Імітація змін у файловій системі

Цим моделюються реальні дії користувача або прикладної системи, додатково виконується видалення файлів для перевірки коректності обробки параметра "delete" у процесі локальної синхронізації, що забезпечує узгодженість структури резервного сховища з основною директорією.

Паралельно у першому терміналі здійснюється моніторинг логів за допомогою команди безперервного відстеження, що дозволяє у режимі реального часу фіксувати всі події, пов’язані з виконанням скрипта, включаючи етапи (див. рис. 2.18):

1. обробки даних;
2. передачу файлів;
3. можливі помилки.
4. статус завершення операцій.

При цьому наявність детального журналювання забезпечує прозорість виконання процесів та створює основу для подальшого аналізу ефективності роботи системи.

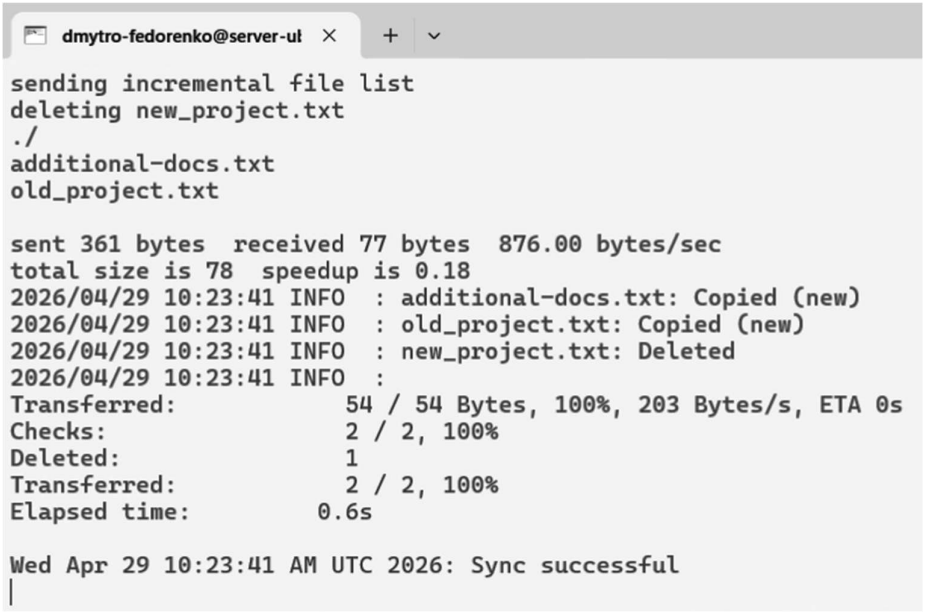


Рисунок 2.18 – Моніторинг логів у режимі реального часу

У третьому терміналі виконується безпосередній запуск bash-скрипта синхронізації, після чого відбувається послідовне виконання всіх передбачених алгоритмом етапів, що включають:

1. локальне дублювання даних із використанням механізму інкрементальної передачі;
2. подальшу відправку підготовленого резервного набору у хмарне середовище.

При цьому всі зміни, внесені у робочу директорію, повинні коректно відображатися як у локальному резервному сховищі, так і у віддаленому хмарному бакеті, що підтверджує узгодженість процесів синхронізації.

Логіка виконання такого сценарію може бути інтерпретована як послідовність дій, у якій спочатку здійснюється створення або модифікація файлів, після чого ініціюється запуск скрипта, далі виконується оновлення локального резервного сховища із передачею лише змінених даних, після чого відбувається передача підготовлених об’єктів у хмару, а завершальним етапом виступає фіксація результатів у журналі, що дозволяє оцінити коректність виконання кожного кроку, у такому підході забезпечується можливість перевірки системи у різних сценаріях навантаження та оцінки її реакції на зміни вхідних даних.

Валідація цілісності. Заключний етап тестування спрямований на перевірку цілісності даних та узгодженості їхнього стану між усіма рівнями зберігання, при цьому основним завданням виступає підтвердження того, що після виконання скрипта структура та вміст файлів у локальному резервному сховищі та хмарному середовищі повністю відповідають робочій директорії, перевірка починається з аналізу локального резервного сховища, де здійснюється порівняння списків файлів або візуальний огляд структури директорій, що дозволяє переконатися у наявності всіх змін, внесених у вихідні дані.

Наступним кроком виконується перевірка хмарного середовища за допомогою спеціалізованих команд, що дозволяють отримати перелік об’єктів у віддаленому сховищі, після чого результати порівнюються з локальним резервним набором даних, що забезпечує підтвердження ідентичності структури та вмісту файлів, додатково може використовуватися веб-інтерфейс хмарного сервісу для візуальної перевірки наявності об’єктів і коректності їхнього розміщення (див. табл.2.3).

Таблиця 2.3 – Результати тестування працездатності системи

Тип операції	Час виконання	Результат синхронізації	Статус
Створення файлу	~1–2 с	Дані передано	Успішно
Зміна файлу	~1–3 с	Оновлення виконано	Успішно
Видалення файлу	~1 с	Видалено в усіх середовищах	Успішно
Масова передача	~5–10 с	Часткова синхронізація	Успішно
Помилка доступу	—	Відмова у доступі	Помилка

Важливим аспектом валідації виступає аналіз журналів виконання, у яких фіксується час виконання операцій, обсяг переданих даних, а також статус завершення процесу, наявність записів про успішне виконання без помилок свідчить про коректність роботи алгоритмів синхронізації, при цьому відсутність критичних збоїв підтверджує стабільність функціонування системи (див. рис. 2.19).

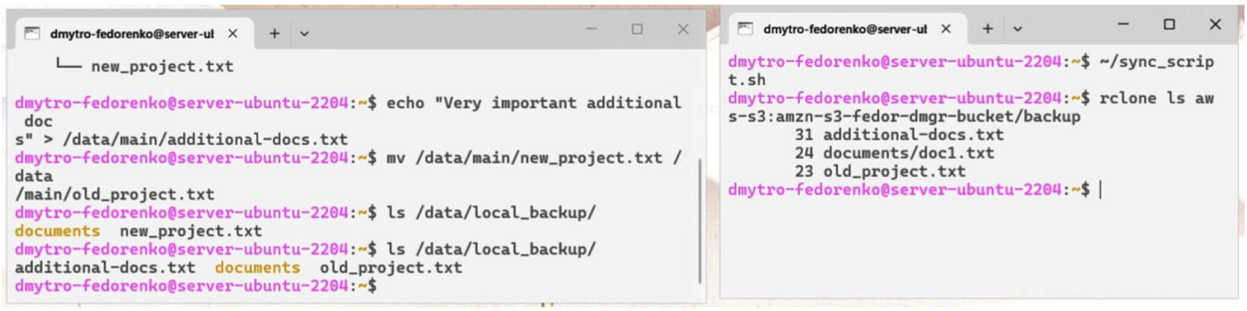


Рисунок 2.19 – Перевірка цілісності даних між середовищами

Додатково доцільним є використання механізмів перевірки контрольних сум файлів, що дозволяє отримати більш точне підтвердження їхньої ідентичності навіть у випадках, коли зміни не відображаються у часових мітках або мають нетиповий характер, у такому підході здійснюється порівняння значень хеш-функцій для відповідних файлів у різних середовищах зберігання, що забезпечує високий рівень достовірності перевірки.

Узагальнено процес валідації можна представити як послідовний перехід від робочої директорії до локального резервного сховища та далі до хмарного бакета із подальшим порівнянням результатів та підтвердженням їхньої узгодженості, що дозволяє оцінити повноту та коректність реплікації даних, за результатами проведеного тестування підтверджується працездатність розробленого комплексу, коректність реалізації алгоритмів синхронізації та здатність системи забезпечувати узгоджене зберігання даних у гібридній інфраструктурі, отримані результати демонструють можливість ефективного використання запропонованого рішення у практичних сценаріях із забезпеченням надійності, відмовостійкості та актуальності даних у всіх середовищах зберігання.

3. ОПТИМІЗАЦІЯ ТА ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ГІБРИДНОГО РІШЕННЯ

3.1 Методи оптимізації процесу синхронізації

Побудова гібридної системи синхронізації даних передбачає не лише реалізацію базових механізмів обміну інформацією між локальним середовищем і хмарною інфраструктурою, але й впровадження додаткових підходів, спрямованих на підвищення ефективності, стабільності та економічності функціонування всієї системи, при цьому особлива увага приділяється коректності конфігурації програмних інструментів, організації журналювання подій та мінімізації навантаження на мережеві ресурси, оскільки навіть незначні помилки на рівні налаштувань можуть призводити до порушень процесу синхронізації, тоді як оптимізовані механізми обробки змін та контроль виконання операцій дозволяють суттєво підвищити продуктивність і надійність запропонованого рішення.

Вирішення проблем конфігурації. Одним із найбільш чутливих аспектів функціонування системи виступає процес налаштування інструментів взаємодії з хмарним середовищем, зокрема утиліти rclone, конфігурація якої виконується через інтерактивний інтерфейс командного рядка, де кожен параметр має критичне значення для коректної роботи системи, при цьому вибір неправильних значень може призвести до помилок автентифікації, відсутності доступу до хмарного бакета або повної неможливості виконання операцій синхронізації, практичний досвід демонструє, що однією з найпоширеніших причин виникнення помилок є використання текстових значень замість числових ідентифікаторів у меню конфігурації, оскільки під час вибору типу сховища, регіону або інших параметрів система очікує введення конкретного номера, що відповідає певному пункту списку, відхилення від цього формату призводить до формування некоректного конфігураційного файлу та подальших збоїв у роботі (див. рис. 3.1).

Додатковим фактором стабілізації виступає використання параметра "env_auth", який дозволяє застосовувати облікові дані, попередньо налаштовані через AWS CLI, що усуває необхідність дублювання ключів доступу у конфігураційних файлах та знижує ризик виникнення помилок автентифікації, у тому числі типу "403 Access Denied", пов’язаних із некоректними або застарілими ключами, при цьому використання єдиного джерела автентифікації забезпечує узгодженість параметрів доступу та спрощує процес адміністрування системи.

Загальна логіка перевірки та оптимізації конфігурації може бути інтерпретована як послідовний процес, у якому спочатку формується конфігурація інструменту, після чого виконується перевірка введених параметрів, у разі виявлення помилок здійснюється їх корекція з подальшою повторною перевіркою, а після усунення неточностей проводиться тестове підключення до хмарного сховища, успішне завершення якого підтверджує коректність налаштувань і готовність системи до виконання операцій синхронізації, у такому підході досягається підвищення стабільності роботи всієї інфраструктури та зменшення ймовірності виникнення критичних збоїв під час експлуатації (див. рис. 3.1).

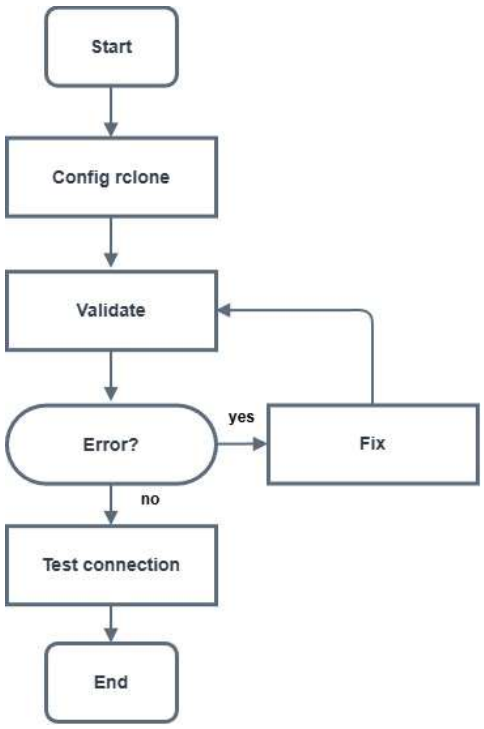


Рисунок 3.1 – Метод налаштування конфігурації rclone

Оптимізація системи логування (Logrotate). У процесі виконання скрипта синхронізації формується значна кількість журналів, що містять інформацію про:

— перебіг операцій;

— передачу файлів;

— можливі помилки;

— загальний статус виконання завдань.

Постійне накопичення таких логів призводить до поступового заповнення дискового простору, що може негативно впливати на продуктивність та стабільність роботи серверного середовища, у зв’язку з цим доцільним є впровадження механізмів автоматизованого керування журналами, реалізованих за допомогою утиліти "logrotate", яка інтегрована в операційну систему та дозволяє регламентувати життєвий цикл лог-файлів [23].

Налаштування виконується через конфігураційний файл, розміщений у системному каталозі, де визначаються правила обробки журналів, зокрема:

— періодичність їх ротації;

— кількість збережених архівів;

— параметри стиснення;

— умови обробки у випадку відсутності або порожнього вмісту файлів.

У типовій конфігурації передбачається щоденна ротація логів із обмеженням кількості архівів до семи, що відповідає збереженню історії за один тиждень, після чого старі файли автоматично видаляються, а попередні версії журналів стискаються для економії дискового простору, такий підхід дозволяє підтримувати баланс між доступністю історичних даних та ефективністю використання ресурсів (див. рис. 3.2) [23].

```
dmytro-fedorenko@server-ubuntu-2204:~$ cat /var/log/sync_service.log
Wed Apr 29 09:54:29 AM UTC 2026: Sync started
sending incremental file list

sent 76 bytes  received 12 bytes  176.00 bytes/sec
total size is 0  speedup is 0.00
2026/04/29 09:54:29 INFO  : There was nothing to transfer
2026/04/29 09:54:29 INFO  :
Transferred:              0 / 0 Bytes, -, 0 Bytes/s, ETA -
Elapsed time:         0.4s

Wed Apr 29 09:54:29 AM UTC 2026: Sync successful
dmytro-fedorenko@server-ubuntu-2204:~$
```

Рисунок 3.2 – Сформований конфіг-файл для ротації логів (Logrotate)

Алгоритм роботи механізму ротації може бути інтерпретований як послідовність етапів, у якій спочатку відбувається накопичення логів у процесі виконання системних операцій, після досягнення визначеного часового інтервалу здійснюється їх архівування, далі виконується стиснення збережених файлів та видалення застарілих записів відповідно до встановлених обмежень, у результаті чого забезпечується контрольований обсяг журналів без втрати актуальної інформації (див. рис. 3.3).

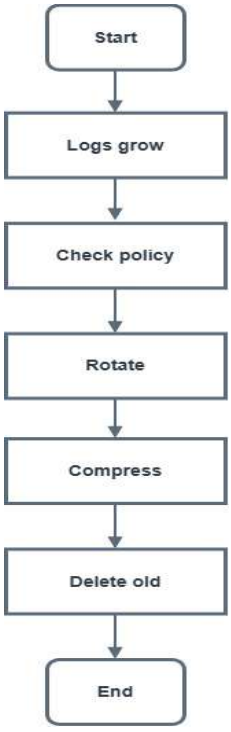


Рисунок 3.3 – Метод ротації логів (Logrotate)

Структуроване логування у поєднанні з автоматизованою ротацією спрощує аналіз роботи системи та дозволяє оперативно виявляти потенційні проблеми, при цьому збереження лише релевантної історії подій зменшує навантаження на файлову систему та запобігає деградації продуктивності, у такому підході досягається оптимізація використання дискового простору без втрати можливості аудиту виконаних операцій.

Зменшення навантаження на мережу. Одним із важливих напрямів оптимізації виступає зменшення обсягу передаваних даних між локальним середовищем та хмарною інфраструктурою, оскільки передача повних копій файлів під час кожного циклу синхронізації є неефективною та створює значне навантаження на мережеві ресурси, у зв’язку з цим застосовується інкрементальний підхід, при якому передаються лише змінені частини файлів або нові об’єкти, такий механізм базується на порівнянні розмірів, часових міток та контрольних сум, що дозволяє точно визначати обсяг необхідних змін і виключати передачу незмінних даних [24].

Алгоритм інкрементальної синхронізації передбачає попередній аналіз стану файлів у вихідному та цільовому сховищах, після чого виконується ідентифікація змін, далі передаються лише ті об’єкти, що були модифіковані або відсутні у цільовому середовищі, а завершальним етапом виступає оновлення структури сховища, що забезпечує узгодженість даних без зайвих витрат ресурсів, у такому підході суттєво зменшується мережевий трафік та скорочується час виконання операцій синхронізації (див. рис. 3.4).

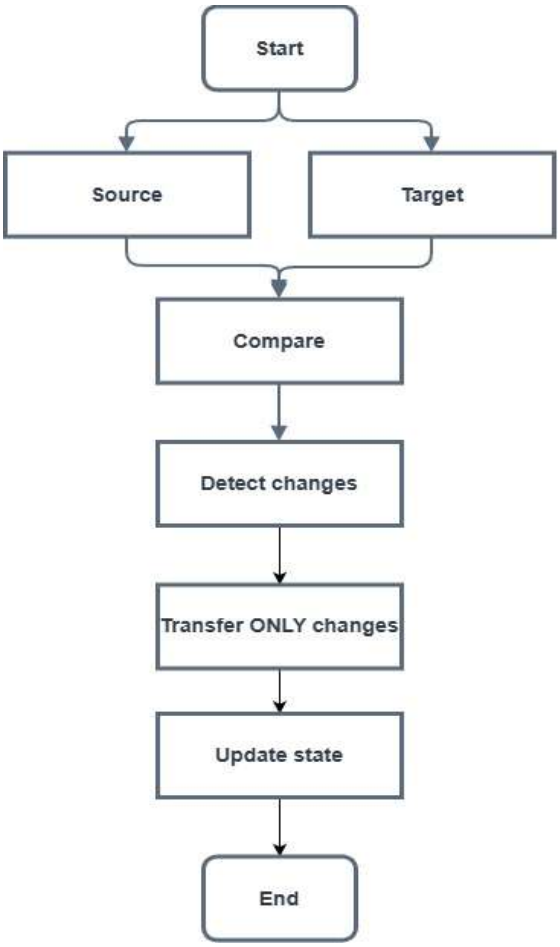


Рисунок 3.4 – Метод зменшення мережевого навантаження

Додаткові методи оптимізації включають використання стиснення даних під час передачі, обмеження швидкості передачі для запобігання перевантаженню каналу зв’язку, а також виконання синхронізації у періоди мінімальної активності, що дозволяє знизити вплив на інші мережеві сервіси, зокрема запуск процесів у нічний час забезпечує більш стабільну роботу системи за рахунок відсутності пікових навантажень.

Результатом впровадження таких підходів виступає суттєве зменшення обсягу переданих даних, підвищення швидкості синхронізації та оптимізація використання мережевих ресурсів, що у поєднанні з коректною конфігурацією інструментів і ефективним управлінням журналами формує збалансовану та продуктивну гібридну інфраструктуру з високим рівнем надійності та економічної доцільності.

3.2 Порівняльний аналіз із комерційними аналогами

Оцінювання ефективності розробленого програмно-апаратного комплексу синхронізації даних у гібридній структурі передбачає його зіставлення з існуючими комерційними рішеннями, що широко застосовуються у сучасних IT-інфраструктурах, до яких належать повністю хмарні сервіси типу SaaS, що надають готові інструменти для зберігання, синхронізації та обміну файлами без необхідності розгортання локальних компонентів, такі рішення характеризуються простотою використання та швидкістю впровадження, проте супроводжуються обмеженнями, пов’язаними із залежністю від постачальника послуг, обмеженою гнучкістю налаштувань та відсутністю повного контролю над даними, у такому контексті гібридна модель, що поєднує локальну інфраструктуру з хмарним середовищем, дозволяє досягти балансу між доступністю, надійністю та керованістю системи, забезпечуючи можливість адаптації архітектури до конкретних вимог та умов експлуатації.

Переваги гібридної структури. Основною відмінністю гібридної архітектури від повністю хмарних рішень виступає розподіл даних між кількома незалежними середовищами зберігання, при якому інформація одночасно присутня у:

— робочій директорії;

— локальному резервному сховищі;

— хмарному бакеті.

Така організація забезпечує підвищений рівень відмовостійкості та дозволяє уникнути втрати даних навіть у випадку виходу з ладу одного з компонентів системи, у повністю хмарних SaaS-рішеннях зберігання централізоване, що створює залежність від доступності конкретного сервісу, у разі технічних збоїв або обмежень доступу користувач втрачає можливість працювати з інформацією, тоді як у гібридній моделі локальна копія залишається доступною незалежно від стану хмарної інфраструктури, що підвищує надійність зберігання [25].

Додатковою перевагою виступає підвищена швидкість доступу до даних, оскільки операції читання та запису виконуються безпосередньо у локальному середовищі без залучення мережевих ресурсів, а хмарне сховище використовується як додатковий рівень резервування та масштабування, що дозволяє оптимізувати витрати та забезпечити гнучкість управління даними, у такій архітектурі можлива реалізація політик резервного копіювання, адаптованих до конкретних вимог, зокрема стратегій багаторівневого зберігання, що можуть бути реалізовані без додаткових витрат на сторонні сервіси, тоді як у межах комерційних платформ подібні механізми часто потребують додаткових фінансових вкладень (див. табл. 3.1).

Таблиця 3.1 – Порівняння гібридної системи з SaaS-рішеннями

Критерій	SaaS-рішення	Гібридна система
Контроль над даними	Обмежений	**Повний / частковий**
Залежність від вендора	Висока	**Низька**
Гнучкість	Обмежена	**Висока**
Вартість	Підписка	**Гнучка (pay-as-you-go)**
Продуктивність	Залежить від мережі	**Висока (локальна обробка)**
Масштабованість	Висока	**Висока**

Логіка забезпечення відмовостійкості у гібридній системі може бути інтерпретована як умовна послідовність рішень, у якій у разі збою локального середовища використовується хмарна копія, при недоступності хмарного сховища застосовується локальний резерв, а за відсутності збоїв система функціонує у штатному режимі, що забезпечує безперервність доступу до даних та підвищує стійкість до відмов, у результаті така структура забезпечує більш високий рівень надійності, гнучкості та продуктивності у порівнянні з централізованими хмарними рішеннями, формуючи основу для ефективного використання у різних сценаріях експлуатації.

Незалежність від вендора. Ще одним суттєвим аспектом функціонування гібридної системи синхронізації даних є питання залежності від постачальника хмарних послуг, яке традиційно виникає при використанні пропрієтарних SaaS-рішень, де інфраструктура, API та механізми доступу жорстко прив’язані до

конкретної платформи, що ускладнює міграцію, масштабування та інтеграцію з іншими сервісами, а також створює додаткові витрати у разі зміни постачальника або переходу на альтернативні технології, у той час як застосування інструментів з відкритим кодом, таких як rclone та rsync, формує принципово інший підхід до організації взаємодії з хмарною інфраструктурою, який базується на універсальних інтерфейсах та стандартизованих механізмах доступу, що дозволяє абстрагуватися від конкретного провайдера і забезпечити гнучкість у виборі середовища зберігання.

Особливість використання rclone полягає у підтримці широкого спектра хмарних сервісів, включаючи:

— Amazon Web Services;
— Google Cloud Platform;
— Microsoft Azure.

Все це дозволяє здійснювати підключення до різних провайдерів через єдиний інтерфейс конфігурації, при цьому логіка синхронізації, сценарії автоматизації та структура даних залишаються незмінними, а зміна хмарного середовища зводиться до редагування параметрів доступу у конфігураційному файлі, що суттєво спрощує процес переходу між платформами та мінімізує часові і ресурсні витрати, пов’язані з перебудовою системи [26].

Практична реалізація незалежності від вендора проявляється у тому, що конфігурація віддаленого сховища у rclone містить лише параметри підключення до конкретного сервісу, такі як:

— "endpoint";
— ключі доступу;
— назва бакета.

Зміна цих параметрів дозволяє перенаправити потік даних до іншого провайдера без внесення змін у bash-скрипти або алгоритми обробки даних, при цьому використання змінних середовища або централізованої конфігурації забезпечує додаткову гнучкість у керуванні доступом і дозволяє уникнути дублювання облікових даних у різних частинах системи.

Алгоритм зміни хмарного провайдера може бути представлений як послідовність дій, що починається з оновлення конфігурації віддаленого сховища, після чого виконується тестове підключення до нового ресурсу, перевірка доступності бакета та коректності автентифікації, далі у разі успішного результату відбувається перенаправлення процесу синхронізації на новий "endpoint", а у випадку виникнення помилок здійснюється корекція параметрів конфігурації до досягнення стабільного з'єднання, що дозволяє виконати міграцію без простоїв системи та втрати даних, причому вся логіка обробки залишається ідентичною незалежно від обраного провайдера (див. рис. 3.5).

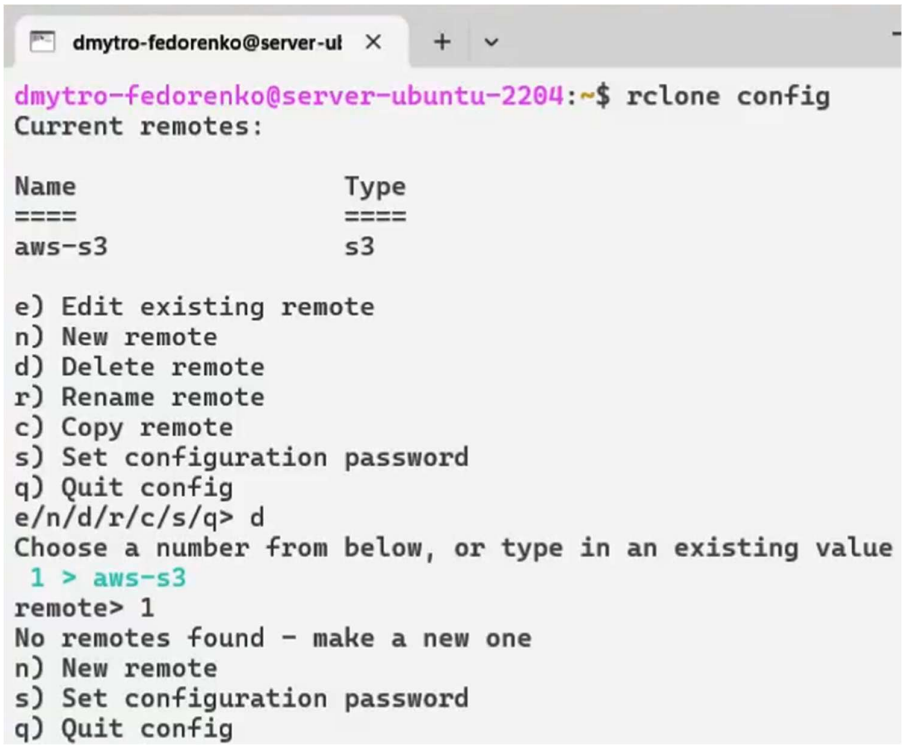


Рисунок 3.5 – Процес зміни хмарного провайдера

На відміну від цього, у традиційних SaaS-рішеннях інтеграція з хмарною платформою часто базується на власних API та інструментах, які не мають універсальної сумісності, що ускладнює перенесення даних і вимагає повної перебудови архітектури у разі зміни сервісу, включаючи:

— модифікацію програмного коду;

— адаптацію політик безпеки;

— повторне налаштування механізмів доступу.

Все це значно збільшує складність управління інфраструктурою та створює залежність від постачальника.

Додатковою перевагою запропонованого підходу є можливість реалізації мультихмарної моделі, при якій одночасно використовуються декілька провайдерів, наприклад Amazon Web Services та Google Cloud Platform, що дозволяє [26]:

1. розподіляти навантаження між різними середовищами;
2. підвищувати рівень відмовостійкості;
3. оптимізувати витрати залежно від тарифної політики кожного сервісу.

При цьому використання єдиного інструменту управління забезпечує централізований контроль над процесами синхронізації (див. рис. 3.6).

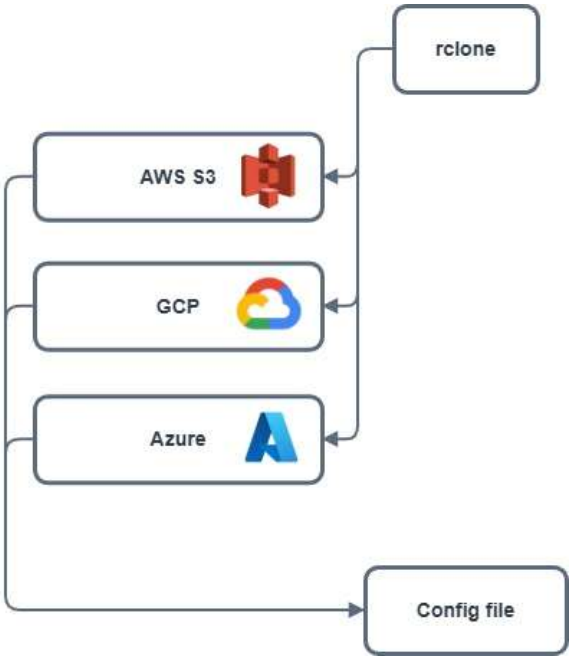


Рисунок 3.6 – Мультихмарна архітектура на базі rclone

Узагальнення такої архітектури демонструє, що локальне резервне сховище виступає базовим джерелом даних, яке передає інформацію до шару абстракції у вигляді rclone, після чого дані можуть синхронізуватися з будь-яким сумісним

хмарним середовищем, що формує універсальну модель взаємодії, у якій відсутня жорстка прив’язка до конкретного постачальника, а зміна інфраструктури не потребує суттєвих змін у загальній логіці системи.

Такий підхід забезпечує повний контроль над процесами синхронізації, спрощує масштабування та модернізацію системи, дозволяє адаптувати інфраструктуру до змінних умов експлуатації та уникати обмежень, характерних для пропрієтарних платформ, що у підсумку формує гнучку, незалежну та технологічно стійку архітектуру зберігання та обробки даних у гібридному середовищі.

3.3 Оцінка вартості та перспективи розвитку

Формування економічної доцільності програмно-апаратного комплексу синхронізації даних у гібридній інфраструктурі безпосередньо пов’язане з аналізом початкових витрат на розгортання, експлуатаційних витрат у процесі функціонування та можливостей їх подальшої оптимізації, причому на етапі розробки і тестування система орієнтується на використання безкоштовних або умовно безкоштовних ресурсів, що суттєво знижує поріг входження і дозволяє реалізувати повноцінне рішення без значних фінансових вкладень, оскільки локальне середовище розгортається на базі віртуалізації із застосуванням VirtualBox, що усуває необхідність придбання окремого серверного обладнання і дозволяє використовувати наявні обчислювальні ресурси персонального комп’ютера, при цьому віртуалізація забезпечує ефективний розподіл ресурсів хост-системи та створює можливість моделювання різних сценаріїв навантаження без додаткових витрат, а основна частина витрат у довгостроковій перспективі пов’язується саме з використанням хмарних сервісів, зокрема об’єктного сховища.

Використання Free Tier та вартість зберігання S3. На початковому етапі доцільним підходом є використання безкоштовного тарифного плану AWS Free

Tier, який надається платформою Amazon Web Services і передбачає обмежений обсяг безкоштовного використання ресурсів, зокрема до 5 ГБ зберігання у Amazon S3 стандартного класу, а також певні ліміти на кількість операцій читання і запису, що дозволяє повністю протестувати систему синхронізації, перевірити її працездатність, оцінити швидкодію та надійність без залучення додаткового фінансування, причому у багатьох випадках додатково можуть надаватися стартові кредити, наприклад у розмірі близько 100 доларів, які забезпечують можливість розширеного тестування інфраструктури протягом обмеженого періоду часу, що формує практично нульову вартість входу у проєкт [27].

Подальша оцінка вартості визначається вибором класу зберігання у Amazon S3, оскільки тарифікація залежить від:

— типу доступу до даних;

— обсягу та частоти використання.

При цьому стандартний клас S3 орієнтований на частий доступ і має орієнтовну вартість близько 0.023 долара за гігабайт на місяць, у той час як класи з рідшим доступом, такі як Standard-IA або One Zone-IA, пропонують нижчу вартість зберігання за рахунок збільшення вартості доступу до даних або зниження рівня відмовостійкості, а архівні класи, включаючи:

— Glacier Instant Retrieval;

— Glacier Flexible Retrieval;

— Glacier Deep Archive.

Вони ж дозволяють суттєво зменшити витрати на довготривале зберігання, де вартість може знижуватися до кількох тисячних долара за гігабайт, що робить їх доцільними для резервних копій, які рідко використовуються (див. рис. 3.7) [27-28].

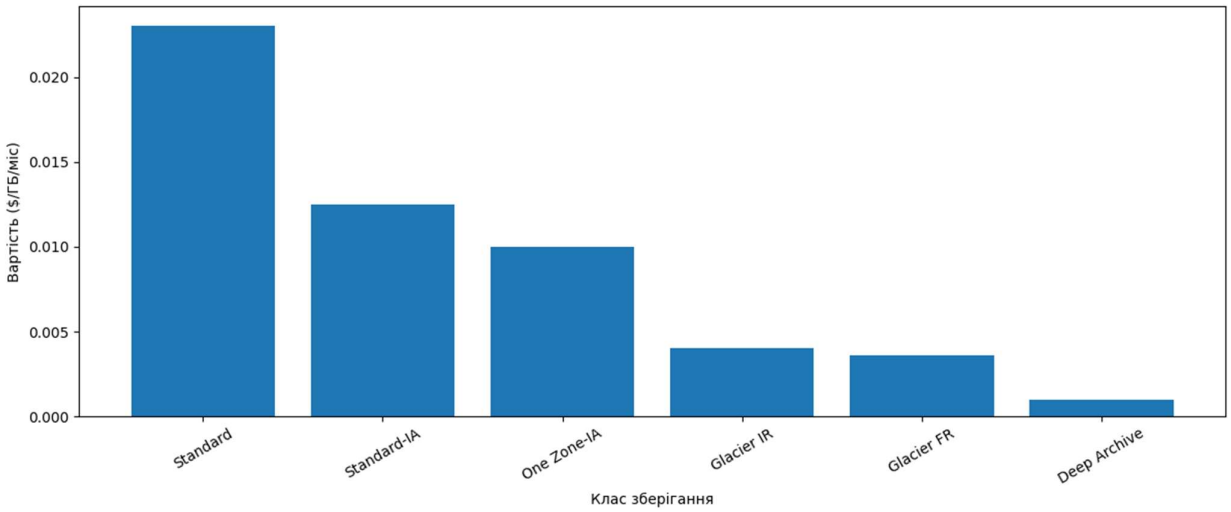


Рисунок 3.7 – Структура витрат гібридної інфраструктури

Загальна вартість експлуатації також залежить від обсягу переданих даних та кількості операцій, оскільки у хмарних сервісах оплата часто включає не лише зберігання, але й трафік, а використання інкрементних алгоритмів синхронізації за допомогою rsync та rclone дозволяє передавати лише змінені частини файлів, що суттєво зменшує навантаження на мережу і безпосередньо впливає на зниження витрат, при цьому відсутність необхідності у використанні комерційних SaaS-платформ із щомісячною підпискою додатково зменшує фінансове навантаження, оскільки використовується програмне забезпечення з відкритим кодом без ліцензійних платежів (див. табл. 3.2).

Таблиця 3.2 – Порівняння класів зберігання Amazon S3 та їх вартості

Клас зберігання	Вартість ($/ГБ/міс)	Час доступу	Сценарій використання
Standard	~0.023	Миттєвий	Частий доступ
Standard-IA	~0.0125	Миттєвий	Рідкий доступ
One Zone-IA	~0.01	Миттєвий	Некритичні дані
Glacier Instant Retrieval	~0.004	Мілісекунди	Архів з швидким доступом
Glacier Flexible Retrieval	~0.0036	Хвилини	Резервні копії
Glacier Deep Archive	~0.00099	Години	Довготривале архівування

Узагальнення економічної моделі демонструє, що початковий етап впровадження характеризується мінімальними або нульовими витратами за рахунок використання локальних ресурсів та безкоштовних тарифів, а подальші витрати мають гнучкий характер і можуть масштабуватися відповідно до реальних потреб системи, що дозволяє адаптувати інфраструктуру як для невеликих проєктів, так і для корпоративного використання без різкого зростання бюджету.

Стратегія здешевлення зберігання (Storage Classes). Подальша експлуатація системи в умовах реального використання супроводжується поступовим зростанням обсягів даних, що безпосередньо призводить до збільшення витрат на їх зберігання, у зв'язку з чим виникає необхідність впровадження механізмів оптимізації вартості, серед яких найбільш ефективним підходом є використання різних класів зберігання в межах Amazon S3, де кожен клас орієнтований на певний сценарій доступу до даних і характеризується власним співвідношенням вартості та швидкості отримання інформації, при цьому стандартний клас забезпечує швидкий і постійний доступ до об'єктів, тоді як архівні класи, зокрема Amazon S3 Glacier, призначені для довготривалого зберігання із суттєво нижчою вартістю, але з більшим часом відновлення.

Раціональне використання цих класів дозволяє суттєво знизити витрати без зміни функціональності системи, оскільки основна ідея полягає у розподілі даних відповідно до частоти їх використання, де актуальна інформація зберігається у швидкодоступному середовищі, а застарілі або рідко використовувані дані автоматично переміщуються у дешевші класи, що забезпечує оптимальний баланс між економічною ефективністю та доступністю, причому реалізація такого підходу здійснюється за допомогою політик життєвого циклу, які налаштовуються безпосередньо у Amazon S3 і дозволяють автоматизувати процес переходу об'єктів між різними рівнями зберігання.

У межах типової стратегії життєвого циклу нові дані спочатку розміщуються у стандартному класі, де вони залишаються протягом визначеного періоду, наприклад 30 днів, після чого автоматично переміщуються до класу з рідшим доступом, такого як Standard-IA, де можуть зберігатися ще протягом 60–90 днів, а

надалі переходять до архівного рівня, зокрема Amazon S3 Glacier, де забезпечується мінімальна вартість зберігання при збереженні можливості відновлення, хоча доступ до таких даних потребує значно більше часу, що вимірюється хвилинами або годинами залежно від обраного підкласу архівування, при цьому для ще довшого зберігання може використовуватися глибокий архів, який мінімізує витрати, але збільшує затримку доступу (див. рис. 3.8) [29].

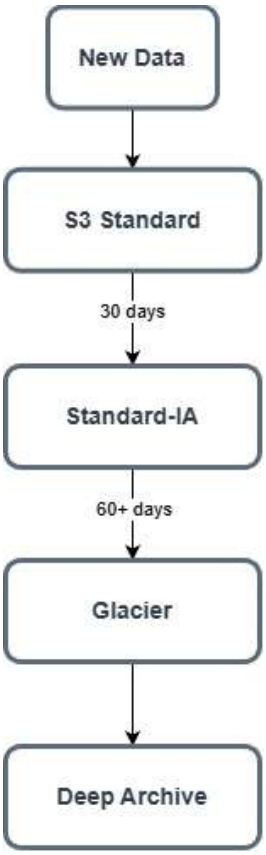


Рисунок 3.8 – Перехід даних між класами зберігання

Такий підхід дозволяє автоматизувати управління великими обсягами інформації без постійного втручання адміністратора, оскільки переміщення між класами виконується згідно з наперед визначеними правилами, а важливим аспектом залишається врахування компромісу між вартістю та швидкістю доступу, де архівні класи є доцільними для резервних копій або історичних даних, але не підходять для критичних систем, що потребують миттєвого доступу до інформації,

у зв’язку з чим правильна класифікація даних на етапі проєктування системи набуває особливого значення.

Додатковим напрямом оптимізації є зменшення обсягу збережених даних шляхом використання компресії перед передачею у хмару, що може бути реалізовано як на рівні файлової системи, так і засобами інструментів синхронізації, таких як rclone, що дозволяє знизити фактичний обсяг переданої інформації і відповідно витрати на її зберігання, при цьому інкрементні алгоритми передачі забезпечують відправлення лише змінених частин файлів, що додатково зменшує навантаження на інфраструктуру.

У перспективі розвиток системи може передбачати інтеграцію механізмів аналізу доступу до даних, які дозволяють визначати доцільність переміщення об’єктів між класами на основі статистики використання, що відкриває можливість динамічного керування життєвим циклом інформації без жорстко заданих часових інтервалів, а також впровадження мультихмарних стратегій зберігання, де різні провайдери використовуються для різних типів даних, що додатково підвищує гнучкість і надійність інфраструктури.

Узагальнення такого підходу демонструє, що впровадження багаторівневої моделі зберігання та політик життєвого циклу забезпечує ефективний баланс між вартістю, продуктивністю та доступністю даних, що створює необхідні умови для масштабування гібридної інфраструктури та її адаптації до змінних вимог у процесі експлуатації.

ВИСНОВКИ

У процесі практичної реалізації сформовано архітектуру комплексу синхронізації даних, що поєднує локальний сервер і хмарне об’єктне сховище. Визначено апаратну топологію та схему взаємодії між клієнтом, сервером і віддаленим бакетом. Обґрунтовано вибір програмного стека, який забезпечує сумісність, масштабованість та API-інтеграцію, створюючи надійну основу для роботи гібридної інфраструктури.

Під час розгортання підготовлено базове середовище та налаштовано доступ до хмарних ресурсів із застосуванням механізмів керування обліковими записами та політик безпеки. Інтеграція інструментів командного рядка (CLI) забезпечила централізоване управління доступом без надлишкового дублювання конфігурацій. Створене середовище гарантує коректну автентифікацію, контроль операцій та високий рівень захисту даних.

Алгоритми реплікації включають створення структури директорій, розробку bash-скриптів обробки даних, інкрементне локальне дублювання та асинхронну передачу змін у хмару, що мінімізує обсяг трафіку і прискорює синхронізацію. Проведене тестування з моделюванням змін у файловій системі та моніторингом журналів підтвердило працездатність системи, коректність операцій і узгодженість даних на всіх рівнях зберігання.

Аналіз роботи системи дозволив оптимізувати конфігурацію, автоматизувати управління журналами та зменшити мережеве навантаження. Гібридна модель довела свою перевагу над готовими комерційними сервісами завдяки підвищеній гнучкості та незалежності від провайдера. Економічна оцінка підтвердила мінімальні початкові витрати за рахунок використання безкоштовних тарифів і відкритого ПЗ, а застосування політик життєвого циклу створює умови для подальшого масштабування системи з оптимальним балансом вартості, продуктивності та доступності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Hybrid cloud infrastructure overview. IBM. URL: https://www.ibm.com/topics/hybrid-cloud (дата звернення: 25.02.2026).
2. Cloud vs on-premises storage architecture. Microsoft Azure. URL: https://learn.microsoft.com/en-us/azure/architecture/guide/technology-choices/storage (дата звернення: 28.02.2026).
3. Advantages and disadvantages of hybrid cloud. Red Hat. URL: https://www.redhat.com/en/topics/cloud-computing/hybrid-cloud (дата звернення: 03.03.2026).
4. Data reliability in distributed systems. Google Cloud. URL: https://cloud.google.com/architecture/reliability (дата звернення: 07.03.2026).
5. Amazon S3 documentation. Amazon Web Services. URL: https://docs.aws.amazon.com/s3/index.html (дата звернення: 10.03.2026).
6. S3 access control mechanisms. AWS IAM Guide. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html (дата звернення: 14.03.2026).
7. Object storage vs file storage. NetApp. URL: https://www.netapp.com/data-storage/object-storage/ (дата звернення: 18.03.2026).
8. Rclone documentation. Rclone.org. URL: https://rclone.org/docs/ (дата звернення: 22.03.2026).
9. AWS CLI user guide. Amazon Web Services. URL: https://docs.aws.amazon.com/cli/latest/userguide/ (дата звернення: 25.03.2026).
10. Data replication tools comparison. Backblaze. URL: https://www.backblaze.com/blog/cloud-backup-vs-cloud-sync/ (дата звернення: 29.03.2026).
11. TLS protocol overview. Cloudflare. URL: https://www.cloudflare.com/learning/ssl/what-is-tls/ (дата звернення: 02.04.2026).
12. Client-side encryption concepts. AWS Security. URL: https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingClientSideEncryption.html (дата звернення: 06.04.2026).
13. API keys and token management. Google Cloud. URL: https://cloud.google.com/docs/authentication (дата звернення: 10.04.2026).
14. Data lifecycle policies. AWS S3 Lifecycle. URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/object-lifecycle-mgmt.html (дата звернення: 13.04.2026).
15. Backup strategy 3-2-1 rule. Veeam. URL: https://www.veeam.com/blog/321-backup-rule.html (дата звернення: 17.04.2026).
16. Logging and monitoring in cloud systems. AWS CloudWatch. URL: https://docs.aws.amazon.com/cloudwatch/ (дата звернення: 20.04.2026).
17. Ubuntu Server documentation. Canonical. URL: https://ubuntu.com/server/docs (дата звернення: 23.04.2026).

18. AWS IAM best practices. Amazon Web Services. URL: https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html (дата звернення: 26.04.2026).
19. Configuring AWS CLI and credentials. AWS Docs. URL: https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html (дата звернення: 29.04.2026).
20. Bash scripting guide. GNU. URL: https://www.gnu.org/software/bash/manual/ (дата звернення: 02.05.2026).
21. Rsync incremental backup. Linux Documentation Project. URL: https://tldp.org/LDP/abs/html/rsync.html (дата звернення: 02.05.2026).
22. Rclone sync command usage. Rclone.org. URL: https://rclone.org/commands/rclone_sync/ (дата звернення: 02.05.2026).
23. Logrotate manual. Linux man-pages. URL: https://linux.die.net/man/8/logrotate (дата звернення: 03.05.2026).
24. Network optimization techniques. Cisco. URL: https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-network-optimization.html (дата звернення: 04.05.2026).
25. SaaS vs hybrid cloud comparison. Oracle. URL: https://www.oracle.com/cloud/what-is-saas/ (дата звернення: 05.05.2026).
26. Multi-cloud and vendor independence. Google Cloud. URL: https://cloud.google.com/multi-cloud (дата звернення: 04.05.2026).
27. AWS Free Tier details. Amazon Web Services. URL: https://aws.amazon.com/free/ (дата звернення: 05.05.2026).
28. Amazon S3 pricing. AWS Pricing. URL: https://aws.amazon.com/s3/pricing/ (дата звернення: 05.05.2026).
29. S3 storage classes overview. AWS Docs. URL: https://aws.amazon.com/s3/storage-classes/ (дата звернення: 05.05.2026).
30. Федоренко Д. О., Торошанко Я. І. Програмно-апаратний комплекс синхронізації даних у гібридній локально-хмарній інфраструктурі. «Сучасний стан та перспективи розвитку IoT» : VII Науково-техн. конф., м. Київ, 20 квітня 2026 р. С.292-294.
31. Федоренко Д. О., Торошанко Я. І. Механізми захисту та контролю доступу в програмно-апаратному комплексі синхронізації даних у гібридній інфраструктурі. «Застосування програмного забезпечення в ІКТ» : VII Всеукр. науково-техн. конф., м. Київ, 23 квітня 2026 р. (подано до публікації).

Додаток А

Лістинг конфігураційного файлу політики доступу AWS IAM (S3SyncPolicy.json)

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::amzn-s3-fedor-dmgr-bucket",
                "arn:aws:s3:::amzn-s3-fedor-dmgr-bucket/*"
            ]
        }
    ]
}
```

Додаток Б

Вихідний код bash-скрипта автоматизованої двоетапної синхронізації даних (sync_script.sh)

```
#!/bin/bash

# Variables
SOURCE="/data/main/"
LOCAL_BACKUP="/data/local_backup/"
REMOTE="aws-s3"
BUCKET="amzn-s3-fedor-dmgr-bucket"
LOG_FILE="/var/log/sync_service.log"

# Logging start
echo "$(date): Sync started" >> "$LOG_FILE"

# 1. Local mirroring (rsync)
rsync -avz --delete "$SOURCE" "$LOCAL_BACKUP" >> "$LOG_FILE" 2>&1

# 2. Cloud sync (rclone)
rclone sync "$LOCAL_BACKUP" "$REMOTE:$BUCKET/backup" --log-file="$LOG_FILE" --log-
level INFO

# Check status
if [ $? -eq 0 ]; then
    echo "$(date): Sync successful" >> "$LOG_FILE"
else
    echo "$(date): Sync failed" >> "$LOG_FILE"
fi
}
```

Додаток В

Конфігурація системи ротації та архівування логів (logrotate)

```
/var/log/sync_service.log {
    daily
    rotate 7
    compress
    missingok
    notifempty
    create 0640 dmytro-fedorenko dmytro-fedorenko
}
```

Додаток Г

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерної інженерії



КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

НА ТЕМУ: «ПРОГРАМНО-АПАРАТНИЙ КОМПЛЕКС СИНХРОНІЗАЦІЇ ДАНИХ У ГІБРИДНІЙ ЛОКАЛЬНО-ХМАРНІЙ ІНФРАСТРУКТУРІ»

Виконав студент групи КІД-41
Федоренко Дмитро Олександрович
Керівник кваліфікаційної роботи:
Торошанко Ярослав Іванович,
канд. техн. наук, доц., с.н.с

Київ – 2026

Мета, об’єкт, предмет дослідження

Мета роботи – оптимізація процесу синхронізації та резервного копіювання даних у гібридній локально-хмарній інфраструктурі за рахунок використання інкрементних методів передачі, автоматизації процесів та об’єктних моделей зберігання

Об’єкт дослідження – процеси синхронізації та реплікації даних у гібридних локально-хмарних середовищах

Предмет дослідження – засоби та технології побудови, налаштування й оптимізації систем синхронізації даних на базі локальних серверів і хмарних сервісів об’єктного зберігання

Актуальність роботи

У роботі розглянуто шляхи до побудови гібридних систем зберігання даних, архітектуру локальних та хмарних сховищ, принципи об’єктного зберігання (Amazon S3) та механізми забезпечення безпеки. Проаналізовано функціональні обмеження інструментів синхронізації (rclone, AWS CLI), зокрема щодо залежності від провайдера та ефективності оптимізації.

Практична частина включає розробку програмно-апаратного комплексу на базі Ubuntu Server, інтегрованого з хмарним середовищем AWS (S3) через rclone. Реалізовано алгоритми локального дублювання (rsync), асинхронної передачі даних та автоматизації за допомогою bash-скриптів. Проведене тестування підтвердило цілісність даних та працездатність системи, а аналіз ефективності та економічне обґрунтування довели доцільність впровадження запропонованої гібридної моделі.

Загальна архітектура рішення

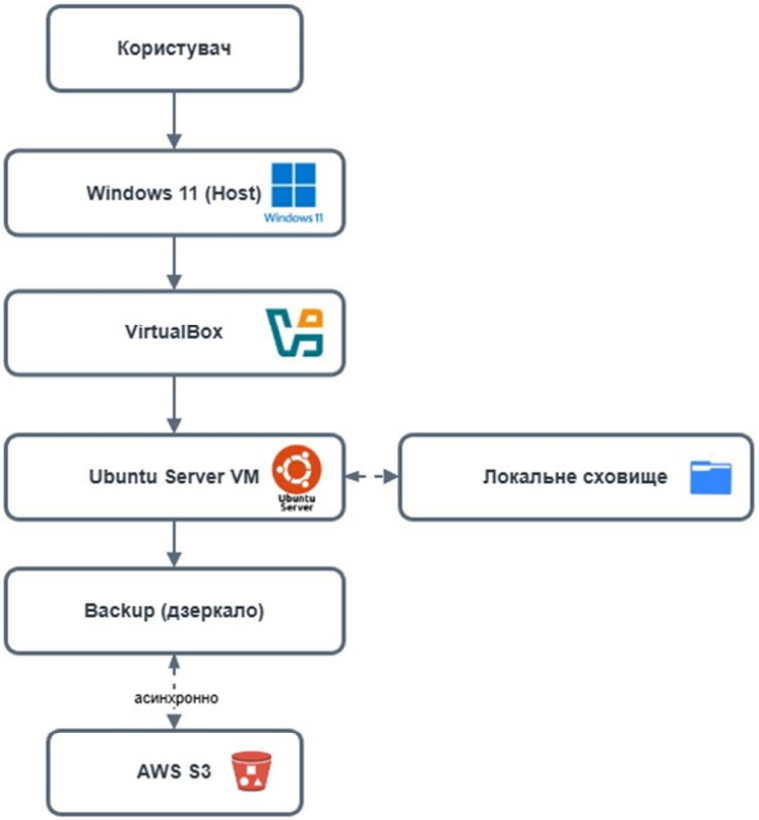


Рис. 2.1 – Архітектура програмно-апаратного комплексу синхронізації на базі Ubuntu Server та AWS

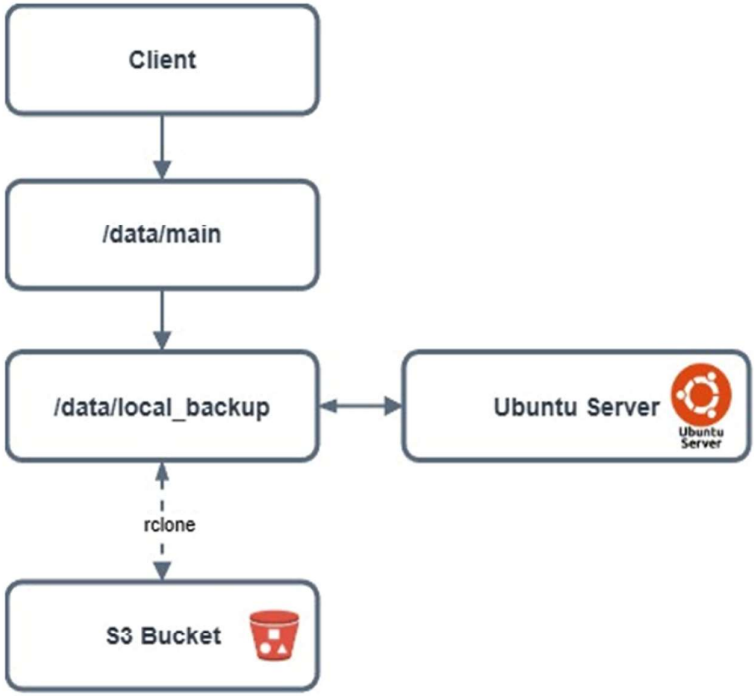


Рис. 2.3 – Схема взаємодії клієнта, локального сервера та AWS S3

Апаратно-програмне середовище

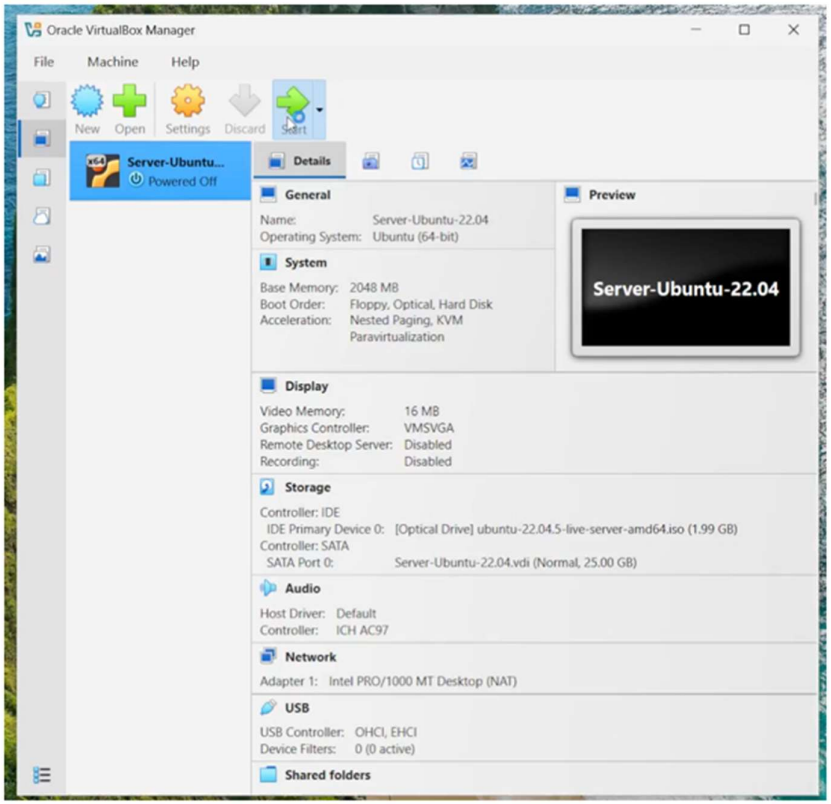


Рис. 2.2 – Віртуалізоване середовище на базі Ubuntu Server

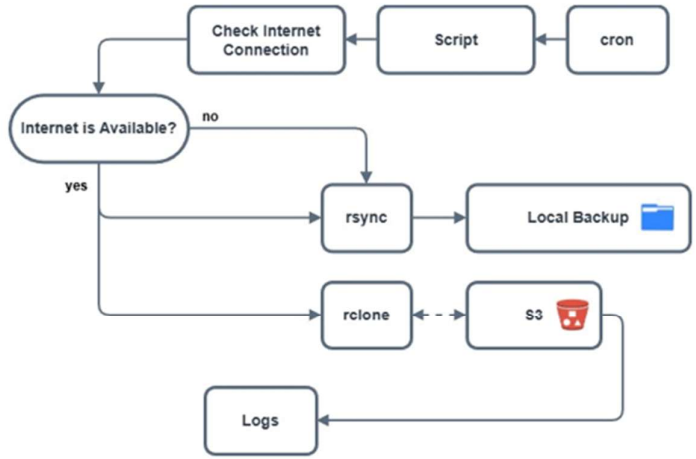


Рис. 2.4 – Структура програмного стека системи

Табл. 2.1 – Характеристики апаратного та програмного середовища

Параметр	Значення
Хостова ОС	Windows 11
Гіпервізор	Oracle VM VirtualBox
Гостьова ОС	Ubuntu Server 22.04 LTS
CPU	2 віртуальні ядра
RAM	2 ГБ
Диск	30ГБ
Мережа	Bridge (окрема IP-адреса)
Основне ПЗ	AWS CLI, rclone

Конфігурація системи

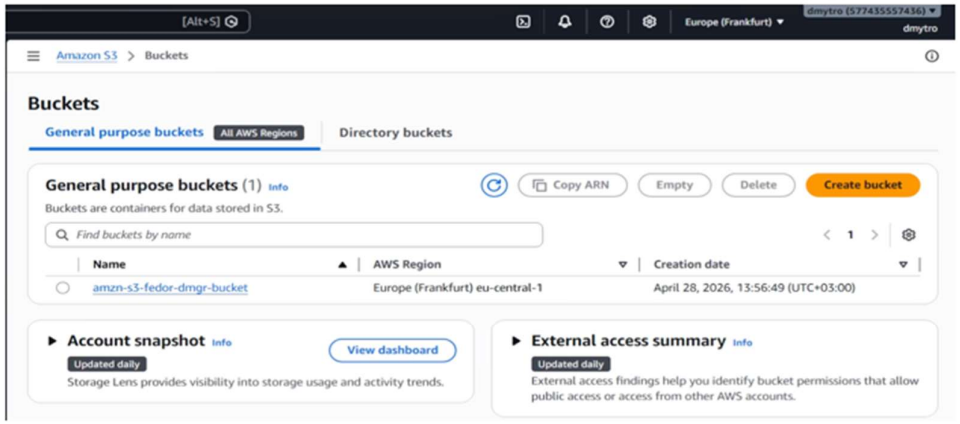


Рис. 2.8 – Конфігурація S3 бакета

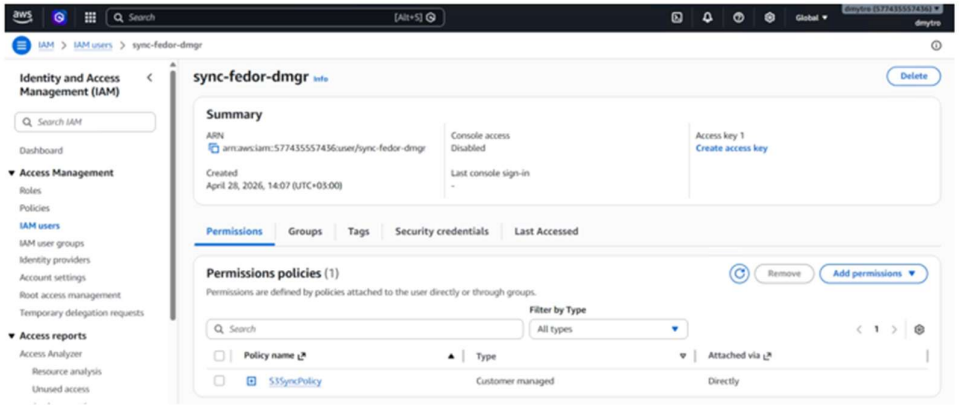


Рис. 2.9 – Налаштування IAM користувача в AWS

Табл. 2.2 – Основні параметри конфігурації AWS та rclon

Параметр	Значення
Тип автентифікації	Access Key зі Secret Key
Регіон	eu-central-1
Тип сховища	Amazon S3 Bucket
Формат виводу AWS CLI	JSON
rclone remote	Тип S3
Метод доступу	env_auth (через AWS CLI)
Endpoint	AWS S3 API

```
dmytro-fedorenko@server-ubuntu-2204:~$ aws configure
AWS Access Key ID [None]: AKIAYM4OXLY6AOBS5ZFQ
AWS Secret Access Key [None]:
Default region name [None]: eu-central-1
Default output format [None]: json
dmytro-fedorenko@server-ubuntu-2204:~$
```

Рис. 2.11 – Налаштування AWS CLI у терміналі

```
dmytro-fedorenko@server-ul
location_constraint = eu-central-1
--------------------
y) Yes this is OK (default)
e) Edit this remote
d) Delete this remote
y/e/d> y
Current remotes:

Name                 Type
====                 ====
aws-s3               s3

e) Edit existing remote
n) New remote
d) Delete remote
r) Rename remote
c) Copy remote
s) Set configuration password
q) Quit config
e/n/d/r/c/s/q> q
dmytro-fedorenko@server-ubuntu-2204:~$ rclone ls aws-s3:amzn-s3-fedor-dmgr-bucket
        0 file.txt
dmytro-fedorenko@server-ubuntu-2204:~$
```

Рис. 2.12 – Конфігурація rclone (інтерактивне меню)

6

Алгоритм роботи системи

```
dmytro-fedorenko@server-ul
 GNU nano 6.2                    /home/dmytro-fedorenko/sync_script.sh *
#!/bin/bash

# Variables
SOURCE="/data/main/"
LOCAL_BACKUP="/data/local_backup/"
REMOTE="aws-s3"
BUCKET="amzn-s3-fedor-dmgr-bucket"
LOG_FILE="/var/log/sync_service.log"

# Logging start
echo "$(date): Sync started" >> "$LOG_FILE"

# 1. Local mirroring (rsync)
rsync -avz --delete "$SOURCE" "$LOCAL_BACKUP" >> "$LOG_FILE" 2>&1

# 2. Cloud sync (rclone)
rclone sync "$LOCAL_BACKUP" "$REMOTE:$BUCKET/backup" --log-file="$LOG_FILE" --log-level INFO

# Check status
if [ $? -eq 0 ]; then
    echo "$(date): Sync successful" >> "$LOG_FILE"
else
    echo "$(date): Sync failed" >> "$LOG_FILE"
fi

^G Help        ^O Write Out   ^W Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

Рис. 2.13 – Структура bash-скрипта синхронізації

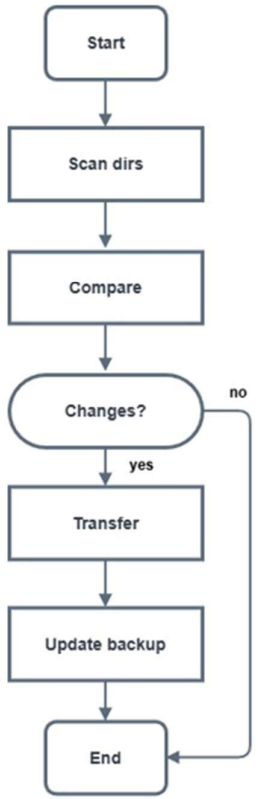


Рис. 2.14 – Алгоритм локального резервного копіювання

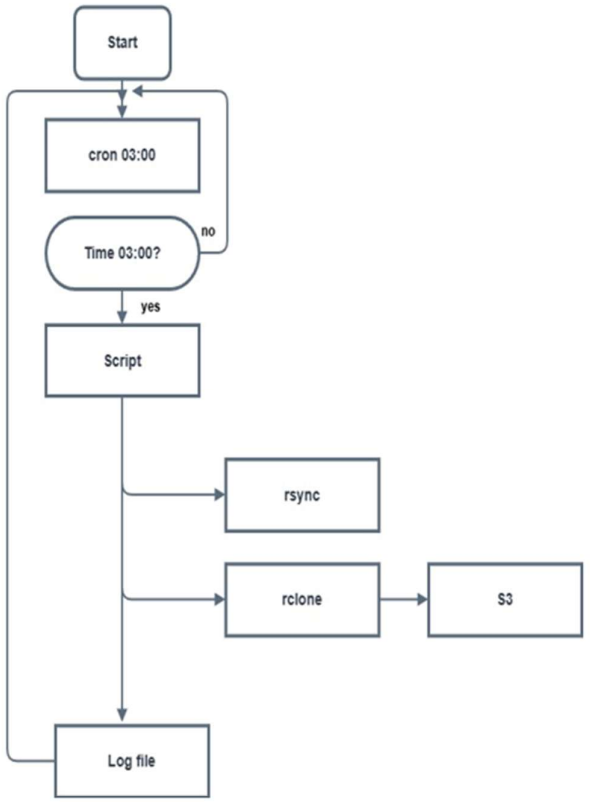


Рис. 2.15 – Алгоритм асинхронної передачі даних у хмару

Тестування системи

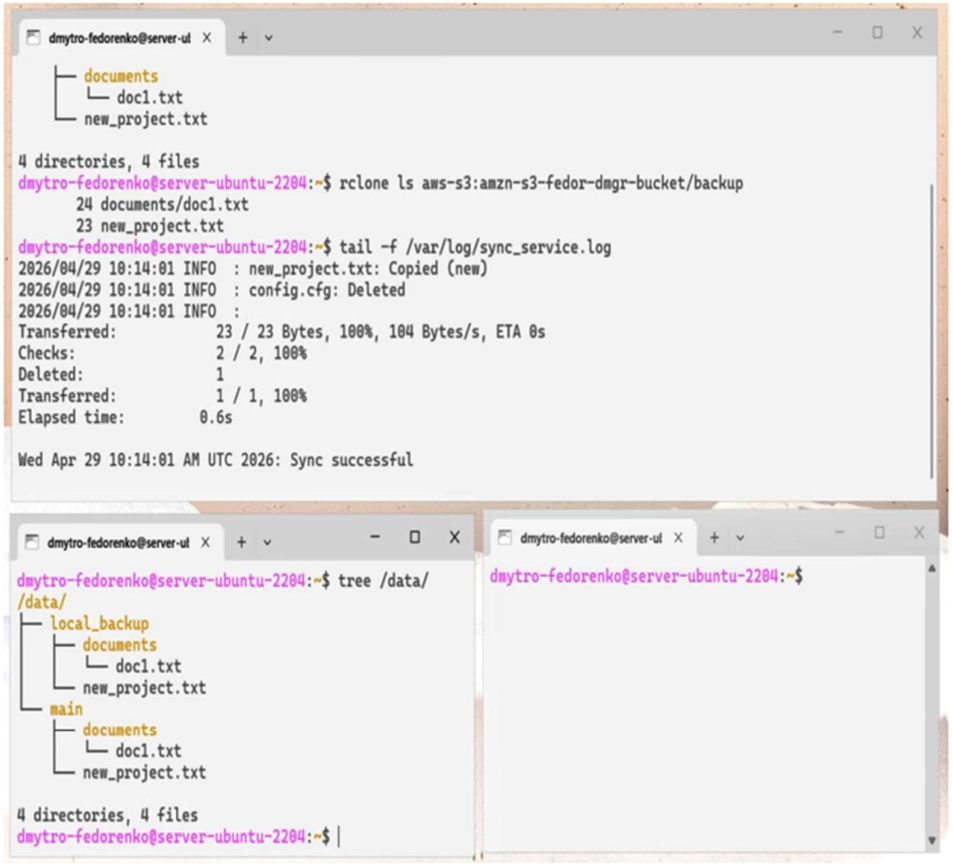


Рис. 2.16 – Використання трьох терміналів для тестування

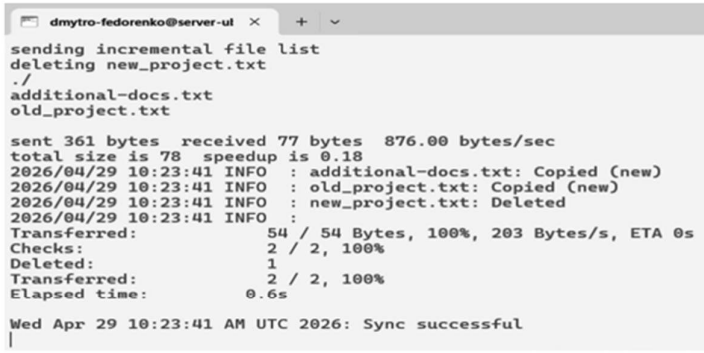


Рис. 2.18 – Моніторинг логів у режимі реального часу

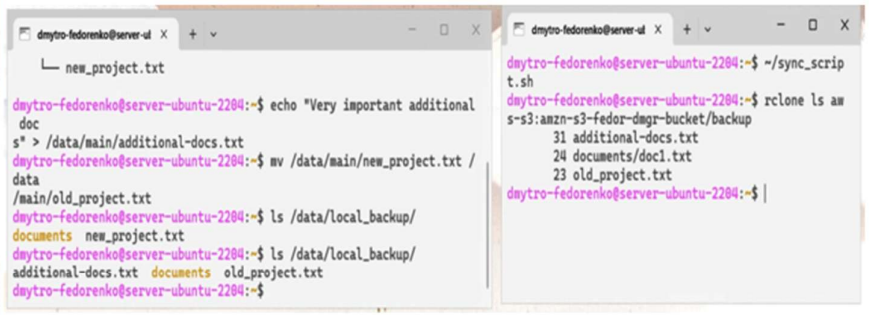


Рис. 2.19 – Перевірка цілісності даних між середовищами

Методи оптимізації

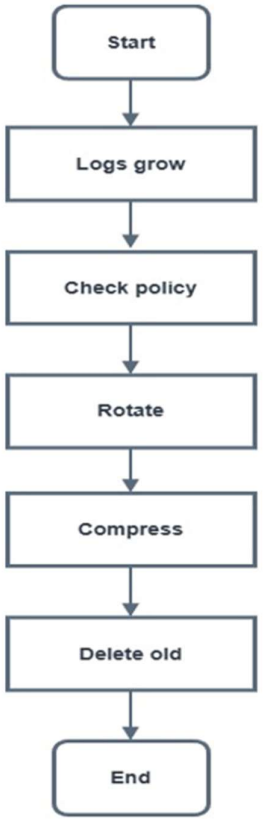


Рис. 3.3 – Метод ротації логів (Logrotate)

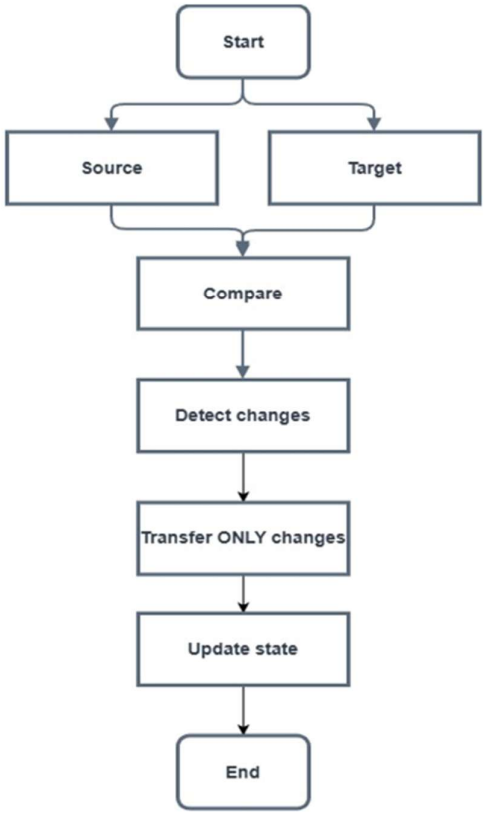


Рис. 3.4 – Метод зменшення мережевого навантаження

Порівняння з SaaS

Табл. 3.1 – Порівняння гібридної системи з SaaS-рішеннями

Критерій	SaaS-рішення	Гібридна система
Контроль над даними	Обмежений	**Повний / частковий**
Залежність від вендора	Висока	**Низька**
Гнучкість	Обмежена	**Висока**
Вартість	Підписка	**Гнучка (pay-as-you-go)**
Продуктивність	Залежить від мережі	**Висока (локальна обробка)**
Масштабованість	Висока	**Висока**

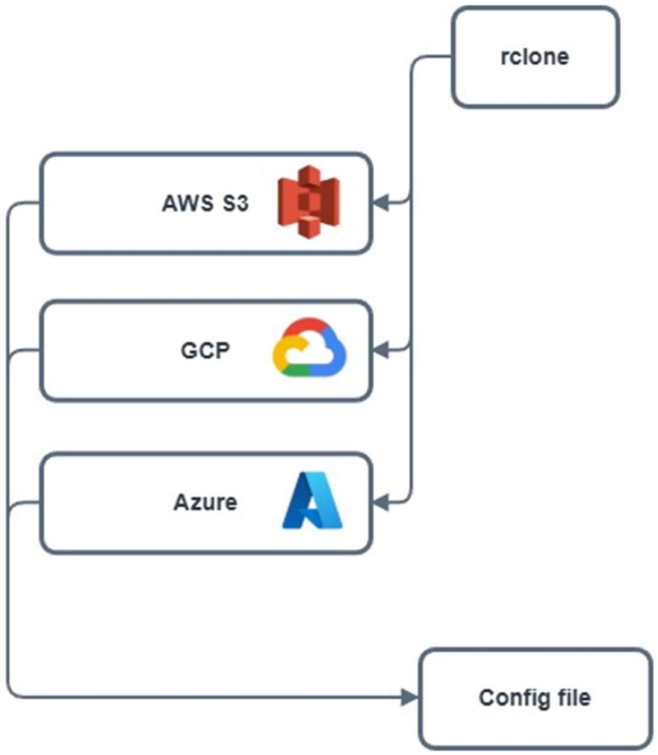


Рис. 3.6 – Мультихмарна архітектура на базі rclone

Економічна ефективність

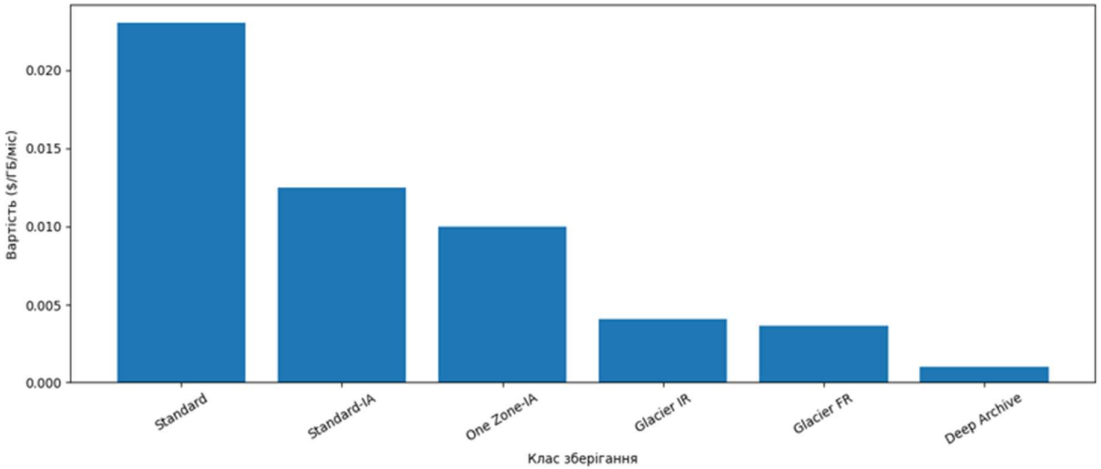


Рис. 3.7 – Структура витрат гібридної інфраструктури

Табл. 3.2 – Порівняння класів зберігання Amazon S3 та їх вартості

Клас зберігання	Вартість ($/ГБ/міс)	Час доступу	Сценарій використання
Standard	~0.023	Миттєвий	Частий доступ
Standard-IA	~0.0125	Миттєвий	Рідкий доступ
One Zone-IA	~0.01	Миттєвий	Некритичні дані
Glacier Instant Retrieval	~0.004	Мілісекунди	Архів з швидким доступом
Glacier Flexible Retrieval	~0.0036	Хвилини	Резервні копії
Glacier Deep Archive	~0.00099	Години	Довготривале архівування

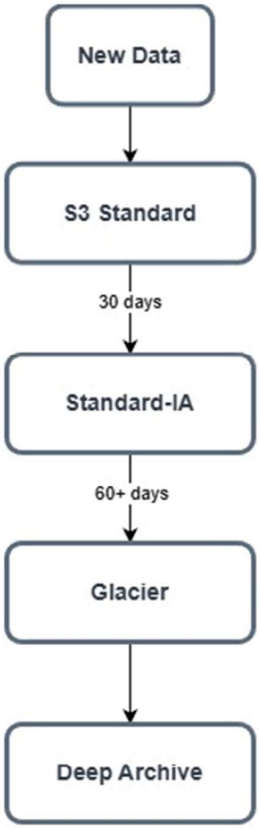


Рис. 3.8 – Перехід даних між класами зберігання

Висновки

У процесі реалізації сформовано архітектуру гібридного комплексу синхронізації, що поєднує локальний сервер і хмарне об’єктне сховище. Визначено топологію, налаштовано взаємодію між клієнтом, сервером і S3, а також обґрунтовано програмний стек, який забезпечує сумісність, масштабованість і безпечну інтеграцію. Розгортання включало підготовку середовища, налаштування доступу через CLI та політики безпеки, що гарантує коректну автентифікацію і контроль операцій.

Реалізовані алгоритми включають інкрементне локальне резервування та асинхронну передачу змін у хмару, що зменшує трафік і прискорює синхронізацію; тестування підтвердило узгодженість даних і стабільність системи. Проведена оптимізація дозволила знизити навантаження, автоматизувати логування та підвищити ефективність, тоді як гібридна модель продемонструвала гнучкість і економічну доцільність завдяки використанню відкритого ПЗ та політик життєвого циклу.

Результати впровадження

1. Федоренко Д.О., Торошанко Я.І. Програмно-апаратний комплекс синхронізації даних у гібридній локально-хмарній інфраструктурі. «Сучасний стан та перспективи розвитку IoT» : VII Науково-техн. конф., м. Київ, 20 квітня 2026 р. С.292-294.
2. Федоренко Д.О., Торошанко Я.І. Механізми захисту та контролю доступу в програмно-апаратному комплексі синхронізації даних у гібридній інфраструктурі. «Застосування програмного забезпечення в ІКТ» : VII Всеукр. науково-техн. конф., м. Київ, 23 квітня 2026 р. (подано до публікації).