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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ МОНІТОРИНГУ ТА ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ХМАРНИХ СХОВИЩ»

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

_____ *Андрій ЛИСЕНКО*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КІД-42
Андрій ЛИСЕНКО
Ім'я, ПРІЗВИЩЕ

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

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

Київ 2025

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
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Лисенко Андрій Григорович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Програмний комплекс для моніторингу та оптимізації продуктивності хмарних сховищ»
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 «24» лютого 2025р. №56
2. Строк подання кваліфікаційної роботи «05» червня 2025р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, стандарти та протоколи передачі даних, архітектура та компоненти хмарних сховищ, мережеві обмеження та їхній вплив на продуктивність, методи вимірювання та аналізу швидкодії, тестування та порівняльний аналіз хмарних платформ.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
    1. Теоретичні засади моніторингу та оптимізації хмарних сховищ
    2. Проєктування та реалізація архітектури програмного комплексу
    3. Застосування методів покращення та перспективи розвитку комплексу
5. Перелік ілюстративного матеріалу: *презентація*
    1. Мета, об'єкт, предмет дослідження
    2. Актуальність роботи
    3. Проблеми продуктивності у хмарних середовищах зберігання
    4. Архітектура програмного комплексу моніторингу та оптимізації
    5. Побудова віртуалізованого середовища на базі VirtualBox
    6. Реалізація базового функціоналу програмного комплексу
    7. Встановлення та налаштування Rclone Remote Control для управління передачею
    8. Побудова готових дашбордів для візуалізації метрик на базі Grafana
    9. Встановлення та налаштування агентів Prometheus – Node Exporter (для деталізованого моніторингу всіх вузлів)

10. Побудова кастомного дашборда для візуалізації метрик на базі Grafana
11. Реалізація методів оптимізації продуктивності
12. Порівняння продуктивності до та після оптимізації
13. Результати тестування і підсумки експериментів
14. Висновки
15. Публікації та апробація роботи

6. Дата видачі завдання «24» лютого 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-19.03.2025	
2	Обробка матеріалу	20.03-02.04.2025	
3	Розробка теоретичної частини	03.04-07.04.2025	
5	Проєктування та реалізація архітектури програмного комплексу	08.04-27.04.2025	
6	Застосування методів покращення та перспективи розвитку комплексу	28.04-11.05.2025	
7	Висновки за результатами аналізу	12.05-15.05.2025	
8	Розробка презентації	16.05-26.05.2025	
9	Передзахист	27.05-29.05.2025	
10	Здача в деканат	30.05-05.06.2025	

Здобувач(ка) вищої освіти _____ Андрій ЛИСЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

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

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 101 стор., 93 рис., 7 табл., 29 джерел.

Мета роботи – оптимізація продуктивності хмарних сховищ за рахунок використання методів моніторингу, візуалізації метрик і автоматизації управління на основі відкритих інструментів.

Об'єкт дослідження – процеси функціонування та продуктивності хмарних сховищ у віртуалізованому середовищі.

Предмет дослідження – інструменти моніторингу, автоматизації та аналітики, що забезпечують стабільність і ефективність хмарних сховищ.

Короткий зміст роботи: У роботі розглянуто підходи до побудови програмного комплексу для моніторингу та оптимізації хмарних об'єктних сховищ із використанням віртуалізації та відкритого програмного забезпечення. Проаналізовано типи хмарних сховищ, основні метрики продуктивності, а також методи збору й візуалізації даних за допомогою Prometheus, Grafana та допоміжних агентів моніторингу.

Основну увагу зосереджено на реалізації системи у тестовому середовищі на базі VirtualBox із трьома вузлами: клієнтським, сервером об'єктного сховища та вузлом моніторингу. Детально описано процеси встановлення MinIO, налаштування збору метрик Prometheus, побудови кастомних дашбордів Grafana та перевірки зв'язності через Rclone. Проведено експериментальну оцінку продуктивності передачі даних до та після оптимізації параметрів багатопоточності та кешування, що дозволило зафіксувати суттєве покращення ефективності роботи системи.

КЛЮЧОВІ СЛОВА: ХМАРНЕ СХОВИЩЕ, МОНІТОРИНГ, ПРОДУКТИВНІСТЬ, ГРАФІЧНИЙ ІНТЕРФЕЙС, PROMETHEUS, GRAFANA, ВІЗУАЛІЗАЦІЯ МЕТРИК, ВІРТУАЛІЗАЦІЯ, ОПТИМІЗАЦІЯ, СИСТЕМА ЗБОРУ ДАНИХ, ДОСТУПНІСТЬ, ЕФЕКТИВНІСТЬ.

ABSTRACT

Text part of the qualification work for obtaining a bachelor's degree: 101 pages, 93 figures, 7 tables, 29 sources.

The purpose of the work is to optimize the performance of cloud storage through the use of monitoring methods, visualization of metrics and automation of management based on open tools.

The object of the study is the processes of functioning and performance of cloud storage in a virtualized environment.

The subject of the study is monitoring, automation and analytics tools that ensure the stability and efficiency of cloud storage.

Summary of the work: The paper considers approaches to building a software package for monitoring and optimizing cloud object storage using virtualization and open source software. Types of cloud storage, basic performance metrics, and methods of data collection and visualization using Prometheus, Grafana, and auxiliary monitoring agents are analyzed.

The main attention is focused on implementing the system in a test environment based on VirtualBox with three nodes: client, object storage server, and monitoring node. The processes of installing MinIO, configuring Prometheus metric collection, building custom Grafana dashboards, and checking connectivity via Rclone are described in detail. An experimental assessment of data transfer performance before and after optimizing multithreading and caching parameters was conducted, which allowed us to record a significant improvement in system efficiency.

KEYWORDS: CLOUD STORAGE, MONITORING, PRODUCTIVITY, GRAPHICAL INTERFACE, PROMETHEUS, GRAFANA, METRICS VISUALIZATION, VIRTUALIZATION, OPTIMIZATION, DATA COLLECTION SYSTEM, AVAILABILITY, EFFICIENCY.

ЗМІСТ

ВСТУП..... 9
РОЗДІЛ 1 ТЕОРЕТИЧНІ ЗАСАДИ МОНІТОРИНГУ ТА ОПТИМІЗАЦІЇ ХМАРНИХ СХОВИЩ..... 10
1.1 Основні поняття та класифікація хмарних сховищ даних..... 10
1.2 Проблеми продуктивності у хмарних середовищах зберігання..... 17
1.3 Метрики продуктивності та методи їх вимірювання……26
1.4 Підходи до моніторингу продуктивності ІТ-систем 35
РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АРХІТЕКТУРИ ПРОГРАМНОГО КОМПЛЕКСУ..... 45
2.1 Архітектура системи моніторингу та оптимізації хмарних сховищ..... 45
2.2 Аналіз та вибір інструментів реалізації 51
2.3 Побудова віртуалізованого середовища на базі VirtualBox..... 60
2.4 Реалізація та налаштування базового функціоналу комплексу 67
РОЗДІЛ 3 ЗАСТОСУВАННЯ МЕТОДІВ ПОКРАЩЕННЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ КОМПЛЕКСУ..... 86
3.1 Реалізація методів покращення продуктивності..... 86
3.2 Проблеми, що виникають при роботі комплексу, та способи їх вирішення 96
3.3 Перспективи масштабування системи в умовах реального ІТ-середовища ... 104
ВИСНОВКИ..... 109
ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ……...……110
Додаток А. Фрагмент JSON-конфігурації дашборда програмного комплексу моніторингу та оптимізації продуктивності хмарного сховища з трьохвузловою архітектурою……112
Додаток Б. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)…….....115

ВСТУП

Зі стрімким зростанням обсягів цифрової інформації та активним впровадженням хмарних технологій питання ефективного зберігання даних стало центральним для сучасних IT-систем. Хмарні сховища забезпечують високу доступність, гнучкість у масштабуванні та централізоване керування, що особливо важливо для динамічних і розподілених інфраструктур.

Моделі зберігання бувають різних типів: об’єктні, блочні та файлові. Об’єктне зберігання найкраще підходить для великих обсягів неструктурованих даних, блочне — для систем, де критична швидкість доступу, а файлове — для організації спільної роботи. Вибір відповідної моделі визначає гнучкість та продуктивність всієї системи.

Продуктивність хмарного сховища оцінюється за низкою ключових метрик: IOPS, latency, throughput та cache hit ratio. Вимірювання цих показників дозволяє вчасно виявляти перевантаження, визначати ефективність використання ресурсів і приймати обґрунтовані рішення щодо оптимізації.

Моніторинг продуктивності може реалізовуватись як агентським способом (встановленням спеціалізованого ПЗ на сервери), так і безагентним — через API або стандартні протоколи SNMP. Інструменти на зразок Prometheus, Zabbix та Netdata дають змогу збирати ці дані в реальному часі, аналізувати історію навантажень, виявляти аномалії та автоматизувати сповіщення. Поєднання збору метрик із сучасними методами оптимізації, як-от кешування, використання файлових систем зі стисненням, багаторівневе зберігання, створює технологічну основу для побудови ефективної, гнучкої та масштабованої системи хмарного зберігання.

РОЗДІЛ 1 ТЕОРЕТИЧНІ ЗАСАДИ МОНІТОРИНГУ ТА ОПТИМІЗАЦІЇ ХМАРНИХ СХОВИЩ

1.1 Основні поняття та класифікація хмарних сховищ даних

Хмарне сховище даних — це технологічна модель, що передбачає зберігання інформації на віддалених серверах із можливістю доступу до неї через мережу Інтернет. Передача даних до таких сховищ здійснюється через захищені канали, що забезпечує конфіденційність і цілісність інформації. Такий підхід дозволяє уникати залежності від фізичних носіїв і локальної IT-інфраструктури, надаючи змогу масштабувати ресурси відповідно до актуальних потреб, а не заздалегідь визначених технічних обмежень. Обсяги зберігання можуть змінюватися гнучко — їх можна оперативно збільшити чи зменшити в залежності від кількості даних та інтенсивності користування (див. рис. 1.1) [1].

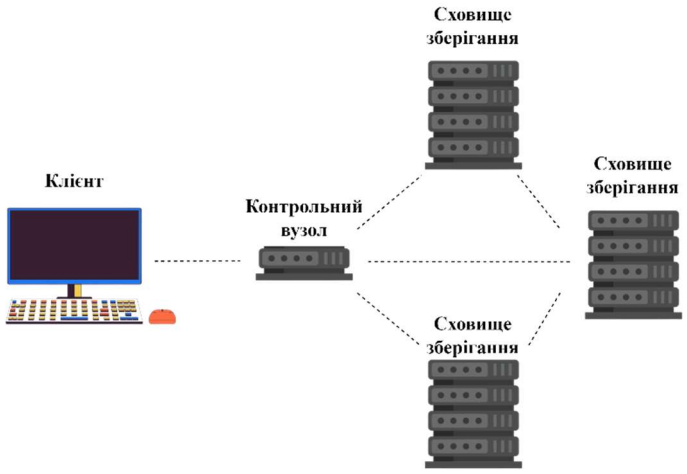


Рисунок 1.1 – Структура хмарного сховища даних

Однією з основних характеристик хмарних сховищ є те, що користувач не взаємодіє безпосередньо з апаратним забезпеченням. Усі дані розміщуються у так званій хмарі та доступні з будь-якої точки світу за наявності з’єднання з

Інтернетом. Фізичне розташування серверів при цьому залишається прозорим, і не потребує втручання або технічного обслуговування з боку кінцевого користувача.

Завдяки такій організації зберігання даних значно підвищується рівень доступності, спрощується технічна підтримка, реалізується автоматизоване резервне копіювання та відновлення, що в традиційних локальних рішеннях часто вимагало значних зусиль і ресурсів. У більшості випадків хмарні сховища надаються у вигляді послуги від спеціалізованих компаній, які орендують або володіють дата-центрами, хоча за певних умов можливо й створення приватного хмарного сховища, розгорнутого в межах підприємства або організації, що вимагає додаткових технічних рішень та ресурсів [1-2].

Типи хмарних сховищ. Типи хмарних сховищ не можна розглядати як єдиний або уніфікований підхід до зберігання даних, адже існує кілька моделей, які принципово відрізняються за архітектурою, способом доступу та сферами застосування. Найпоширенішими з них є [1]:

— об’єктне сховище;

— блочне сховище;

— файлове сховище.

Об’єктне сховище (Object Storage) побудоване на концепції зберігання даних у вигляді окремих незалежних об’єктів, кожен з яких має унікальний ідентифікатор, набір метаданих та власний вміст. Доступ до таких об’єктів здійснюється через мережу, як правило, за допомогою HTTP-запитів через RESTful API, що робить цю модель зручною для роботи у розподілених середовищах. Серед найвідоміших прикладів — Amazon S3, Google Cloud Storage і MinIO, останній із яких розглядатиметься далі у роботі. Основна перевага об’єктного сховища полягає у можливості легкого масштабування до колосальних обсягів — аж до петабайтів, що дозволяє уникнути значних витрат на локальне зберігання подібного обсягу даних. Завдяки своїй архітектурі об’єктне сховище забезпечує високу відмовостійкість, адже зазвичай експлуатується у хмарному або корпоративному середовищі, де впроваджено системи моніторингу, реплікації й автоматичного відновлення. Особливо ефективно ця модель працює зі зберіганням

неструктурованих даних, таких як зображення, відео або резервні копії. Водночас вона практично не пристосована для безпосередньої інтеграції в традиційні файлові системи, адже не підтримує прямий доступ з операційної системи, що унеможливлює її повноцінне використання у локальному середовищі без спеціалізованих інструментів (див. рис. 1.2) [1-2].

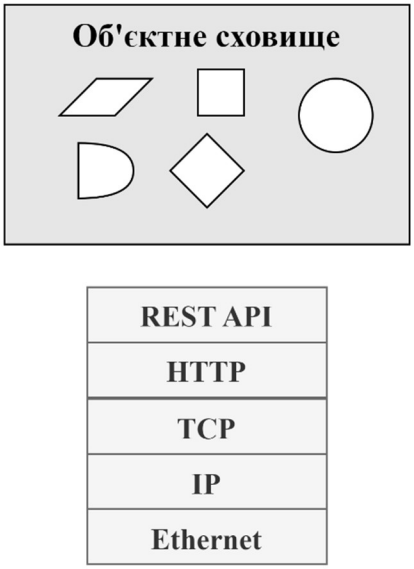


Рисунок 1.2 – Принцип роботи об’єктного сховища

Блочне сховище (Block Storage) організовує дані у вигляді фіксованих блоків, які можуть монтуватися як окремі віртуальні жорсткі диски. На відміну від об’єктного, така модель не оперує метаданими на рівні самих блоків, а контроль і управління здійснюються безпосередньо файловою системою операційної системи користувача. Типовими прикладами таких рішень є Amazon EBS або Azure Disk Storage. Блочне сховище часто використовується як основа для розгортання віртуальних машин, баз даних або серверного програмного забезпечення, адже забезпечує високу продуктивність і мінімальні затримки. Можливість інсталювати операційні системи та працювати з ними так, ніби йдеться про фізичний диск, робить цей тип зберігання особливо корисним у середовищах віртуалізації. Однак масштабування у порівнянні з об’єктними сховищами є менш гнучким і потребує складнішого управління, особливо у великих середовищах. Крім того,

функціонування блочного сховища нерозривно пов’язане з конкретною інфраструктурою віртуалізації, що обмежує його універсальність і підвищує залежність від обраної платформи (див. рис. 1.3) [1-2].

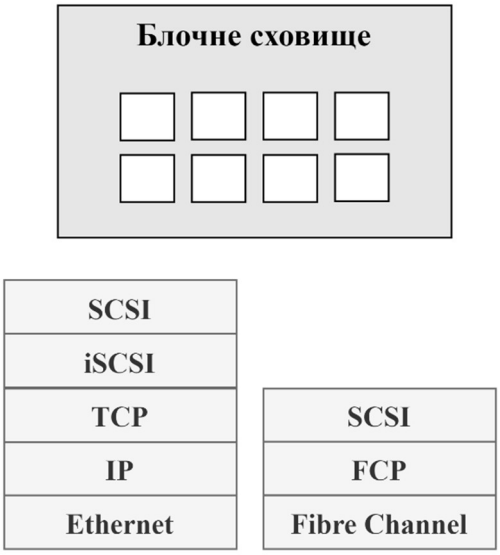


Рисунок 1.3 – Принцип роботи блочного сховища

Файлове сховище (File Storage) у своїй структурі найкраще відображає класичну модель організації даних, з якою знайомі більшість користувачів. Дані тут представлені у вигляді файлів, упорядкованих за ієрархією каталогів, доступ до яких забезпечується через протоколи на кшталт NFS, SMB або FTP. У цьому випадку передбачено клієнт-серверну архітектуру, де один пристрій виступає як сервер і надає доступ до своєї файлової системи іншим пристроям, які діють як клієнти. Такий підхід є найбільш зручним для щоденної роботи, спільного редагування документів та обміну файлами в офісному середовищі. Попри інтуїтивність і доступність, файлове сховище має обмежену масштабованість — при потребі в обробці великих обсягів даних необхідно втручатися в апаратну частину: додавати накопичувачі, змінювати масиви або перебудовувати архітектуру, що не лише ускладнює управління, а й знижує загальну гнучкість. Крім того, забезпечення відмовостійкості без зовнішніх засобів резервного копіювання або реплікації також є проблематичним. У разі, якщо пристрій, який виконує роль сервера, буде вимкнено або вийде з ладу, усі підключені до нього

клієнти втратять доступ до даних, що створює ризики як для особистого користування, так і для бізнесу (див. рис. 1.4) [1-2].

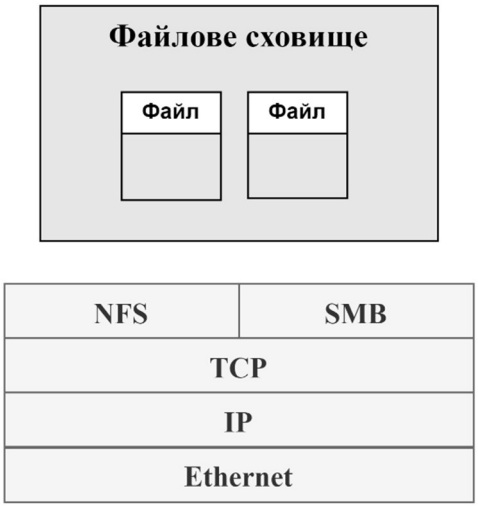


Рисунок 1.4 – Принцип роботи файлового сховища

У підсумку, кожен тип хмарного сховища орієнтований на конкретні потреби й сценарії використання, а вибір між ними залежить від вимог до масштабованості, продуктивності, сумісності з інфраструктурою та рівня доступності.

Порівняльний аналіз та приклади використання. Існування трьох основних типів — об’єктного, блочного та файлового — пояснюється різною природою сценаріїв їх застосування. Кожен із типів краще підходить для певних цілей, тому універсального рішення, здатного однаково ефективно охопити всі можливі задачі, фактично не існує. Розмежування між цими типами є логічним наслідком технічних вимог та функціональної специфіки.

У випадках з мультимедіа, резервними копіями чи зберіганням журналів подій найдоцільніше використовувати об’єктне сховище. Коли йдеться про критично важливі системи, що працюють із базами даних, більш прийнятним виявляється блочне сховище. Для спільної роботи з документами та організації доступу до файлів більш природним виглядає використання файлового сховища. Відповідно, реальні приклади сервісів ілюструють це розмежування: Google Drive і Dropbox — типові представники хмарних файлових сховищ, де акцент робиться

на спільному редагуванні документів і синхронізації файлів між користувачами. Amazon S3 або MinIO працюють як об’єктні сховища, орієнтовані на зберігання великої кількості незмінних даних, таких як мультимедійний контент або резервні копії. Рішення на кшталт Amazon EBS чи Azure Disk демонструють приклад блочного сховища, що активно застосовується для баз даних, віртуальних машин або інших систем, яким потрібен прямий доступ до блоків даних. Попри те, що всі ці сервіси належать до категорії хмарних, їхня внутрішня архітектура суттєво відрізняється, що прямо впливає на характер і ефективність використання (див. рис. 1.5) [1-2].

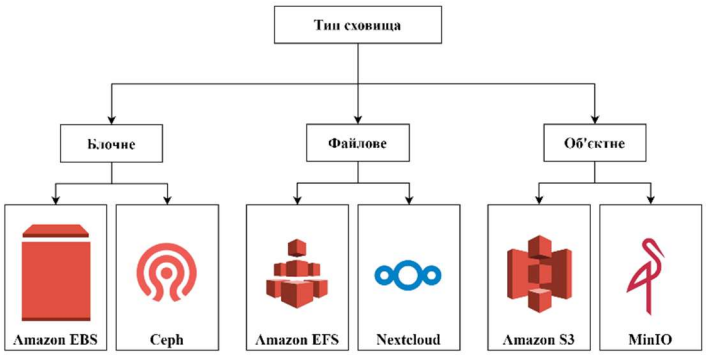


Рисунок 1.5 – Приклади використання різних типів сховища

У підсумку, визначення оптимального типу хмарного сховища напряму пов’язане з особливостями завдань, які мають бути реалізовані. Для роботи з документами, організації офісного середовища та зручного обміну файлами підійде класичне файлове сховище. У випадку з масштабними мультимедійними архівами чи складними резервними копіями доцільно звертатися до об’єктного формату. А для забезпечення стабільної роботи систем з високим навантаженням на читання і запис — застосовувати блочні рішення. Незалежно від вибору, всі вони представляють собою різні реалізації однієї концепції — хмарного сховища, що надає доступ до ресурсів у віддаленому середовищі з урахуванням специфічних потреб користувача.

Інтеграція хмарних сховищ у IT-інфраструктуру. Інтеграція хмарних сховищ у IT-інфраструктуру є невід’ємною складовою сучасних технологічних середовищ,

оскільки такі сховища не функціонують у відриві від інших компонентів системи. Хоча кількість типів сховищ обмежена, кожне з них має бути здатним до взаємодії з іншими елементами інфраструктури, об'єднуючись у єдине середовище. Відповідно, хмарні рішення впроваджуються не як окрема функціональність, а як розширення розподіленої IT-архітектури, що унеможливлює автономне існування більшості цифрових систем.

Інтеграція відбувається через стандартизовані інтерфейси API, які дозволяють використовувати сховища в рамках таких процесів [1-2]:

— резервне копіювання;

— масштабування ресурсів;

— підтримка високонавантажених сервісів.

Часто хмарні сховища використовуються як розширення локальних дата-центрів, формуючи гібридну хмару, що забезпечує додаткову гнучкість у випадках нестачі потужностей або потреби в підвищеній доступності даних. В інших ситуаціях вони можуть стати основою для розміщення даних у мікросервісній архітектурі, підтримуючи контейнери, модулі або специфічні сховища залежно від потреб застосунку (див. рис. 1.6).

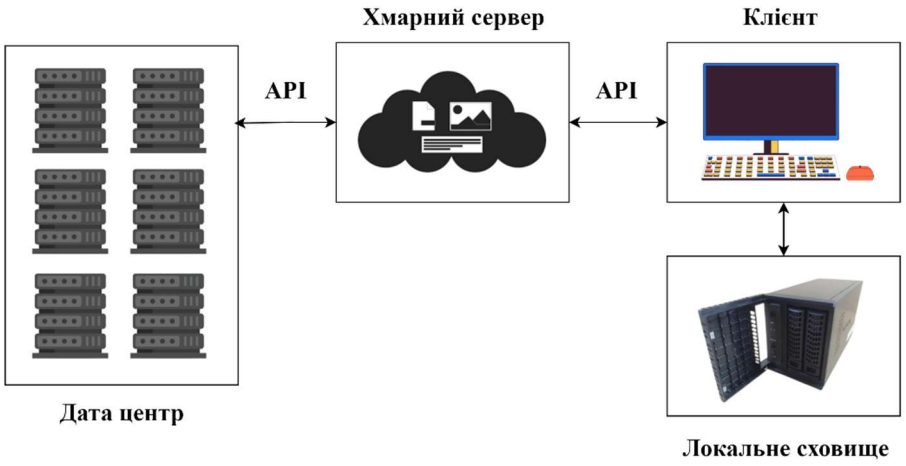


Рисунок 1.6 – Принцип інтеграції хмарного сховища через API

Завдяки такій інтеграції досягається не лише гнучкість і масштабованість, а й значне спрощення адміністрування. Наприклад, популярні платформи на зразок

Kubernetes мають вбудовану підтримку хмарних сховищ через Container Storage Interface (CSI) плагіни, що дозволяє автоматизувати процеси зберігання без потреби у ручному втручанні, що не лише знижує кількість помилок, а й підвищує стабільність та ефективність роботи всієї системи [1].

Зростання обсягів даних робить подібну інтеграцію критично необхідною. Уніфікована класифікація хмарних сховищ на об’єктні, блочні та файлові дозволяє чіткіше визначати оптимальні рішення для конкретних завдань. Усвідомлення їх функціональних особливостей дає змогу впроваджувати зберігання даних у IT-інфраструктуру максимально ефективно, посилюючи її надійність, керованість та здатність адаптуватися до зростаючих вимог.

1.2 Проблеми продуктивності у хмарних середовищах зберігання

У хмарних середовищах зберігання даних, попри значні переваги у масштабованості, доступності та гнучкості, питання продуктивності залишається одним із найскладніших і найважливіших. Навіть при ретельно спроєктованій архітектурі та налаштованій інфраструктурі, системи можуть демонструвати неочікувані збої або уповільнення, що пов’язано з низкою взаємозалежних технічних чинників. Проблеми часто виникають не через єдину причину, а як наслідок впливу кількох факторів одночасно, які охоплюють як фізичний рівень обладнання, так і логіку програмної взаємодії.

Фізичні характеристики дата-центрів, зокрема затримки у мережах, навантаження на сховища, типи дисків, обмеження пропускної здатності або навіть розподіл навантаження між кластерами, безпосередньо впливають на швидкість доступу до даних. До цього додаються аспекти програмного забезпечення — від архітектурних особливостей програм, що взаємодіють зі сховищем, до алгоритмів кешування, стиснення чи реплікації даних, що можуть створювати додаткове навантаження або затримки в обробці запитів [3].

У багатьох випадках виявлення причин зниження продуктивності потребує детального аналізу всієї ланки обробки даних — від користувацького запиту до відповіді системи — з урахуванням внутрішньої логіки роботи хмарної платформи, поточної конфігурації сховищ та рівня синхронізації між сервісами. Також варто враховувати специфіку типу сховища: об’єктні, блочні та файлові системи по-різному поводяться під навантаженням, особливо коли йдеться про паралельні звернення або обробку великих обсягів дрібних даних [1, 3-4].

З огляду на це, продуктивність не може розглядатися як статичний показник, а скоріше як динамічна характеристика, яка залежить від поєднання технічного стану інфраструктури, налаштувань середовища, типу навантаження та навіть часу доби. У середовищах з високими вимогами до швидкодії критично важливо забезпечити гнучку систему моніторингу й адаптивного масштабування, що дає змогу вчасно реагувати на коливання продуктивності та уникати втрат доступності або затримок у роботі сервісів.

Висока затримка (latency) доступу до даних. У структурі продуктивності хмарних систем зберігання даних затримка доступу до інформації виступає одним із найбільш чутливих показників, що безпосередньо впливає на стабільність та оперативність усіх операцій. У системах, де очікується миттєва реакція на запити або необхідна обробка значних обсягів транзакцій у реальному часі, затримка (latency) може стати стримувальним фактором, що значно знижує ефективність взаємодії з даними. Навіть незначне підвищення часу відгуку призводить до затримок у виконанні операцій, які в іншому випадку відбувалися б майже миттєво, що стає особливо помітним під час роботи з великою кількістю дрібних об’єктів [3-4].

Сукупний вплив таких затримок може мати різну природу. Часто джерелом проблем стає сама мережа:

— маршрутизація;

— буферизація пакетів;

— додаткові обробки.

З іншого боку, значна частина затримок виникає на рівні обробки запитів на стороні хмарної платформи:

— зчитування даних з дисків;

— формування черг;

— зберігання у розподілених об’єктних системах.

Усе це вимагає часу, який за високого навантаження або нестачі обчислювальних ресурсів (процесора, пам’яті, швидкодії дискових підсистем) починає зростати. У підсумку, навіть незначне збільшення часу доступу до кожного окремого елемента може перетворитися на критичний тягар, особливо якщо таких елементів — десятки або сотні тисяч (див. рис. 1.7) [3-4].

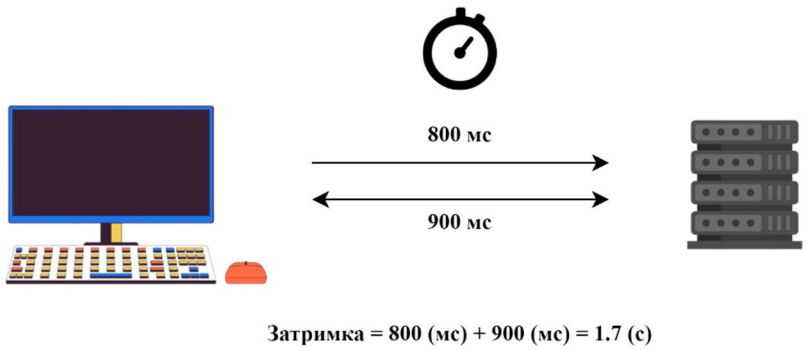


Рисунок 1.7 – Проблема високої затримки доступу до даних

Типовий приклад — спроба зчитування великої кількості малих файлів, які поодинці не становлять проблеми, але в сукупності, за умови високої латентності, спричиняють суттєве сповільнення. У таких випадках продуктивність падає не через фізичне перевантаження, а саме через затримки обробки кожного запиту окремо, що розтягує загальний час обробки і робить неможливим ефективне виконання задач, які потребують швидкої реакції.

Обмежена пропускна здатність (throughput). Окрім затримки, обмежена пропускна здатність (throughput) також є істотним бар’єром для досягнення високої продуктивності у хмарних середовищах зберігання. Цей параметр визначає, з якою швидкістю система здатна передавати значні обсяги інформації між клієнтом і хмарною платформою, і тісно пов’язаний з фізичними характеристиками

інфраструктури. Найбільш очевидним є вплив пропускної здатності мережі — якщо канал передачі даних застарілий або перевантажений, швидкість доступу до сховища буде значно знижена незалежно від того, наскільки потужним є сервер або оптимізована архітектура (див. рис. 1.8) [1, 3-4].

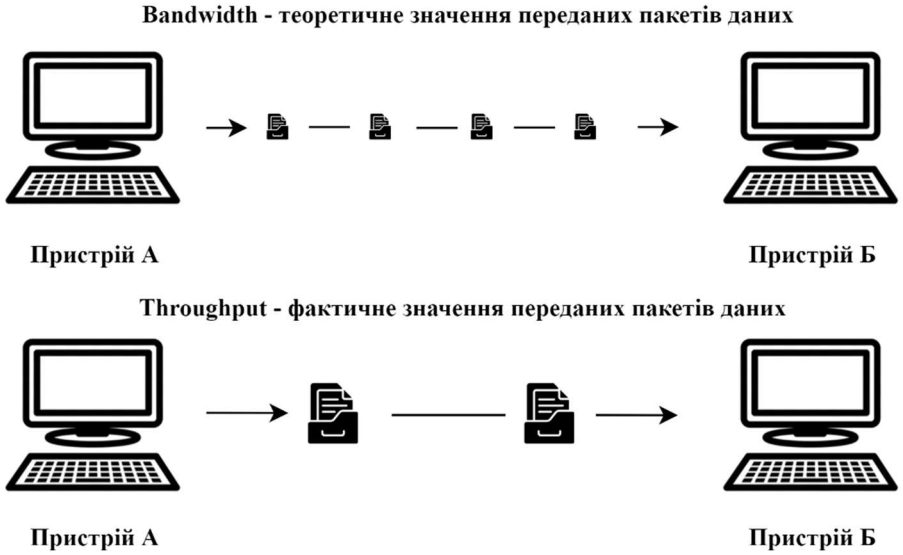


Рисунок 1.8 – Проблема обмеженої пропускної здатності у обміні даними

Стан мережевого обладнання, його відповідність сучасним стандартам (наприклад, 2,5 або 10 Гбіт/с замість традиційних 1 Гбіт/с), а також загальна топологія інфраструктури визначають потенційну ширину каналу для передавання даних. У старих системах із застарілою мережею навіть високопродуктивне сховище не забезпечить очікуваної швидкодії, тоді як у сучасних середовищах, побудованих із використанням швидкісних комутаторів і оптимізованої маршрутизації, той самий обсяг даних буде передано у рази швидше.

Однак на пропускну здатність впливають не лише фізичні параметри, а й програмні аспекти:

— тип доступу;

— обрані протоколи (HTTP, S3, NFS, SMB);

— алгоритми паралельної обробки запитів.

Наприклад, при зверненні до об’єктного сховища через S3 недостатня кількість паралельних потоків може призвести до того, що потенціал каналу не буде повністю використаний. У випадках, коли сервер обробляє запити послідовно або обмежено в кількості одночасних з’єднань, швидкість взаємодії з даними знижується пропорційно [1, 3-4].

Таке уповільнення особливо помітне під час передачі мультимедійних файлів, резервного копіювання або синхронізації великих масивів інформації з дата-центрами. У цих сценаріях критично важливо забезпечити не лише достатню пропускну здатність каналу, а й оптимальну архітектуру обробки даних із підтримкою масштабованої паралельності. Інакше навіть незначні обмеження будуть множитися у часі, створюючи ефект «вузького горла», що впливає на загальну продуктивність системи [3-4].

Нестабільність продуктивності (performance variability). Нестабільність продуктивності у хмарних сховищах є поширеним явищем, що проявляється у вигляді коливань швидкості обробки даних або затримок доступу до них у різні періоди часу. Причини такого ефекту безпосередньо пов’язані з природою віртуалізованих середовищ, де фізичні ресурси серверів динамічно розподіляються між великою кількістю віртуальних машин. У результаті кожна інстанція отримує не гарантований, а обмежений обсяг обчислювальної потужності відповідно до заданої конфігурації, що змінюється в залежності від загального навантаження на хост [1, 3].

До цього додається ефект спільного використання мережевих і дискових ресурсів, коли, наприклад, кілька користувачів звертаються до одного й того ж диска або сегмента сховища. У таких умовах продуктивність значно знижується, адже кожне паралельне звернення потребує власного часу обробки, що порушує передбачуваність результатів. Ще одним чинником є автоматичне масштабування та балансування навантаження, що має на меті підтримання стабільної роботи всієї інфраструктури, однак у критичних ситуаціях воно може розподіляти ресурси нерівномірно. Наприклад, процес, який потребує негайного завершення, отримує

недостатній пріоритет, у той час як менш важливі завдання можуть завершуватися швидше лише через алгоритмічні особливості планувальника (див. рис. 1.8) [3-4].

Такі коливання створюють труднощі не тільки в обслуговуванні інтерактивних запитів, а й у роботі автоматизованих сценаріїв:

— резервного копіювання;

— обміну даними з дата-центрами;

— обробки пакетної інформації.

Нестабільність, що виникає навіть на нетривалий час, може порушити цілісність бізнес-процесів або викликати затримки в логіці роботи систем, які залежать від очікуваної швидкості відповіді [3-4].

Навантаження на дискову підсистему. Проблеми продуктивності хмарних сховищ також тісно пов’язані з навантаженням на дискову підсистему. Вона часто стає вузьким місцем, особливо коли зростає кількість паралельних операцій введення-виведення (I/O), які генеруються великою кількістю одночасних запитів. В основі таких труднощів — обмежена кількість IOPS (операцій введення-виведення за секунду), які може забезпечити конкретний диск або масив (див. рис. 1.9) [3-4].

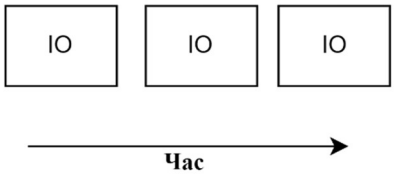


Рисунок 1.9 – Проблема навантаженості на дискову систему (IOPS)

Якщо у системі не передбачено використання SSD для кешування або немає відповідної оптимізації, доступ до даних сповільнюється в рази. Твердотілі накопичувачі зазвичай використовуються для буферизації часто запитуваних даних, а за їх відсутності відбувається безпосередній доступ до повільнішого сховища, що значно впливає на час обробки запитів. Особливо це помітно в умовах

великої кількості дрібних об’єктів — кожен такий файл або запис потребує окремої обробки, а в сукупності це створює значне навантаження.

У випадках, коли мова йде про системи аналітики, медіаобробки чи логування, що працюють у режимі реального часу, саме продуктивність дискової підсистеми визначає, наскільки швидко та ефективно система зможе виконати поставлені завдання. Недостатня її пропускна здатність або неправильна конфігурація можуть звести нанівець переваги всієї хмарної інфраструктури, зокрема при масштабних копіюваннях, обміні даними або обслуговуванні клієнтських запитів [1, 3-4].

Високе навантаження на процесор (CPU). Високе навантаження на процесор часто є одним із критичних факторів, що впливають на загальну продуктивність хмарного сховища. Обробка запитів, шифрування, стиснення, а також інші операції з файлами виконуються безпосередньо на стороні серверів, де розгорнуто сховище. За умов недостатньої обчислювальної потужності збільшується час реакції на запити, що, у свою чергу, призводить до уповільнення роботи сервісу або навіть його часткової відмови. Зростає ризик виникнення тайм-аутів, а також збоїв у роботі окремих компонентів через брак доступних ресурсів. Усі ці процеси взаємопов’язані, й навіть незначна нестача процесорного часу може спричинити каскадну деградацію інших сервісів (див. рис. 1.10).

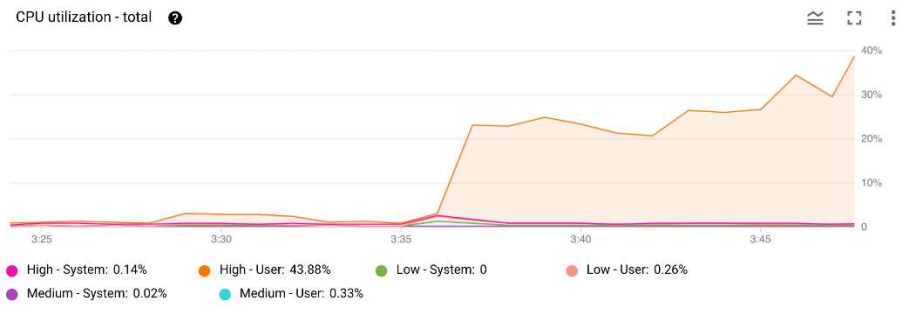


Рисунок 1.10 – Проблема високого навантаження на процесор (CPU)

Особливо це ускладнює автоматичне масштабування, коли навантаження зростає, а можливість залучення додаткових ресурсів обмежена. При повному вичерпанні доступних потужностей стає неможливим навіть базове

обслуговування поточного трафіку. Системи, які покладаються на процесорну обробку — від файлових механізмів зберігання до проксі-серверів і сервісів типу reverse proxy, — втрачають стабільність і можуть працювати з перервами або взагалі припинити відповідати. За таких умов критично важливим стає постійний моніторинг процесорного навантаження, зокрема оцінка внеску кожного окремого процесу у загальне використання CPU, особливо тих, які оперують з масивами даних чи обробляють запити до сховища [1, 3-4].

Навантаження на мережеві інтерфейси. Іншим потенційним обмеженням продуктивності може бути перевантаження мережевих інтерфейсів. У сценаріях з високою інтенсивністю обміну даними, особливо у гібридних або розподілених системах, здатність мережевої інфраструктури ефективно обробляти запити стає визначальною. Велика кількість одночасних підключень або надмірний трафік можуть вивести з ладу навіть добре налаштовані канали, якщо вони не мають достатнього запасу пропускної здатності. У таких випадках перевантаження мережевих інтерфейсів або маршрутизаторів викликає сповільнення, втрати пакетів або повну недоступність сервісу (див. рис. 1.11) [3-4].

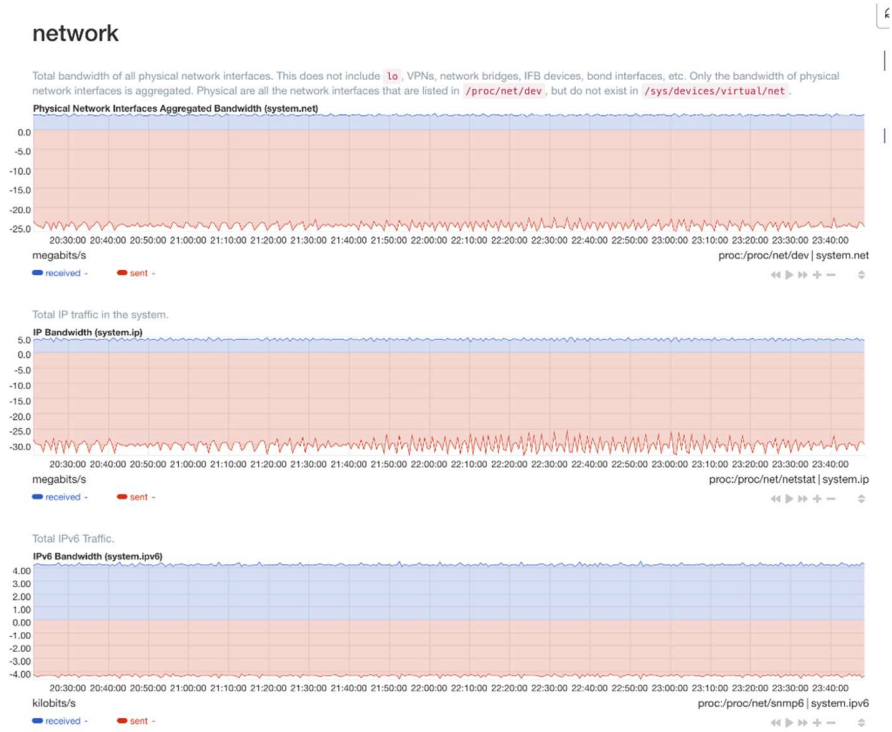


Рисунок 1.11 – Проблема високого навантаження на мережеві інтерфейси

Нерівномірне балансування навантаження між вузлами сховища лише поглиблює ситуацію: окремі вузли можуть опинитися в стані перенавантаження, у той час як інші простоюють. Для уникнення подібних вузьких місць у мережевому середовищі застосовують:

— агрегування каналів;

— розподіл трафіку за VLAN;

— окремі високошвидкісні (наприклад, 10-гігабітні) мережі.

Недостатній моніторинг і реакція на навантаження. Недостатній моніторинг і слабка реакція на зміну навантаження стають одним із першоджерел системних збоїв, оскільки без вчасного виявлення перевантаження будь-якого компонента інфраструктури неможливо запобігти погіршенню продуктивності або зупинці сервісу. При відсутності комплексної системи спостереження, такої як Prometheus у поєднанні з Grafana чи подібними засобами візуалізації та аналізу, адміністратор не має змоги виявити перевищення лімітів споживання ресурсів, визначити, де саме утворюються вузькі місця, і як змінюється навантаження протягом доби або тижня. Без такої аналітики відсутня не лише можливість попередити проблеми, а й ускладнюється їх своєчасне вирішення.

Нестача даних для діагностики позбавляє змоги оперативно реагувати навіть на незначні збої, які в іншому разі могли б бути усунуті за кілька хвилин. Якщо не здійснюється регулярний моніторинг стану компонентів, зростає ризик накопичення прихованих помилок, що з часом призводять до збоїв і потребують значно більше зусиль і часу для усунення. У корпоративному середовищі це означає не лише затрати часу, а й пряму втрату коштів. Відсутність чітко налагодженого контролю над ресурсами унеможливлює профілактику та оптимізацію, а отже — сприяє повторенню помилок у майбутньому [3-4].

Загальна природа таких збоїв має системний характер, адже навіть у випадках, коли сховище здається прозорим і «безмежним» для кінцевого користувача, на стороні провайдера завжди присутні спеціалісти, які слідкують за працездатністю інфраструктури, контролюють навантаження, виконують регулярне обслуговування та оновлення. Коли йдеться про об’єктні або гібридні

сховища, де користувач не взаємодіє безпосередньо з інфраструктурою, всі ці механізми залишаються критично необхідними саме на рівні провайдера, який забезпечує стабільну роботу сервісу [2-4].

У власних розгортаннях без постійного спостереження за навантаженням та збиранням журналів логів не вдасться забезпечити ні стабільність, ні масштабованість системи. Наявність журналів подій у поєднанні з візуалізацією історії навантаження дозволяє швидко визначити джерело проблеми та локалізувати її, що, у свою чергу, скорочує час на реакцію та мінімізує наслідки для користувачів. Тому в усіх типах сховищ — незалежно від архітектури — впровадження спеціалізованих інструментів моніторингу є не лише доцільним, а й необхідним кроком, що дозволяє підтримувати працездатність системи на належному рівні, виявляти потенційні проблеми до їх критичного прояву та оперативно реагувати на будь-які зміни в навантаженні [1-4].

1.3 Метрики продуктивності та методи їх вимірювання

Продуктивність хмарних систем зберігання визначається здатністю інфраструктури обробляти запити стабільно, швидко та з можливістю масштабування, тому коректна оцінка таких систем вимагає застосування низки технічних інструментів. Для цього використовуються метрики, які кількісно фіксують навантаження, час відгуку, ефективність кешування, рівень використання ресурсів та інші характеристики, що дає змогу сформувати повноцінну аналітичну модель поточного стану системи. Відсутність таких даних робить неможливим розуміння реального стану сховища, що, у свою чергу, ускладнює забезпечення надійності, продуктивності й масштабованості всієї інфраструктури. Без регулярного вимірювання продуктивності виникає ризик запізнілого реагування на перевантаження, що знижує ефективність управління ресурсами та унеможливлює своєчасне масштабування в міру зростання навантаження.

Інфраструктура, яка не фіксує продуктивність у реальному часі, позбавлена механізму оперативного реагування: будь-яке розширення або зміна архітектури відбувається навмання, без розуміння слабких місць. Такі ситуації згодом призводять до повторного виникнення однакових проблем через відсутність історичних даних і метрик. Натомість впровадження механізмів вимірювання продуктивності дає змогу адаптувати інфраструктуру під навантаження ще до того, як вони почнуть критично впливати на роботу системи.

Ключові метрики продуктивності хмарного сховища. Оцінювання продуктивності не обмежується однією змінною, оскільки для комплексного аналізу потрібна система взаємозалежних показників, які разом формують уявлення про здатність сховища виконувати завдання різного типу. Однією з таких метрик є IOPS (Input/Output Operations Per Second) — кількість операцій читання і запису, яку дискова система здатна виконати за секунду. Цей показник є критично важливим для інтенсивно навантажених систем, таких як OLTP-бази даних, веб-додатки з високим рівнем запитів та віртуалізовані середовища (див. рис. 1.12) [5-6].

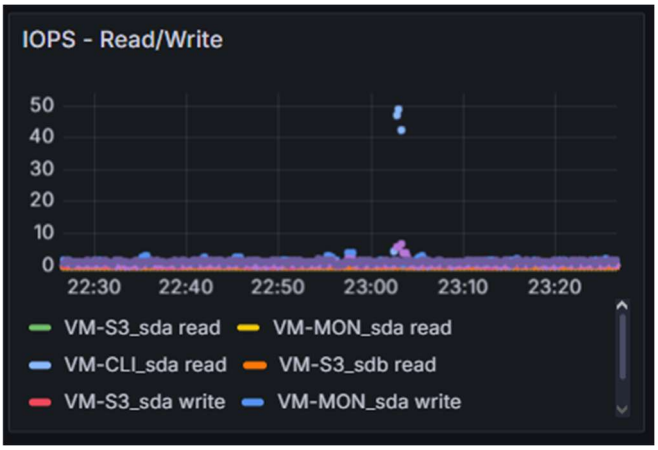


Рисунок 1.12 – Метрики кількості операцій читання/запису (IOPS) (візуалізація метрик через Grafana)

Ще одним значущим показником виступає затримка (latency), тобто час від моменту надсилання запиту до отримання відповіді. Вимірюється вона зазвичай у мілісекундах і поділяється на затримку читання та запису. Чим нижчі значення, тим

кращий результат, особливо в сценаріях реального часу або транзакційних операціях [5-6] (див. рис. 1.13).

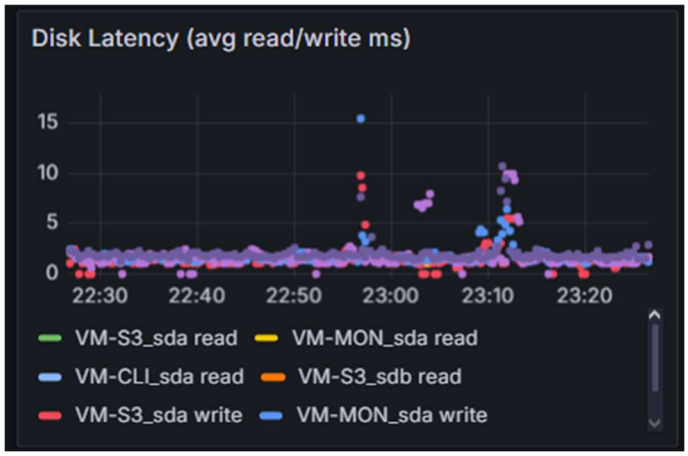


Рисунок 1.13 – Метрики затримки (latency)
(візуалізація метрик через Grafana)

Не менш важливим є показник пропускної здатності (throughput), що визначає обсяг переданих даних за одиницю часу, зазвичай у мегабайтах або гігабайтах на секунду. Така метрика є визначальною для роботи з великими обсягами даних — резервними копіями, потоковим відео чи масивними архівами (див. рис. 1.14) [5-6].

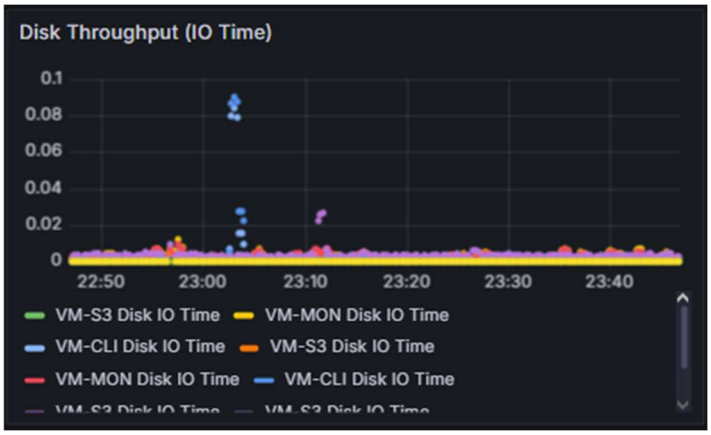


Рисунок 1.14 – Метрики показників пропускної здатності (throughput)
(візуалізація метрик через Grafana)

До основних метрик також належать показники використання ресурсів, зокрема CPU та оперативної пам’яті (CPU & memory utilization). Вони відображають поточне споживання обчислювальних потужностей і критичні для сервісів, що здійснюють шифрування, стиснення або обробку даних у реальному часі. Їх надмірне споживання або нестабільність можуть вказувати на потребу масштабування або оптимізації процесів (див. рис. 1.15) [5-6].

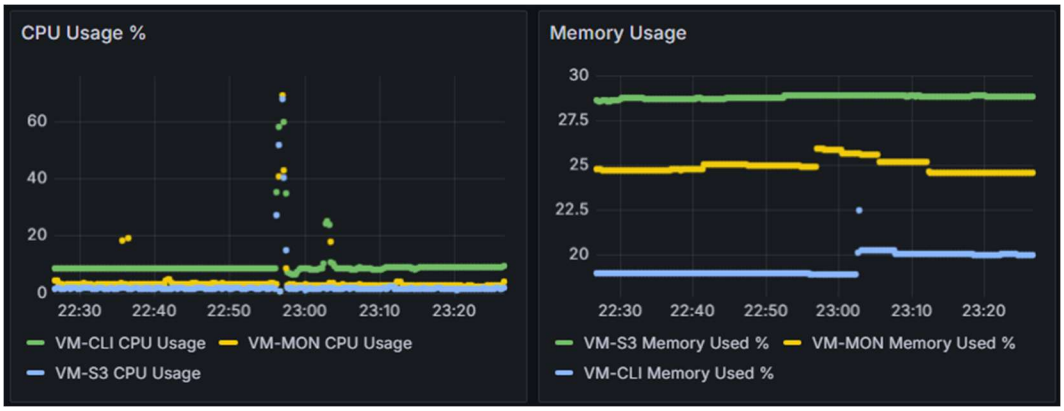


Рисунок 1.15 – Метрики використання ресурсів (CPU, Memory)
(візуалізація метрик через Grafana)

Ще одним важливим аспектом є частота помилок, яка фіксує кількість невдалих запитів або збоїв — як-от внутрішні помилки (наприклад, 500 Internal Server Error), так і відмови на рівні з’єднання. Моніторинг цих даних дозволяє оперативно виявляти та локалізовувати проблеми, своєчасно вживати заходів без суттєвого впливу на продуктивність (див. рис. 1.16) [5-6].

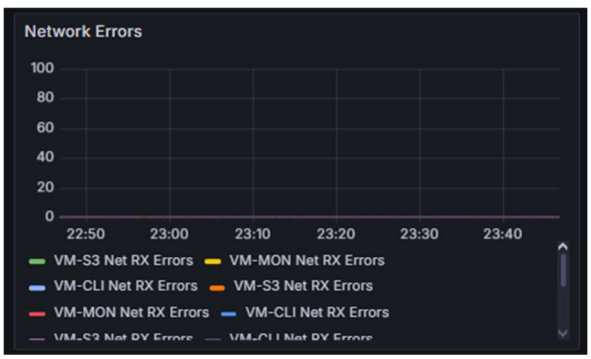


Рисунок 1.16 – Метрики частоти помилок (Errors)
(візуалізація метрик через Grafana)

У підсумку, лише поєднання метрик дає можливість отримати цілісну картину стану інфраструктури, забезпечити її гнучке масштабування та стабільну роботу навіть в умовах змінного навантаження.

Методи збору та вимірювання метрик. Методи збору та вимірювання метрик у хмарних системах охоплюють декілька підходів, кожен із яких забезпечує доступ до критичних показників продуктивності інфраструктури. Залежно від архітектури, рівня доступу до операційної системи та вимог до точності й масштабованості, збір даних може здійснюватися:

- з ядра операційної системи;
- через інстальовані агенти моніторингу;
- за допомогою API.

Отримання системної інформації через файлові інтерфейси /proc і /sys поширене насамперед у Linux-середовищі, де велика частина інформації про стан системи доступна через вбудовані текстові файли. Наприклад, файл "proc/diskstats" містить детальну статистику щодо роботи дисків, "proc/meminfo" — дані про використання оперативної пам’яті, а "proc/stat" — інформацію про завантаження процесора (див. рис. 1.17) [7, 27].

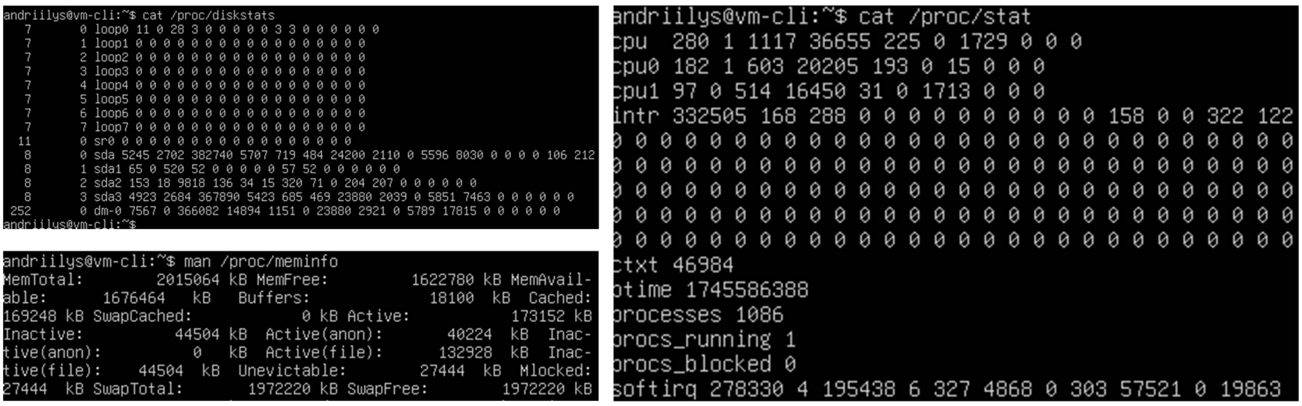


Рисунок 1.17 – Метод збору та вимірювання метрик
через файлові інтерфейси Linux

Такий метод дозволяє отримати точні метрики за мінімальної підтримки програмного забезпечення, однак вимагає розуміння структури системних файлів і

вміння інтерпретувати сирі значення. У більшості випадків взаємодія з такими джерелами даних відбувається через командний рядок, і результати потребують подальшої обробки для аналізу [7, 27].

Для спрощення процесу та централізації збору метрик у розподілених системах застосовуються агенти моніторингу. Їх встановлюють безпосередньо на вузли інфраструктури, де вони збирають та передають системні дані до серверів моніторингу. Одним із найпоширеніших прикладів є Node Exporter, який працює у зв'язці з Prometheus і забезпечує збирання базових метрик, таких як:

— завантаження CPU;

— обсяг використаної пам'яті;

— активність дисків;

— мережевий трафік.

Іншим прикладом є Telegraf, агент від InfluxData, що підтримує велику кількість плагінів для збору, обробки та візуалізації інформації з багатьох джерел. Ще одним є Netdata, який орієнтований на моніторинг у реальному часі з багатою графічною візуалізацією. Агенти можуть автоматично опрацьовувати системні дані з proc і sys, нормалізовувати їх і передавати через різні протоколи — HTTP, gRPC або власні специфічні формати — до зовнішніх платформ візуалізації та зберігання (див. рис. 1.18) [7-8, 11].

```
# HELP apt_autoremove_pending Apt packages pending autoremoval.
# TYPE apt_autoremove_pending gauge
apt_autoremove_pending 0
# HELP apt_package_cache_timestamp_seconds Apt update last run time.
# TYPE apt_package_cache_timestamp_seconds gauge
apt_package_cache_timestamp_seconds 1.745589865764801e+09
# HELP apt_upgrades_held Apt packages pending updates but held back.
# TYPE apt_upgrades_held gauge
apt_upgrades_held{arch="amd64",origin=". stable:stable/stable"} 1
apt_upgrades_held{arch="amd64",origin="Ubuntu:noble/noble-updates"} 5
# HELP go_gc_duration_seconds A summary of the pause duration of garbage collection cycles.
# TYPE go_gc_duration_seconds summary
go_gc_duration_seconds{quantile="0"} 2.4987e-05
go_gc_duration_seconds{quantile="0.25"} 3.4965e-05
go_gc_duration_seconds{quantile="0.5"} 4.0225e-05
go_gc_duration_seconds{quantile="0.75"} 5.1336e-05
go_gc_duration_seconds{quantile="1"} 0.000473331
go_gc_duration_seconds_sum 0.473117224
go_gc_duration_seconds_count 7830
```

Рисунок 1.18 – Метод збору та вимірювання метрик
через агента Node Exporter (від Prometheus)

Третій підхід базується на використанні API, які надають сучасні хмарні сховища. Наприклад, платформи на зразок MinIO, Amazon S3 або Wasabi дозволяють отримувати доступ до детальної статистики роботи сховища через REST або gRPC API. Дані, що надаються такими інтерфейсами включають:

— кількість оброблених запитів;

— час відповіді;

— розподіл навантаження між вузлами;

— кількість збоїв;

— специфічні помилки.

Така модель взаємодії спрощує інтеграцію з популярними сторонніми системами моніторингу — Grafana, Zabbix, Datadog, які дозволяють будувати наочні дашборди та здійснювати оперативний аналіз поведінки хмарної системи. Завдяки цьому забезпечується гнучке масштабування спостереження за інфраструктурою, а також своєчасне виявлення аномалій чи перевантажень у роботі сервісів (див. рис. 1.19) [7, 9-10, 15].

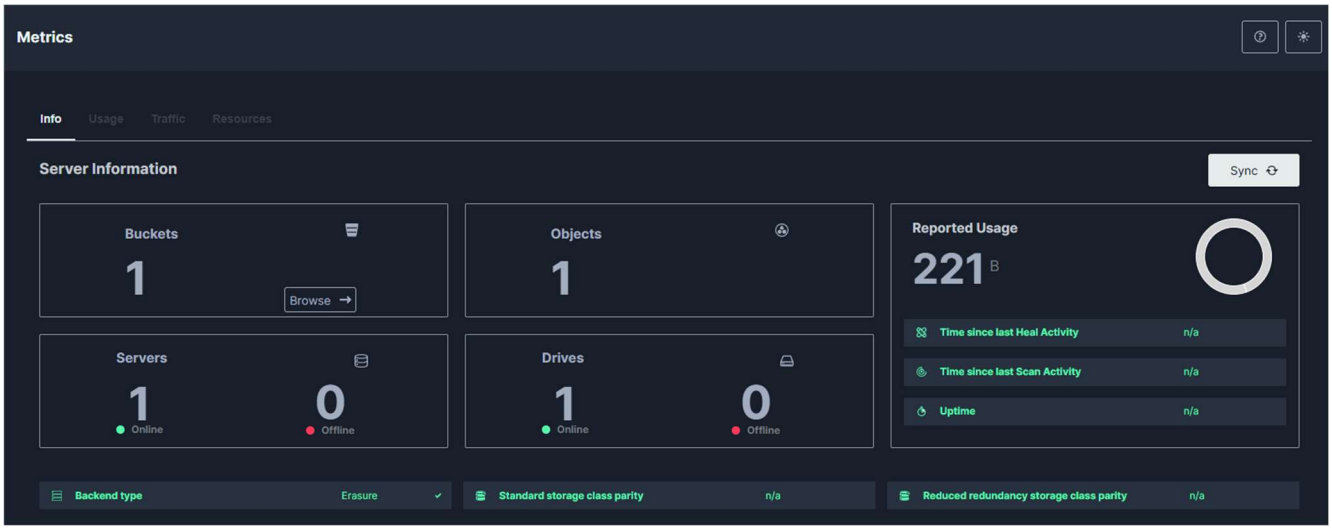


Рисунок 1.19 – Метод збору та вимірювання метрик через API (від MinIO)

Приклади систем для збору та аналізу метрик. Серед поширених рішень для збору та аналізу метрик можна виділити низку платформ, що забезпечують

централізовану обробку та зберігання даних про стан систем. Найбільш помітне місце серед них займає Prometheus — система моніторингу, орієнтована на часові ряди, яка реалізує власну модель збору даних. Завдяки підтримці мови запитів PromQL, системі алертів та інтеграції з Grafana, Prometheus часто використовується в архітектурах, де важливо мати гнучкий та незалежний підхід до збору метрик. Зберігання здійснюється локально у спеціалізованому форматі без потреби у зовнішній базі даних, що спрощує розгортання та експлуатацію, зберігаючи при цьому високу точність і контроль над даними (див. рис. 1.20) [7, 9-10].

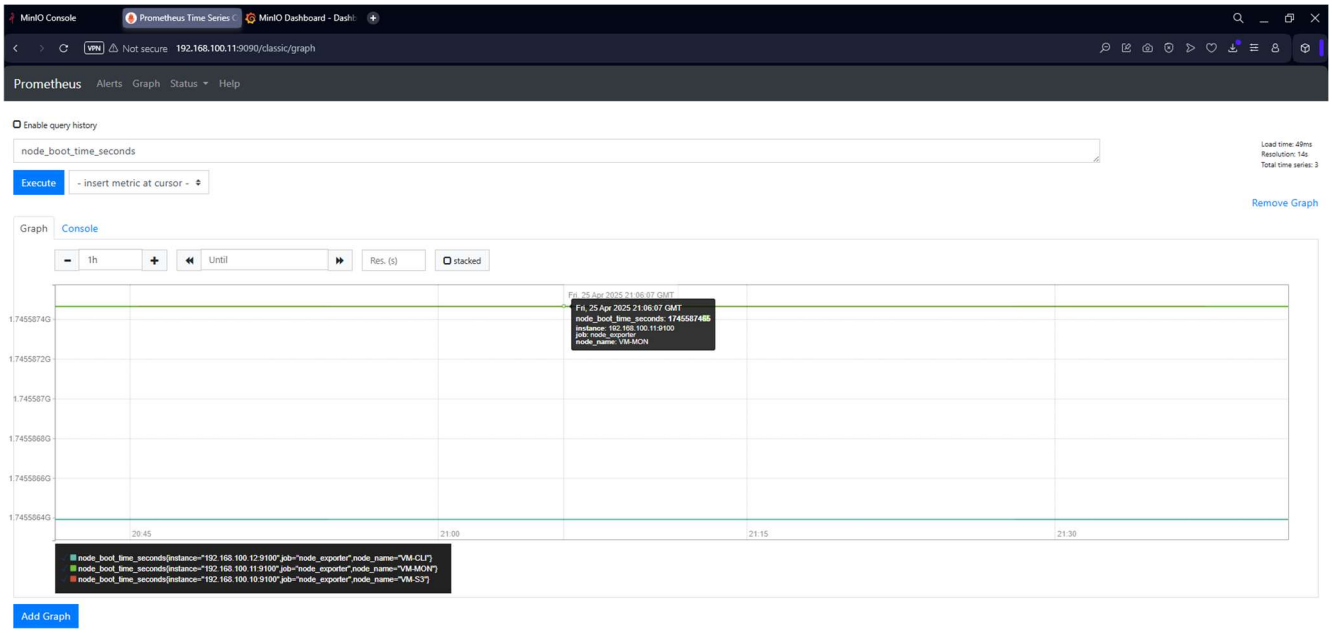


Рисунок 1.20 – Приклад збору та вимірювання метрик через Prometheus (за допомогою агента Node Exporter)

Водночас Grafana не займається збором метрик, але виступає потужним інструментом їх візуалізації. Платформа дозволяє підключення до джерел даних на кшталт Prometheus, InfluxDB, Elasticsearch, Azure Monitor та інших, забезпечуючи гнучку побудову:

— дашбордів;

— шаблонів сповіщень;

— шаблонів графіків.

Саме тому Grafana часто використовується у зв’язці з Prometheus як інтерфейс для зручного аналізу стану систем у реальному часі (див. рис. 1.21) [7-9].

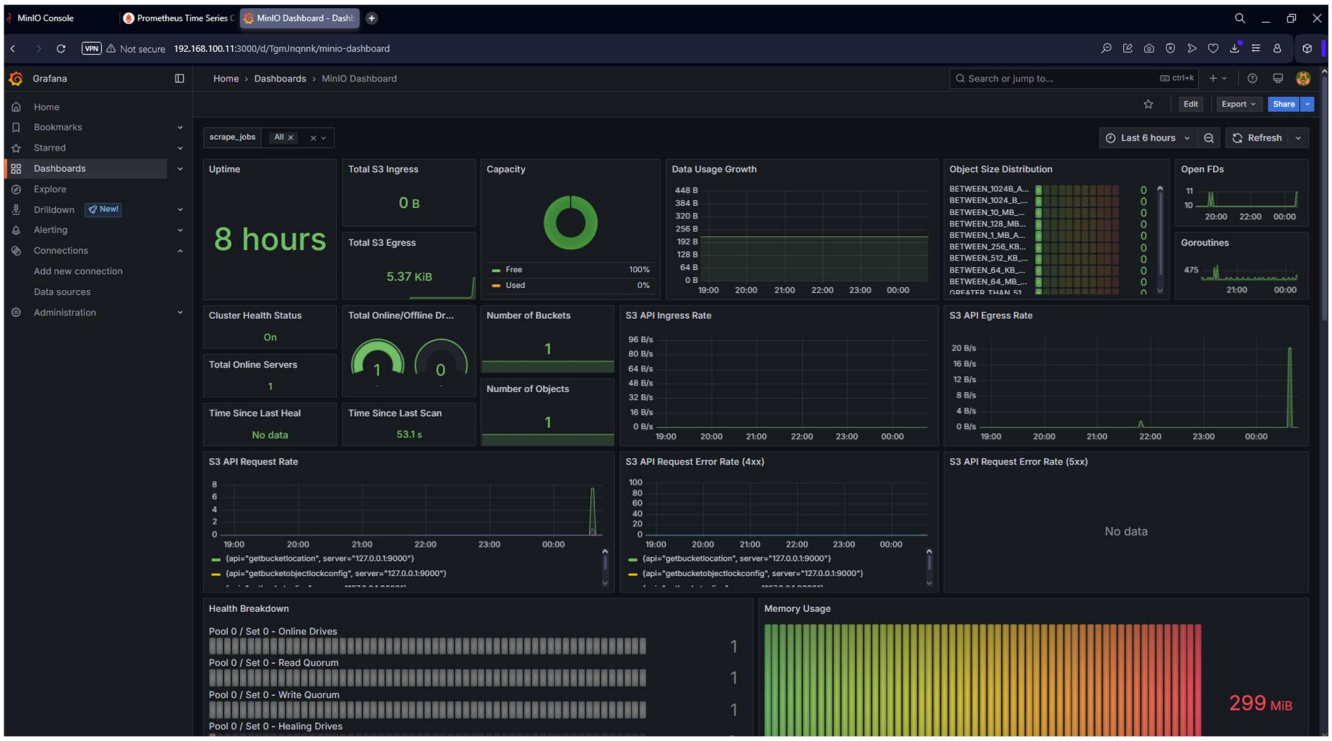


Рисунок 1.21 – Приклад збору та вимірювання метрик через Grafana (візуалізація зібраних метрик від Prometheus)

Окрім відкритих рішень, існують комерційні та гібридні системи, наприклад, Zabbix, Nagios, Datadog, які не лише збирають метрики, але й аналізують логи, виконують сповіщення, проводять діагностику. Подібні платформи широко застосовуються в корпоративному середовищі, де важлива глибока інтеграція з інфраструктурою та розширені функції. Проте у контексті побудови компактної або початкової системи моніторингу саме Prometheus та Grafana становлять оптимальну комбінацію: перша платформа відповідає за збір часових рядів, а друга — за візуалізацію отриманих даних для оперативного реагування на зміни в роботі сервісів [7-9].

Інтерпретація та застосування метрик на практиці. Однак зібрати метрики — лише перший крок, адже подальше управління системою потребує їх коректної інтерпретації. Наприклад, поєднання низького IOPS із високою затримкою вказує

на перевантаження дискової підсистеми або проблеми з кешем. Високе використання процесора, у свою чергу, може бути наслідком неефективного оброблення запитів чи неоптимального шифрування. Низький показник cache hit ratio свідчить про невдало реалізовану політику кешування або відсутність кешу взагалі, тоді як часті помилки 5xx зазвичай пов’язані з недоступністю сервісу або апаратними збоями [5-7].

Правильне розуміння таких сигналів дозволяє не лише оцінити поточний стан системи, але й вчасно вжити заходів — наприклад, перейти на SSD, виконати балансування навантаження або змінити конфігурацію програмного забезпечення. Постійне спостереження за продуктивністю стає необхідною умовою підтримки стабільної роботи хмарних сховищ і масштабованих сервісів, де кожне відхилення може стати причиною втрат або перебоїв у роботі.

Ефективне використання метрик потребує їх адаптації до типу навантаження та особливостей архітектури. Базові показники — такі як системне навантаження, latency, error rate, throughput — у поєднанні з розумною автоматизацією аналізу дають змогу створити динамічну систему, здатну масштабуватись, уникати критичних навантажень і швидко реагувати на зміни. Саме це перетворює збір та інтерпретацію метрик із формальної процедури на основу стабільного керування сучасною інфраструктурою [5-7].

1.4 Підходи до моніторингу продуктивності IT-систем

Моніторинг залишається одним із найважливіших елементів сучасної IT-інфраструктури, оскільки забезпечує виявлення аномалій, своєчасне реагування на збої, контроль ресурсів, аналіз навантаження та підтримання рівня обслуговування відповідно до SLA. У контексті хмарних сховищ він стає основним інструментом для оптимізації:

— продуктивності систем зберігання;

— рівномірного розподілу навантаження;

— попередження аварійних ситуацій.

Залежно від специфіки системи, моніторинг може реалізовуватись через встановлення спеціалізованих агентів або шляхом опитування відкритих інтерфейсів і протоколів — кожен з цих підходів має власні особливості, переваги та обмеження, які слід враховувати під час побудови архітектури контролю стану інфраструктури [5-7].

Агентський (agent-based) моніторинг. Агентський підхід до моніторингу передбачає встановлення на вузли системи спеціального програмного модуля — агента, який у вигляді системного процесу або служби періодично зчитує параметри апаратних і програмних ресурсів, формує метрики та надсилає їх до основної моніторингової системи. Завдяки безпосередній інтеграції з операційною системою, такий метод дозволяє отримувати розширену інформацію про стан CPU, дискових підсистем, оперативної пам’яті, температури обладнання, логів та інших глибинних показників, що забезпечує повніше розуміння реального стану об’єкта спостереження. Крім того, агентський підхід часто дозволяє кастомізувати збір даних, створюючи власні метрики відповідно до специфіки конкретного проєкту, що забезпечує додаткову гнучкість та адаптацію до нестандартних сценаріїв (див. рис. 1.22) [7, 11].

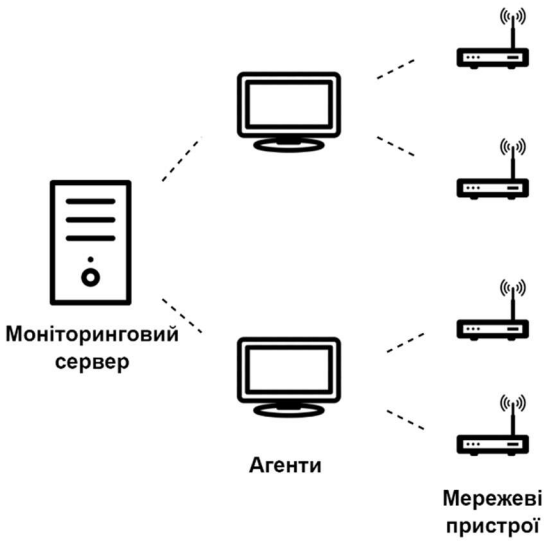


Рисунок 1.22 – Принцип роботи агентного моніторингу

До важливих переваг цього методу належить мінімальна затримка при збиранні інформації, оскільки передача метрик відбувається майже в реальному часі. Разом з тим, одним із суттєвих недоліків є потреба в інсталяції та обслуговуванні агента на кожному окремому вузлі, що створює додаткове навантаження на систему та вимагає адміністративного супроводу, що може стати критичним фактором у середовищах із великою кількістю вузлів або обмеженими апаратними ресурсами [11].

Прикладами реалізації агентського моніторингу є Node Exporter — агент для Prometheus, орієнтований на Linux-системи, який надає доступ до широкого спектра базових системних метрик. Аналогічно, Zabbix Agent використовується для збору даних у середовищі Zabbix, а Netdata забезпечує глибоку деталізацію внутрішніх процесів з мінімальними затримками, що робить його придатним для задач низькорівневого моніторингу. Усі ці інструменти дозволяють не лише відстежувати загальний стан систем, а й впроваджувати специфічні, нестандартні метрики, адаптовані до потреб конкретного проєкту [8, 11].

Безагентний (agentless) моніторинг. Безагентний підхід до моніторингу передбачає зчитування метрик без встановлення додаткового програмного забезпечення на хості, а отримання даних здійснюється дистанційно через стандартні протоколи, такі як SNMP, ICMP, HTTP або HTTPS із шифруванням. Для цього необхідно забезпечити мережевий доступ до віддалених вузлів, звідки й виконуються запити. Такий варіант особливо доцільний у випадках, коли немає змоги або доцільності встановлювати агент, зокрема на вбудованих пристроях, закритих платформах або сторонньому обладнанні, де доступ до внутрішньої системи обмежений (див. рис. 1.23) [7, 12-13].

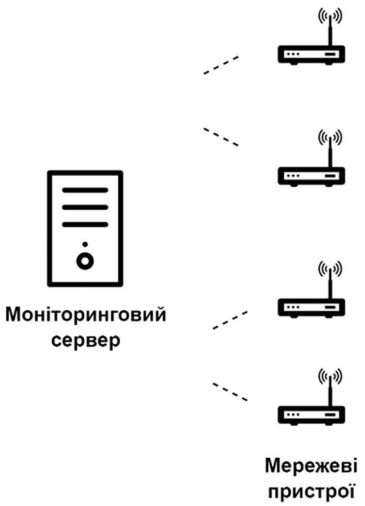


Рисунок 1.23 – Принцип роботи безагентного моніторингу

Перевагою такого підходу є спрощене розгортання, адже відсутня потреба в локальному доступі до кожного вузла для інсталяції агентів, що істотно знижує вимоги до обслуговування та пришвидшує початкову інтеграцію в інфраструктуру, що полегшує контроль стану пристроїв у розподілених середовищах, особливо за умови великої кількості об’єктів, які важко адмініструвати індивідуально [12].

Разом із тим, такий тип моніторингу має і свої обмеження. Перш за все, віддалений доступ через стандартні протоколи часто дозволяє отримати лише обмежений набір метрик, що не завжди забезпечує повноцінну картину внутрішнього стану системи. Також характерною особливістю є вища затримка при зборі даних у порівнянні з агентським підходом, а масштабування на великих інфраструктурах може вимагати додаткових рішень для балансування запитів і збереження стабільності. У межах такого підходу Zabbix, наприклад, підтримує роботу через SSH, SNMP або IPMI, дозволяючи інтегрувати обладнання без встановлення агентів, зокрема мережеву інфраструктуру, NAS або інші пристрої. У Prometheus подібна логіка реалізується через можливість надсилання метрик за допомогою HTTP-endpoint, якщо додаток самостійно публікує дані без потреби в окремому експортері [12-13].

Моніторинг через SNMP (Simple Network Management Protocol). Одним із найпоширеніших прикладів безагентного моніторингу є використання протоколу

SNMP (Simple Network Management Protocol), який традиційно застосовується для спостереження за мережевими пристроями. У структурі цього протоколу всі метрики представляються у вигляді об’єктів — Object Identifiers (OID), що формують ієрархію показників та можуть відображати як стандартні параметри, так і кастомізовані дані залежно від обладнання (див. рис. 1.24) [7, 12-13].

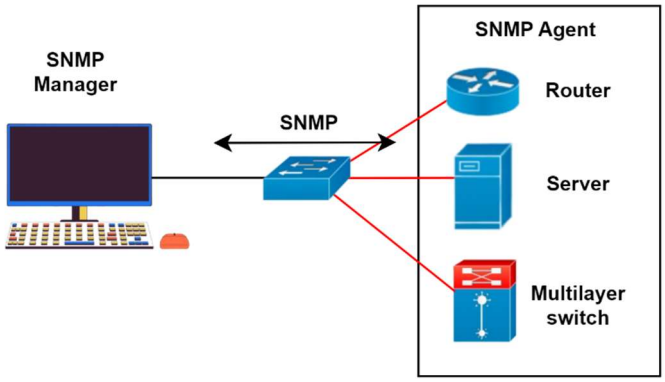


Рисунок 1.24 – Принцип роботи безагентного моніторингу через SNMP

Поширеність SNMP пояснюється його тривалою історією, а також тим, що підтримка цього протоколу реалізована у більшості мережевих пристроїв, таких як маршрутизатори, комутатори, принтери, системи зберігання. Його застосування не потребує інсталяції додаткових програмних модулів на самих пристроях, що робить його практичним для швидкого розгортання у гетерогенному середовищі. Протокол дозволяє як періодичне зчитування метрик, так і роботу за подієвою моделлю, коли дані надсилаються лише у разі зміни параметрів або виникнення події [13].

Однак SNMP має і низку характерних недоліків. Зокрема, робота з OID і MIB-таблицями потребує певної кваліфікації, особливо якщо йдеться про використання SNMPv2 або SNMPv3, де з’являється складність у налаштуванні автентифікації та шифрування. Попри свою функціональність, SNMP зазвичай не надає високорівневих метрик для продуктивності, що обмежує можливості аналітики, особливо у великих та динамічних середовищах. Саме через свою архітектуру він рідше використовується для збору даних з прикладного рівня [7, 13].

У контексті сучасних моніторингових рішень Zabbix має вбудовану підтримку SNMP, що дозволяє активно використовувати його для контролю інфраструктурних вузлів без залучення агентів. Prometheus також передбачає інтеграцію з SNMP через snmp_exporter, дозволяючи збирати дані з широкого спектра пристроїв за допомогою цього класичного протоколу. Незважаючи на певні обмеження, SNMP залишається одним із найбільш універсальних і сумісних механізмів безагентного моніторингу [7-8, 12-13].

Моніторинг через REST API. Моніторинг через REST API набув значного поширення серед сучасних хмарних платформ, контейнеризованих середовищ і систем зберігання, які дедалі частіше реалізують доступ до статистичних даних та метрик саме через RESTful API. За допомогою таких інтерфейсів надається можливість отримувати інформацію про стан сервісу, навантаження, обсяг трафіку, кількість запитів, продуктивність, затримки, помилки та інші важливі параметри без необхідності прямого втручання в інфраструктуру або інсталяції додаткових агентів. Такий підхід широко використовується у випадку з контейнерними платформами, як-от Docker або Kubernetes, де кожен вузол або сервіс самостійно публікує необхідні метрики через HTTP-endpoint (див. рис. 1.25) [7, 14-15].

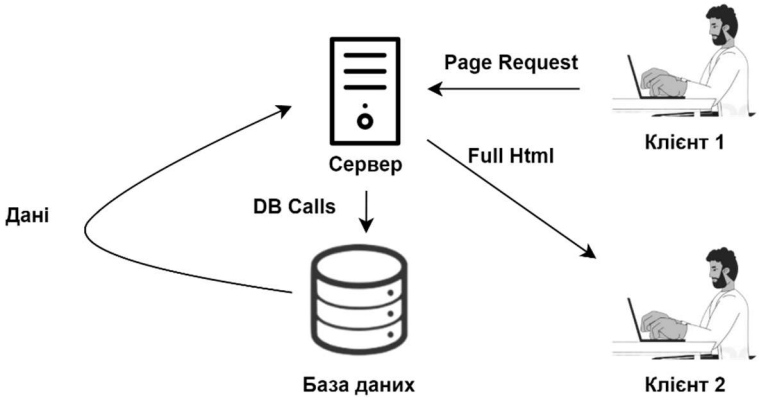


Рисунок 1.25 – Принцип роботи безагентного моніторингу через REST API

Гнучкість REST API дозволяє формувати запити із точним переліком необхідних параметрів, виключаючи надлишкові дані, що сприяє зменшенню навантаження на системи моніторингу й знижує обсяг трафіку. Завдяки можливості

вибіркового збору інформації підвищується ефективність запитів, що особливо важливо при моніторингу масштабованих середовищ з великою кількістю елементів. Відсутність потреби у встановленні клієнтського ПЗ на вузли мінімізує втручання в середовище, зменшує ризики сумісності та спрощує технічну підтримку [14].

Ще однією важливою перевагою є високий ступінь інтеграції з зовнішніми інструментами, включно зі скриптами власної розробки або сторонніми системами аналітики. REST API легко поєднується з інфраструктурними рішеннями, такими як Prometheus або Grafana, а також може бути основою для побудови індивідуальних рішень з урахуванням специфіки архітектури [7-9, 14].

Водночас, використання REST API має і низку обмежень. Запити до більшості сучасних API підпорядковуються правилам rate limiting, що обмежує частоту звернень і може вплинути на актуальність отриманих метрик, особливо у високонавантажених середовищах. Для доступу до інтерфейсу часто необхідна авторизація через токени або ключі доступу, що потребує окремого менеджменту й створює додаткові вимоги до безпеки. Крім того, без наявності детальної документації складно сформувати коректний запит, а специфіка деяких API потребує ретельного вивчення структури відповідей, що ускладнює автоматизацію інтеграцій [7, 14-15].

На прикладі хмарного об’єктного сховища MinIO можна простежити типову реалізацію моніторингу через REST API. Ця система надає власні endpoint's, які дозволяють зчитувати інформацію про:

— кількість запитів;

— обсяг переданих даних;

— використання ресурсів;

— загальний стан сервісу.

Крім прямого моніторингу, API цього сховища інтегрується з експортерами для Prometheus, що дає змогу організувати централізоване збирання метрик з урахуванням готової збудованої екосистеми спостереження. Подібна модель добре масштабовується і зручна у поєднанні з контейнеризованими середовищами, де

REST API виконує роль основного каналу передачі діагностичної інформації [7-8, 14].

Порівняння підходів на прикладах популярних систем. Порівнюючи підходи до моніторингу на прикладі популярних систем, можна визначити специфіку їх використання залежно від:

— типу інфраструктури;

— кількості вузлів;

— потреб у масштабуванні;

— доступності протоколів збору даних.

Prometheus виявляється більш ефективним у великих хмарних середовищах, де пріоритетними є швидкий збір метрик, аналіз часових рядів і використання гнучкої мови запитів PromQL. Його архітектура ґрунтується на пул-моделі збору, що забезпечує централізований контроль над метриками за рахунок опитування експортерами, серед яких найпоширенішим є Node Exporter. Система також підтримує SNMP через відповідний експортер та має вбудовану інтеграцію з REST API, що робить її придатною для автоматизованих середовищ з відкритими інтерфейсами [7-8, 14].

Zabbix надає ширший набір інструментів для гібридних систем, де зустрічаються різні типи пристроїв, від серверів до мережевого обладнання. Його перевага полягає у поєднанні агентного та безагентного моніторингу, включно з підтримкою SNMP, SSH, IPMI, а також REST API для інтеграції зі сторонніми сервісами. Централізоване управління, зручна система тригерів і налаштовувані шаблони дозволяють адаптувати Zabbix до складних сценаріїв, у тому числі з попередженням про події в реальному часі [7, 12-14].

Netdata орієнтований на швидку візуалізацію показників локальної продуктивності, особливо у разі моніторингу окремих вузлів або невеликих систем. Завдяки автономному агенту та вбудованому веб-інтерфейсу з мінімальною затримкою ця система надає детальні графіки стану системи без необхідності додаткових налаштувань. Хоча підтримка SNMP та REST API реалізована

частково, основний фокус Netdata залишається на простоті встановлення та негайному доступі до показників без складної конфігурації [6-7, 12-14].

Таблиця 1.1 – Порівняння підходів на прикладах популярних систем

Система	**Підтримка агентів**	**Agentless**	**SNMP**	**REST API**	**Особливості**
Prometheus	Так (Node Exporter)	Частково	Так (через SNMP exporter)	Так	Пул-модель збору, підтримка алертів, PromQL
Zabbix	Так (Zabbix Agent)	Так (SSH, SNMP, IPMI)	Так	Частково	Потужна система тригерів, централізоване керування
Netdata	Так (автономний агент)	Ні	Обмежено	Частково	Веб-інтерфейс у реальному часі, мінімальна затримка

У результаті вибір платформи визначається конкретними потребами (див. табл. 1.1): масштабованість і аналітика — у випадку Prometheus, гнучкість і розширюваність — у Zabbix, локальна деталізація з фокусом на візуалізацію — у Netdata. Такий поділ дозволяє точніше оцінити ефективність моніторингу залежно від контексту застосування [6-8, 11-13].

Вибір підходу відповідно до завдань моніторингу. Підсумовуючи аналіз підходів до моніторингу та їхніх можливостей, доцільно зосередити увагу на виборі рішення відповідно до конкретних умов впровадження, з урахуванням ключових характеристик середовища, у якому працює система. Одним із визначальних факторів є тип інфраструктури — фізична, віртуалізована або хмарна, оскільки

кожна з них накладає свої обмеження та вимоги до збору метрик. Наприклад, у хмарному середовищі часто переважає агентless-моніторинг через REST API, тоді як у локальних системах можливе встановлення агентів із глибоким доступом до внутрішніх показників [7, 11-13].

Не менш важливою є наявність доступу до вузлів, зокрема можливість встановлення програмного забезпечення або реалізації віддаленого опитування. У випадках, коли доступ обмежений або регламентований політиками безпеки, вибір надається методам, що не вимагають розгортання агентів або мають підтримку стандартів SNMP. Додатково враховується тип обладнання, оскільки вимоги до моніторингу серверів суттєво відрізняються від вимог до мережевого обладнання, систем зберігання чи периферійних пристроїв, які часто використовують різні протоколи або формат подачі телеметрії [7, 12-13].

Швидкість реакції системи — ще один критичний параметр. В окремих випадках необхідно отримувати дані в режимі реального часу, що потребує постійного опитування або використання подієвої моделі, тоді як для більшості завдань достатньо періодичного збору інформації із заданим інтервалом. Поряд із цим, слід враховувати і глибину збору метрик: деякі системи функціонують на основі лише базових параметрів, тоді як інші потребують поглибленої аналітики, зокрема кастомних лічильників, показників кешування, IO або специфічних параметрів продуктивності окремих процесів [5-7].

Поєднання всіх цих чинників дозволяє формувати адаптовану до потреб архітектуру моніторингу, яка відповідає вимогам масштабованості, надійності та ефективності. Такий підхід забезпечує не лише стабільний контроль за станом інфраструктури, а й слугує основою для її подальшої оптимізації, зниження простоїв та підвищення загальної продуктивності IT-середовища.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АРХІТЕКТУРИ ПРОГРАМНОГО КОМПЛЕКСУ

2.1 Архітектура системи моніторингу та оптимізації хмарних сховищ

Архітектура програмного комплексу, що розроблюється для цілей моніторингу та оптимізації хмарного сховища, базується на класичній моделі взаємодії між:

— клієнтом;

— системою зберігання;

— моніторинговим компонентом;

— панеллю керування.

Такий підхід дозволяє не лише формалізувати механізми взаємодії з об’єктним середовищем, а й забезпечити наскрізний контроль за продуктивністю, своєчасне виявлення проблемних ділянок та корекцію параметрів функціонування з урахуванням реальних умов навантаження. У зв’язку з цим доцільно розглянути кожен функціональний компонент архітектури, зосереджуючи увагу на їх ролі, порядку взаємодії та технологічних зв’язках (див. рис. 2.1).

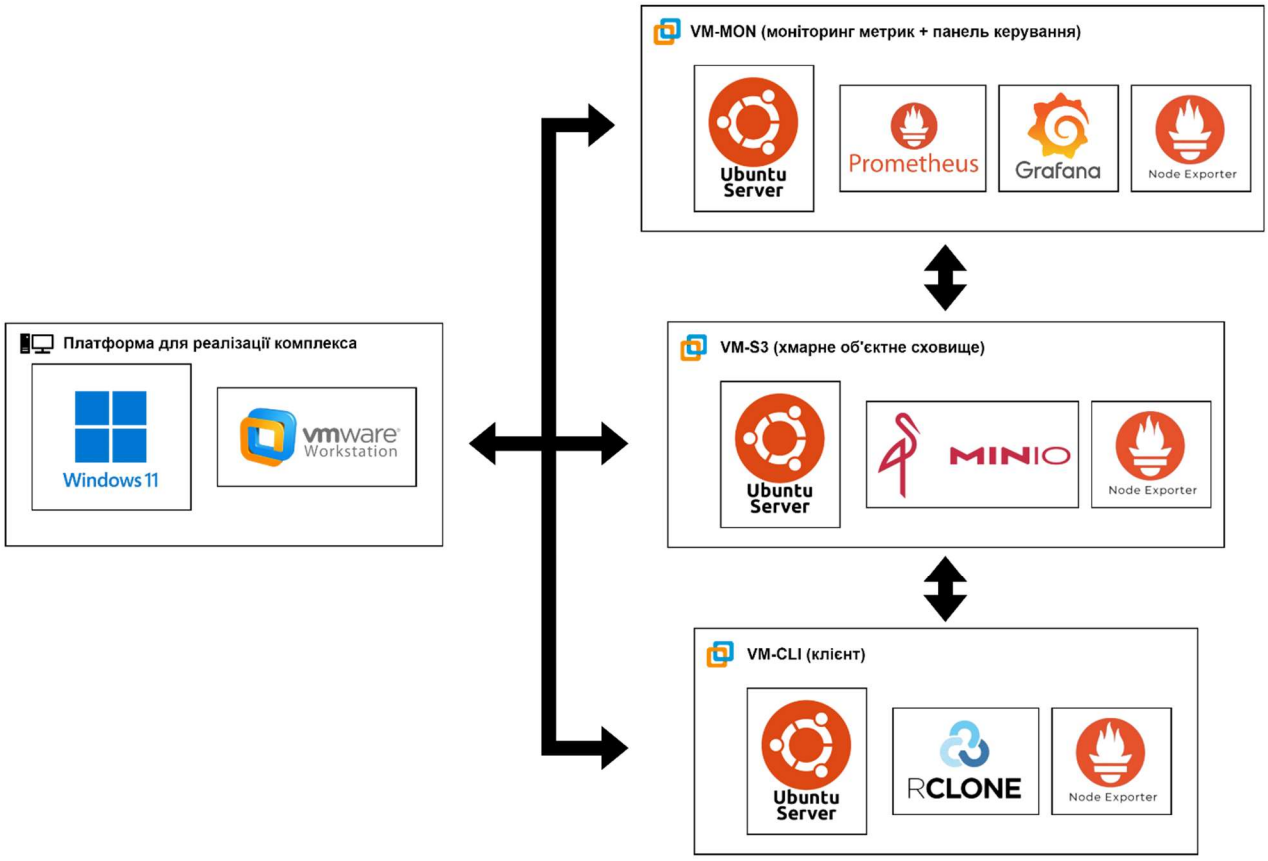


Рисунок 2.1 – Архітектура програмного комплексу для моніторингу та оптимізації продуктивності хмарного сховища

Клієнтська частина – Rclone. Початковим елементом структури є клієнтська частина, що реалізована у вигляді віртуальної машини в середовищі VirtualBox. Вона імітує типове локальне середовище користувача або внутрішній сервер невеликої організації, у якому відпрацьовуються сценарії доступу до хмарного сховища. Для підключення до віддалених ресурсів використовується утиліта Rclone — гнучкий інструмент командного рядка, що забезпечує синхронізацію, копіювання та монтування віддалених сховищ як локальних директорій. У межах розроблюваної системи він виконує функцію проміжної ланки між файловою системою клієнта та об’єктним сховищем MinIO [10, 22-23].

Серед можливостей Rclone, які активно застосовуються у цій архітектурі, особливо актуальними є підтримка багатопотокової передачі даних, кешування запитів у пам’яті або на локальному диску, а також монтування сховища у вигляді віртуального тому за допомогою функції Rclone Mount. Комунікація між Rclone і

MinIO відбувається через S3-сумісний інтерфейс, з використанням HTTPS-протоколу, з типовими портами 443 або 9000 залежно від налаштувань. Ідентифікація та авторизація реалізовані шляхом використання пари ключів — Access Key та Secret Key, які гарантують безпечну аутентифікацію користувача та збереження конфіденційності під час усієї взаємодії з об’єктним сховищем [1, 10, 23].

У результаті такої побудови забезпечується базовий рівень надійної інтеграції локального середовища із хмарною інфраструктурою, що відкриває можливості для подальшого аналізу ефективності, реагування на зміну навантаження та реалізації політик оптимізації на наступних рівнях архітектури (див. рис. 2.2).

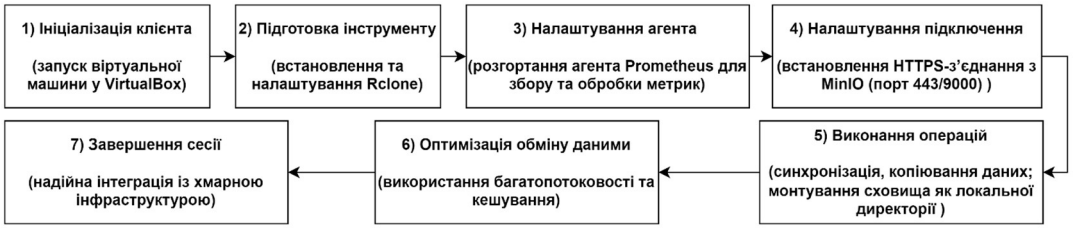


Рисунок 2.2 – Алгоритм роботи клієнтської частини із використанням Rclone та агента збору метрик

Об’єктне сховище – MinIO. Об’єктне сховище, реалізоване в межах описаної архітектури, представлено системою MinIO — високопродуктивним і водночас легким S3-сумісним рішенням, що дозволяє локальне розгортання без необхідності використання зовнішніх хмарних провайдерів. Сховище підтримує багатокористувацький доступ, політики контролю прав і масштабування у межах обраної файлової системи, завдяки чому забезпечується гнучкість у налаштуванні та адаптація під специфіку конкретного середовища. У рамках проєкту MinIO розгортається як окрема віртуальна машина, яка використовує файлову систему типу ext4 або ZFS із увімкненим стисненням, що сприяє зниженню навантаження на сховище, пришвидшує доступ до даних і зменшує загальний обсяг збережених об’єктів [1, 10].

Для підвищення стійкості до відмов використовується логічне розділення об’єктів на гарячі (Tier 1) і холодні (Tier 2) дані. Усі об’єкти, що надходять до сховища через Rclone, зберігаються у вигляді файлів із унікальними ключами, метаданими та підтримкою версіювання, що дозволяє відновлювати попередні стани та організовувати архівацію. У межах тестового середовища передбачено перевірку декількох режимів роботи, включно з прямим синхронізованим завантаженням та варіантом монтування через "Rclone Mount", що дає змогу порівняти продуктивність і затримки при читанні та записі у різних конфігураціях (див. рис. 2.3) [1, 10, 23].

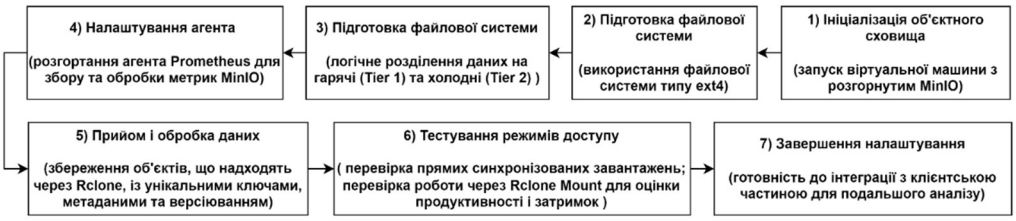


Рисунок 2.3 – Алгоритм роботи об’єктного сховища MinIO в архітектурі системи

Система моніторингу – Prometheus. Моніторинг функціонування сховища реалізовано за допомогою системи Prometheus, яка є одним із найбільш поширених рішень для збору, зберігання та обробки метрик у реальному часі. Prometheus розгортається на окремій віртуальній машині з налаштуванням регулярного опитування MinIO через спеціалізований компонент MinIO Exporter. Цей компонент надає доступ до показників, таких як:

— кількість об’єктів у сховищі;
— середній час обробки запитів;
— використання оперативної пам’яті;
— активні з’єднання;
— загальний обсяг переданих даних.

Окрім цього, Prometheus також здійснює моніторинг клієнтської частини, зокрема утиліти Rclone, за допомогою механізму RC (Remote Control) або сторонніх скриптів. Таким чином, охоплення метрик забезпечується як на рівні

клієнтського середовища, так і серверної інфраструктури, що дозволяє проводити цілісний аналіз продуктивності системи та своєчасно виявляти потенційні проблеми в роботі сховища (див. рис. 2.4) [7-8, 23].

Панель керування – Grafana. Завершальним елементом архітектури програмного комплексу виступає Grafana — платформа візуалізації, яка інтегрується з Prometheus як джерелом метрик та надає інструменти для побудови інформаційних панелей. Завдяки можливості створення кастомізованих графіків і таблиць, інтерфейс Grafana забезпечує зручну візуалізацію даних, включно з навантаженням на сховище, швидкістю читання й запису, кількістю активних з'єднань та рівнем використання кешу на клієнтській стороні. Серед її функцій:

— налаштування тригерів при досягненні критичних значень;

— динамічні фільтри за часовими діапазонами, об'єктами та клієнтами.

Відображення ключових параметрів у реальному часі дає змогу оцінити ефективність взаємодії компонентів і своєчасно здійснювати оптимізаційні дії (див. рис. 2.4) [7-9].



Рисунок 2.4 – Алгоритм роботи системи моніторингу на базі Prometheus і Grafana

Віртуальні машини об'єднані в одну внутрішню мережу (Internal Network), що дає змогу здійснювати прямий обмін даними між вузлами без виходу в зовнішню мережу. Такий підхід дозволяє уникнути зайвих затримок і зосередитися на дослідженні внутрішньої взаємодії сервісів.

Для кожного компонента виділено окремий порт:

— MinIO використовує HTTPS із портом 9000;

— Prometheus працює через HTTP на порту 9090;

— Grafana функціонує як панель керування на порту 3000;

— Rclone Remote Control активується через порт 5572.

Усі сервіси взаємодіють між собою в єдиному середовищі, утворюючи повноцінну інфраструктуру для моніторингу, керування та аналізу роботи об’єктного сховища у хмарній архітектурі (див. рис. 2.5) [8-10, 22-23].

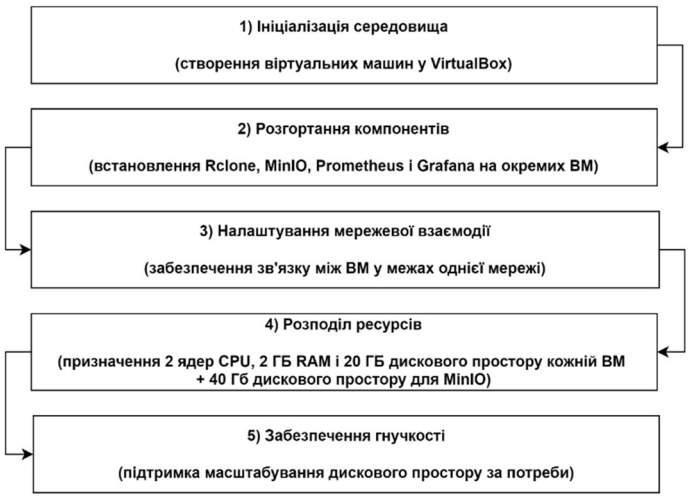


Рисунок 2.5 – Процес налаштування віртуального середовища для програмного комплексу по моніторингу та оптимізації продуктивності хмарного сховища

Застосування такого поєднання інструментів, як Rclone, MinIO, Prometheus і Grafana, дозволяє створити злагоджену систему моніторингу та оптимізації хмарного сховища. Побудована архітектура забезпечує:

— гнучке масштабування;

— можливість інтеграції зі сторонніми сервісами;

— оперативну реакцію на зміни у поведінці системи зберігання даних.

У подальших розділах детальніше розглядається реалізація сценаріїв автоматизації параметрів моніторингу, зокрема приклади побудови візуалізації у Grafana, а також описується, як працює система в умовах обмеженого середовища VirtualBox. Віртуалізаційна платформа, попри наявні обмеження, дозволяє змоделювати основні функціональні процеси, які можуть бути масштабовані або перенесені у більш потужне середовище для виробничого використання [7-10, 22-23].

2.2 Аналіз та вибір інструментів реалізації

У процесі формування системи моніторингу та оптимізації хмарного сховища, що передбачає побудову повноцінного програмного комплексу для підтримки зазначених функцій, критично важливим етапом є ретельний аналіз і вибір інструментів, які відповідатимуть як функціональним, так і нефункціональним вимогам. Ідеться не лише про технічну сумісність компонентів між собою, а й про відповідність їх можливостей до цілей, що ставляться перед усією системою. Саме тому доцільно розглядати кожен обраний інструмент крізь призму реальних сценаріїв застосування, характеру навантаження та потенціалу до масштабування.

Перш ніж перейти до інтеграції, необхідно здійснити комплексне дослідження ключових технологій, що беруть участь в архітектурі. Такий підхід дозволяє не лише виявити найдоцільніше поєднання засобів, а й заздалегідь врахувати можливі обмеження, а також сформувати уявлення про гнучкість налаштування кожного компонента. У випадку з обраною реалізацією розгляд зосереджується навколо таких інструментів:

— MinIO (об’єктне сховище сумісне з протоколом Amazon S3);
— Prometheus (система збору та зберігання метрик);
— Grafana (платформа візуалізації та аналітики);
— Rclone (утиліта для синхронізації та доступу до віддалених файлових систем).

Кожен із зазначених компонентів виконує свою функцію в межах єдиної логічної структури:

1. MinIO відповідає за фізичне зберігання даних і доступ до них через API.
2. Prometheus здійснює регулярне опитування джерел телеметрії, збираючи та агрегуючи показники.
3. Grafana забезпечує інтерактивну візуалізацію отриманих даних.

4. Rclone виступає як клієнтський інструмент для взаємодії з об’єктним сховищем, зокрема в частині завантаження, завантаження та синхронізації інформації [1, 7-10, 23].

Окрім основних характеристик, важливо врахувати особливості налаштування кожного інструмента в умовах обмежених ресурсів, типових для тестового або локального середовища. У деяких випадках доцільно розглядати альтернативи або додаткові модулі, якщо базові можливості потребують розширення. Результати такого аналізу формують підґрунтя для подальшого проєктування архітектури та впровадження автоматизованих сценаріїв моніторингу.

Вибір об’єктного сховища – MinIO. Вибір об’єктного сховища для моделювання системи моніторингу здійснювався з урахуванням вимог до:

— локального розгортання;
— продуктивності;
— сумісності з популярними інструментами;
— можливості масштабування в умовах обмежених ресурсів.

Саме ці фактори стали вирішальними при зосередженні уваги на MinIO як базовому рішенні для побудови внутрішньої інфраструктури, орієнтованої на використання S3-сумісного API. Такий підхід дозволяє легко інтегрувати сховище з утилітами, які вже підтримують протоколи Amazon S3, серед яких rclone, aws-cli, s3cmd та інші, що значно спрощує процес експлуатації та адміністрування [1, 10, 23].

На відміну від сервісів на зразок Amazon S3 або Google Cloud Storage, які передбачають використання зовнішніх хмарних інфраструктур, MinIO дозволяє розгорнути повноцінне об’єктне сховище локально, зокрема на споживчому обладнанні, у віртуальному середовищі або навіть у Docker-контейнерах, що робить його надзвичайно зручним для симуляції реальних сценаріїв у межах обмеженого середовища, наприклад, у межах тестової лабораторії чи внутрішнього проєкту без виходу в інтернет. Завдяки відкритому вихідному коду MinIO забезпечує прозорість внутрішньої логіки роботи та допускає адаптацію під

специфічні потреби конкретного проєкту, що неможливо реалізувати в закритих комерційних продуктах [1, 10, 15].

Серед переваг, що формують додаткову вагу на користь MinIO, слід виділити повну підтримку API Amazon S3, а також гнучкість розгортання — як у вигляді окремого вузла, так і в кластерному режимі. Підтримка сучасних систем ініціалізації, таких як "systemd", можливість використання Kubernetes та сумісність із Docker Compose роблять інтеграцію швидкою та технологічно прозорою. Оптимізація під сучасне обладнання дозволяє досягати високих показників продуктивності навіть у віртуалізованому середовищі, зокрема у VirtualBox або VMware, що дає змогу моделювати реальні навантаження в лабораторних умовах без надмірного споживання ресурсів (див. рис. 2.6) [1, 3, 10, 14, 22, 27].

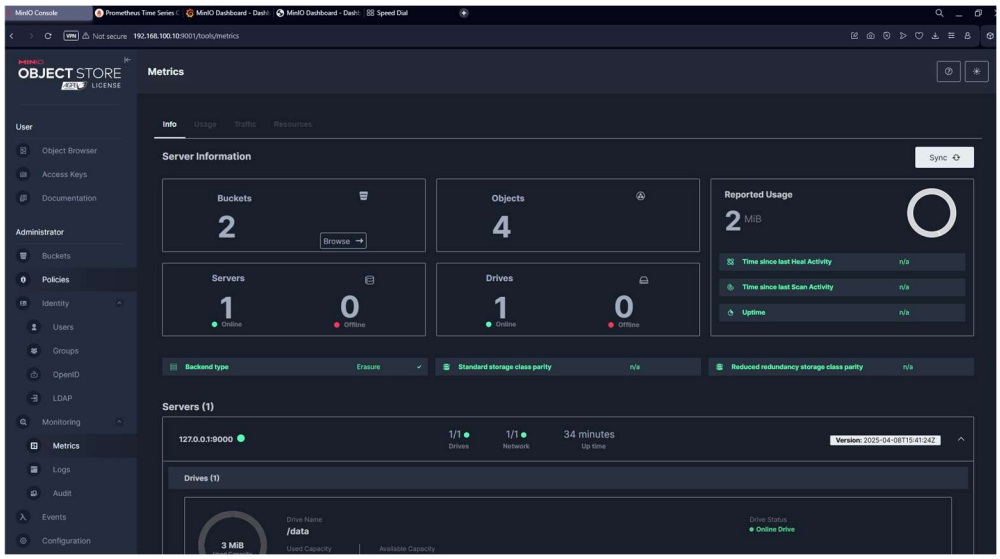


Рисунок 2.6 – Веб-інтерфейс локального хмарного об’єктного сховища MinIO

Гнучкість архітектури MinIO забезпечується підтримкою горизонтального масштабування шляхом розподілу даних між окремими вузлами або дисками. Окрім цього, вбудований веб-інтерфейс надає доступ до базових функцій керування, моніторингу та налаштування без потреби взаємодії через командний рядок, а підтримка Prometheus дозволяє інтегрувати сховище у вже наявну систему моніторингу. Завдяки активній спільноті та регулярно оновлюваній документації,

яка є структурованою і доступною для новачків, процес ознайомлення з продуктом не викликає труднощів навіть у користувачів із базовим досвідом [8, 10].

Таблиця 2.1 – Порівняння альтренатив MinIO

Система	**Переваги**	**Недоліки**
Ceph	Висока відмовостійкість, масштабованість, підтримка S3	Важкий у налаштуванні, потребує багато ресурсів, складна підтримка
OpenIO	Масштабоване сховище з підтримкою S3, відкритий код	Менш активна спільнота, складніша документація
MinFS	Мінімалістична реалізація S3-сховища	Обмежена функціональність, слабка підтримка
LocalStack (S3)	Емулятор сервісів AWS, зокрема S3	Не для продуктивного використання, лише для тестів

Оцінка альтернативних рішень (див. табл. 2.1) дозволила сформувати повніше уявлення про специфіку кожного з них. Наприклад, Ceph забезпечує високу відмовостійкість і глибоку масштабованість, однак його складність у налаштуванні та значні вимоги до ресурсів роблять його менш придатним для моделювання на обмеженій інфраструктурі. OpenIO демонструє відкриту архітектуру з підтримкою S3, проте має менш активну спільноту та менш зручну документацію. MinFS орієнтований на мінімалістичний функціонал, що, хоч і спрощує розгортання, обмежує гнучкість при інтеграції. LocalStack у свою чергу розрахований переважно на тестування API, а не на повноцінне зберігання даних у продуктивному режимі [10, 14, 24].

З урахуванням цих характеристик, саме MinIO забезпечує найбільш збалансоване поєднання простоти використання, функціональної повноти, продуктивності та доступності, що робить його оптимальним вибором у межах

створення моделі системи хмарного сховища для подальшої інтеграції з системою моніторингу та аналізу.

Вибір інструменту моніторингу – Prometheus. У процесі вибору системи моніторингу для моделювання інфраструктури особливу увагу було зосереджено на компонентах, здатних забезпечити ефективне відстеження стану сервісів у реальному часі. Серед доступних варіантів найбільш оптимальним виявився Prometheus, оскільки поєднує критично важливі властивості для побудови спостережуваної системи з мінімальними вимогами до ресурсів. Його підтримка pull-моделі дозволяє опитувати сервіси самостійно, не покладаючись на ініціативу з боку клієнтів, що знижує навантаження на кожен окремий вузол. Завдяки цьому досягається стабільність збору метрик навіть у середовищах з обмеженою пропускною здатністю чи ресурсами [5-8].

Крім того, Prometheus дозволяє зберігати історичні дані, що критично для аналізу динаміки роботи систем, виявлення аномалій та побудови довгострокових звітів. Його сумісність з об’єктним сховищем MinIO, обраним як локальна альтернатива хмарним рішенням, забезпечує єдність інфраструктурного середовища та спрощує розгортання у віртуалізованих середовищах, зокрема на VirtualBox. Додатковою перевагою виступає наявність численних експортерів, включно з тими, що дозволяють зчитувати метрики від rclone, системного моніторингу та самого MinIO, що відкриває можливість глибокої інтеграції без потреби у складному конфігуруванні сторонніх агентів [8, 10, 22-23].

Prometheus також відзначається високою продуктивністю при роботі з великими обсягами метрик, зберігаючи здатність швидко обробляти запити завдяки гнучкій мові PromQL, що дозволяє не лише створювати складні метрики, а й легко їх візуалізувати у зв’язці з Grafana або налаштовувати детальну систему оповіщень. Відкрита ліцензія, активна спільнота та регулярне оновлення документації сприяють швидкому освоєнню та адаптації інструменту до індивідуальних сценаріїв. Враховуючи поширеність Prometheus у DevOps-середовищах, можна розраховувати на велику кількість шаблонів, гайдів та

прикладів налаштування, що значно прискорює процес впровадження навіть для невеликих команд або локальних моделей (див. рис. 2.7) [7-9].

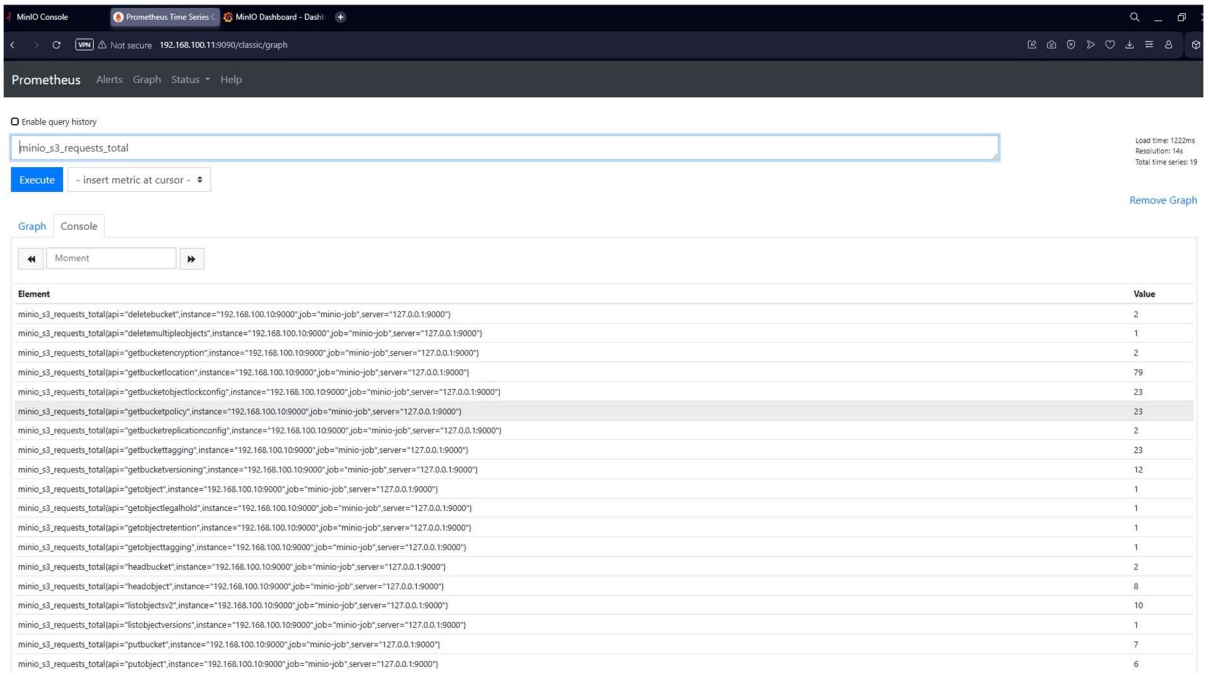


Рисунок 2.7 – Веб-інтерфейс інструмента моніторингу Prometheus

Таблиця 2.2 – Порівняння альтренатив Prometheus

Система	**Переваги**	**Недоліки**
Zabbix	Висока точність, розвинена система сповіщень	Складність налаштування, важке масштабування, важчий агентський підхід
Nagios	Класичне рішення з потужним сповіщенням	Застарілий підхід до моніторингу, менш гнучка мова запитів
Netdata	Дуже наочна візуалізація, швидке встановлення	Більше підходить для моніторингу системи, ніж додатків
InfluxDB + Telegraf	Висока точність збереження метрик, інтеграція з Grafana	Більше орієнтовано на time-series базу, не має такої гнучкої pull-моделі

У порівнянні з альтернативами, такими як Zabbix, Nagios, Netdata або InfluxDB у зв'язці з Telegraf (див. табл. 2.2), Prometheus забезпечує найкраще співвідношення між простотою використання, гнучкістю масштабування та технічною досконалістю. На відміну від Zabbix, не потребує складного агентського підходу, на відміну від Nagios, орієнтований на сучасні парадигми побудови моніторингу, а у порівнянні з Netdata, не обмежується лише моніторингом локальних систем. InfluxDB з Telegraf хоча й забезпечує високу точність метрик, менш адаптивний у сценаріях з перевагою pull-моделі [8, 25].

У підсумку, з огляду на вимоги до продуктивності, сумісності, ресурсоощадності, а також орієнтацію на хмарні та контейнеризовані архітектури, Prometheus демонструє найвищу ефективність серед доступних рішень для моніторингу сервісів.

Вибір системи візуалізації – Grafana. В межах побудови програмного комплексу моніторингу хмарного сховища та виявлення його потенційних проблем, окрім компонентів зберігання та збору метрик, постала потреба у впровадженні засобу візуалізації. Хоча Prometheus уже забезпечує гнучке збирання та збереження даних, ефективна інтерпретація результатів потребує зручного графічного подання, яке дозволить [7-9]:

— оперативно оцінювати стан системи;

— виявляти відхилення;

— приймати рішення на основі наочних графіків.

Для цього обрано Grafana — систему, яка безпосередньо сумісна з Prometheus як джерелом даних, що значно полегшує інтеграцію без необхідності додаткових конекторів чи адаптерів. Її функціональність охоплює [8-9]:

— створення інформативних та кастомізованих дешбордів;

— налаштування алертів із можливістю надсилання повідомлень через популярні канали (Telegram, Slack чи email);

— підтримку фільтрів і елементів взаємодії.

Grafana дозволяє будувати різноманітні типи графіків, таблиць, теплових мап та інших форм подання, надаючи при цьому можливість легко імпортувати готові

шаблони дашбордів. Такий підхід суттєво скорочує час на налаштування візуалізації, що особливо актуально на етапі тестування або демонстрації функціональності системи. Зручне керування правами доступу дозволяє забезпечити рольову взаємодію між різними групами користувачів, а система оновлень та розширень через плагіни створює передумови для гнучкого масштабування та адаптації під специфічні вимоги проєкту (див. рис. 2.8) [8].



Рисунок 2.8 – Веб-інтерфейс системи візуалізації метрик Grafana

Таблиця 2.3 – Порівняння альтренатив Grafana

Система	**Переваги**	**Недоліки**
Kibana (Elastic)	Потужна інтеграція з Elastic Stack	Не призначена для Prometheus, складна конфігурація
Chronograf	Частина Telegraf/InfluxDB, проста інтеграція	Орієнтована лише на власний стек, менш потужна графіка
Dashboards в Zabbix	Вбудована в систему моніторингу	Менш гнучка, складна побудова складних графіків
Metabase	Для BI-аналітики, SQL-запити	Не призначена для моніторингу метрик у реальному часі

Порівняльний аналіз альтернатив — таких як Kibana, Chronograf, Metabase або Zabbix dashboards (див. табл. 2.3) — продемонстрував певні переваги кожного з них у межах власного стеку, проте жоден із розглянутих інструментів не забезпечує такої ж рівня універсальності, гнучкості та сумісності з Prometheus, як Grafana. Наприклад, Kibana тісно інтегрується лише з Elastic Stack і має складні налаштування, Chronograf обмежений екосистемою InfluxDB, а Metabase радше підходить для BI-аналітики, ніж моніторингу в реальному часі [8-9, 26].

У підсумку, Grafana у поєднанні з Prometheus формує цілісне рішення для побудови системи моніторингу, яке дозволяє не лише ефективно збирати й аналізувати метрики, а й зручно представляти їх у візуальному вигляді, що підвищує загальну функціональність програмного комплексу та забезпечує простоту розгортання, підтримки та масштабування в майбутньому.

Загальне обґрунтування обраного стеку – MinIO, Prometheus, Grafana. Поєднання компонентів MinIO, Prometheus та Grafana разом із додатковими утилітами на кшталт rclone формує цілісну узгоджену систему, яка забезпечує можливість побудови гнучкого сховища даних із сумісністю до інтерфейсу S3. Така конфігурація дає змогу не лише імітувати роботу реальних об’єктних сервісів зберігання, але й паралельно здійснювати постійний моніторинг стану системи, зокрема [8-10, 22-23]:

— продуктивності;

— обсягу даних;

— навантаження на обчислювальні ресурси.

Завдяки інтеграції з Prometheus стає можливим збирання великого масиву метрик у реальному часі, які потім зручно виводяться у Grafana через візуальні панелі, що забезпечує користувача швидким доступом до діагностичної інформації, дозволяючи оперативно аналізувати відхилення від норми, оцінювати ефективність та коригувати параметри системи. Візуалізація даних, зібраних зі сховища, відкриває ширші можливості не лише для моніторингу, а й для прогнозування ресурсного споживання або своєчасного масштабування [8-9].

Оскільки така конфігурація сумісна з різними середовищами — Docker, Kubernetes, bare-metal — вона зберігає універсальність та гнучкість на всіх етапах розгортання, що дозволяє легко переносити рішення між платформами, проводити тестування на окремих вузлах або ж масштабувати систему у повноцінному кластері без суттєвих змін у її архітектурі [17].

Застосування такого стеку є типовим підходом у корпоративних системах зберігання і моніторингу, тому використання його у навчальних або лабораторних середовищах дає можливість моделювати сценарії, наближені до реальних, що відкриває доступ до практичної роботи з інструментами DevOps-інфраструктури та створює умови для набуття релевантного досвіду з побудови, супроводу й масштабування сучасних сервісних систем.

2.3 Побудова віртуалізованого середовища на базі VirtualBox

Однією з основних вимог до побудови навчально-дослідницької інфраструктури або програмного комплексу в умовах обмежених ресурсів є можливість реалізації такого середовища на звичайному персональному комп’ютері. VirtualBox у цьому випадку виступає як безкоштовний інструмент для створення та керування віртуальними машинами, що дає змогу моделювати складні мережеві структури без виходу за межі локального середовища. Завдяки гнучкості налаштувань, навіть на базовому обладнанні можна відтворити функціонування кількох логічно ізольованих сервісів, кожен з яких буде відповідати за окрему частину системи [8-10, 22-23]:

— зберігання;

— моніторинг;

— взаємодію з користувачем.

У такому підході передбачається побудова міні-інфраструктури, яка складається з трьох основних віртуальних машин. Перша, умовно позначена як

VM-S3, виконує роль об’єктного сховища на базі MinIO. Друга — VM-MON — відповідає за моніторинг і візуалізацію даних, об’єднуючи у собі Prometheus і Grafana. Третя віртуальна машина, VM-CLI, імітує клієнта, який проводить операції зі збереження й зчитування даних, що дозволяє протестувати роботу всієї системи в умовах максимально наближених до реального середовища.

Загальна концепція віртуалізованого середовища. Загальна концепція віртуалізованого середовища полягає у створенні ізольованих компонентів, які працюють автономно, проте взаємодіють між собою через налаштовані мережеві зв’язки. Такий підхід дозволяє не лише чітко розподілити функції між вузлами, але й дає змогу емулювати роботу розподілених систем, включно з підтримкою публічних і приватних підмереж, тунелюванням та керуванням доступом, що особливо важливо в межах навчального або тестового стенду, де є потреба зберігати баланс між функціональністю системи та апаратними обмеженнями.

Оптимізація ресурсів виявляється критичною, оскільки в умовах обмеженого обсягу оперативної пам’яті — наприклад, 8 ГБ — потрібно забезпечити функціональність усіх компонентів без перевантаження системи. Кожна віртуальна машина отримує лише необхідний мінімум ресурсів, наприклад, 2 ГБ оперативної пам’яті, обмежений обсяг дискового простору та мінімальний відсоток обчислювальної потужності CPU, що дозволяє досягнути максимальної ефективності в межах наявного середовища, зберігаючи при цьому гнучкість і масштабованість, необхідну для розширення або адаптації у майбутньому.

У підсумку, віртуалізоване середовище, побудоване на VirtualBox, не лише забезпечує необхідну інфраструктурну модель, але й створює платформу для відпрацювання практичних навичок із побудови, налаштування та тестування систем управління та моніторингу сховищ даних [22].

Опис віртуальної машини – VM-S3 (MinIO). Першою віртуальною машиною у складі побудованої інфраструктури виступає VM-S3, яка реалізує функціональність об’єктного сховища на базі MinIO. Її основою є операційна система Ubuntu Server 22.04 LTS з виділеними ресурсами в обсязі двох віртуальних ядер та 2 ГБ оперативної пам’яті. Конфігурація дискової підсистеми включає два

окремих томи: основний обсягом 20 ГБ, призначений для встановлення операційної системи та самого MinIO, а також додатковий диск обсягом 40 ГБ, який монтується як сховище для об’єктів MinIO (див. рис. 2.9).



Рисунок 2.9 – Загальні налаштування віртуального вузла VM-S3

Функціональне призначення цієї машини полягає в імітації сумісного з S3 локального сховища, що забезпечує тестування роботи системи в умовах, наближених до використання реального хмарного сервісу. Для реалізації моніторингу додатково налаштовується експортер метрик, який надає доступ до "Prometheus endpoint", що дозволяє здійснювати збирання статистики роботи MinIO. Додатковий диск при цьому монтується у відповідний каталог і використовується виключно для об’єктного зберігання. Захищене з’єднання (HTTPS) та механізми автентифікації налаштовуються за допомогою змінних середовища або конфігураційних файлів, що спрощує гнучкість адаптації до різних умов запуску [8, 10, 22].

Опис віртуальної машини – VM-MON (Prometheus + Grafana). Другою віртуальною машиною є VM-MON, відповідальна за реалізацію функцій моніторингу та візуалізації. На ній також розгорнуто Ubuntu Server 22.04 LTS з аналогічним ресурсним профілем — два ядра CPU, 2 ГБ оперативної пам’яті та 20 ГБ дискового простору. Основними службами, що запускаються в межах цієї машини, є Prometheus та Grafana, які забезпечують централізоване збирання та

представлення інформації про навантаження і стан інших компонентів системи, зокрема VM-S3 та VM-CLI (див. рис. 2.10).

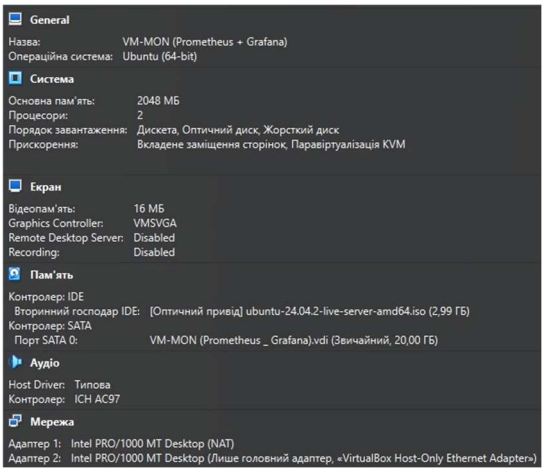


Рисунок 2.10 – Загальні налаштування віртуального вузла VM-MON

Prometheus конфігурується для здійснення регулярних запитів до ендпоінту "/minio/v2/metrics/cluster", що дозволяє отримувати актуальні метрики безпосередньо від MinIO. Зі свого боку Grafana інтегрується з Prometheus як джерелом даних, а для побудови графіків та візуальних панелей використовуються готові шаблони, адаптовані до специфіки сховища. Така конфігурація дає змогу комплексно оцінювати продуктивність, виявляти потенційні вузькі місця та аналізувати поведінку системи під навантаженням [8-10, 22].

Опис віртуальної машини – VM-CLI. Третьою віртуальною машиною в інфраструктурі є VM-CLI — клієнтський вузол, призначений для імітації активності користувача. Система функціонує на базі Ubuntu Server 22.04 LTS з мінімальними, аналогічними попереднім, ресурсами: два ядра процесора, 2 ГБ оперативної пам’яті та диск обсягом 10 ГБ. Через невелику кількість операцій, які передбачаються на цьому вузлі, зазначених характеристик достатньо для ефективного виконання поставлених завдань (див. рис. 2.11).

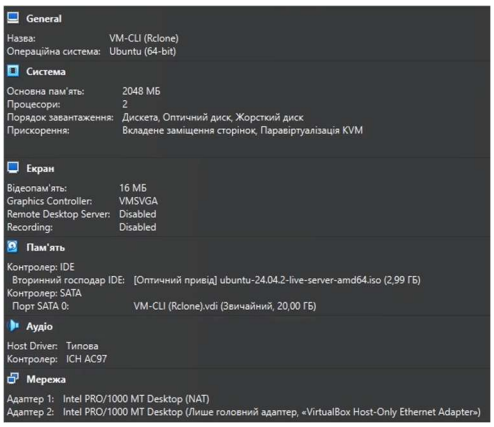


Рисунок 2.11 – Загальні налаштування віртуального вузла VM-CLI

Основна роль цієї машини полягає у встановленні та використанні Rclone — інструмента для роботи з об'єктними сховищами. За його допомогою здійснюється завантаження та зчитування тестових файлів, що дозволяє моделювати активність користувача, тестувати стабільність системи при різних сценаріях доступу до даних, а також створювати штучні навантаження та ситуації з помилками. Завдяки цьому реалізується перевірка надійності роботи всієї інфраструктури та її здатності до обробки запитів у симульованих умовах продуктивного середовища [22-23].

Налаштування мережевої топології. Мережева топологія побудована з урахуванням принципів локальної ізоляції, що передбачає всю взаємодію між віртуальними машинами виключно в межах внутрішньої віртуальної мережі. Такий підхід гарантує, що будь-які зовнішні чинники не матимуть впливу на передачу даних між елементами системи. Внутрішня мережа реалізована за допомогою типу з'єднання Host-Only, що дозволяє віртуальним машинам обмінюватися даними між собою, залишаючись відокремленими від основної локальної мережі та інтернету (див. рис. 2.12) [22].

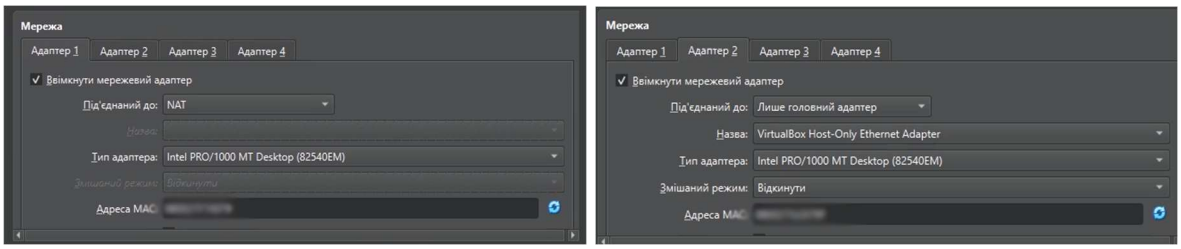


Рисунок 2.12 – Загальні налаштування мережевих інтерфейсів віртуальних вузлів

Поряд із цим застосовано адаптер NAT, який забезпечує вихід у глобальну мережу, проте лише для цілей оновлення систем і встановлення необхідних пакетів. Такий обмежений доступ дозволяє зберігати ізольованість основної частини системи, одночасно забезпечуючи її актуальність і функціональність під час розгортання програмного забезпечення.

Важливим елементом налаштування стала фіксація IP-адрес вручну для кожної віртуальної машини. Встановлення статичної IP-адресації забезпечує стабільність підключення та спрощує налаштування взаємодії між компонентами, такими як Prometheus, Rclone чи MinIO. За відсутності статичних IP-адрес мережеві параметри змінювалися б після кожного перезапуску, що унеможливлює підтримку сталої структури зв’язків та автоматизації. Крім того, внутрішній мережевий інтерфейс дозволяє уникнути будь-якого впливу на зовнішнє середовище або інші пристрої в локальній інфраструктурі, що особливо важливо в умовах тестування або емульованих середовищ (див. рис. 2.13) [8-10, 22-23].

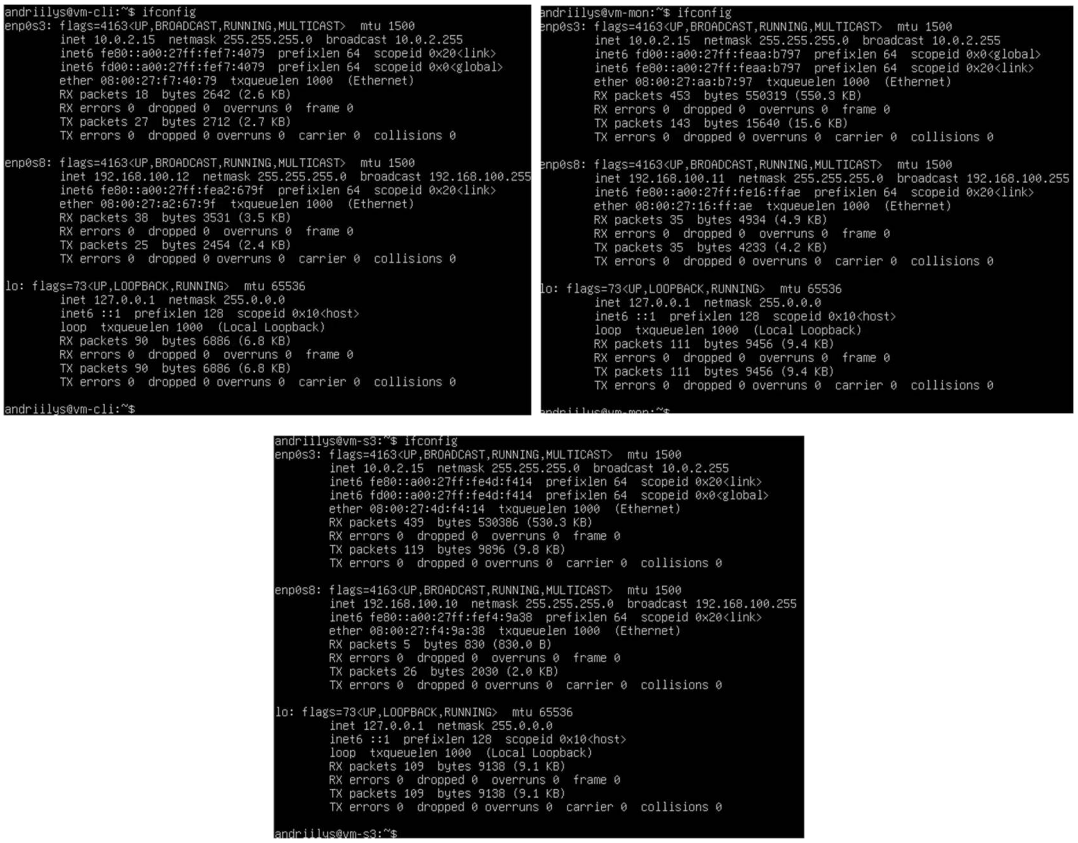


Рисунок 2.13 – Кожен вузол налаштований на статичні IP-адреси для надійної взаємної роботи

У таблиці 2.4 нижче наведено конфігурацію мережевих адаптерів для кожної віртуальної машини, а також відповідні IP-адреси, прив’язані до внутрішнього сегменту.

Таблиця 2.4 – Конфігурація адаптерів віртуальних машин

Віртуальна машина	**Адаптер 1 (NAT)**	**Адаптер 2 (Host-Only)**	**IP-адреса адаптера 2**
VM-S3	Увімкнено	Увімкнено	192.168.100.10
VM-MON	Увімкнено	Увімкнено	192.168.100.11
VM-CLI	Увімкнено	Увімкнено	192.168.100.12

Результат побудови середовища. Після завершення налаштувань створено повноцінну тривузлову модель мікроінфраструктури, яка відображає типову архітектуру з розподілом на сховище, моніторинг та клієнтський вузол. Забезпечено можливість моделювання практичних сценаріїв — зокрема, завантаження файлів до об’єктного сховища MinIO через "rclone" та збір продуктивних метрик з подальшою візуалізацією. Моніторинг дозволяє отримувати інформацію про трафік, кількість об’єктів, помилки та інші показники, які критичні для оцінки роботи системи (див. рис. 2.14) [8-10, 22-23].

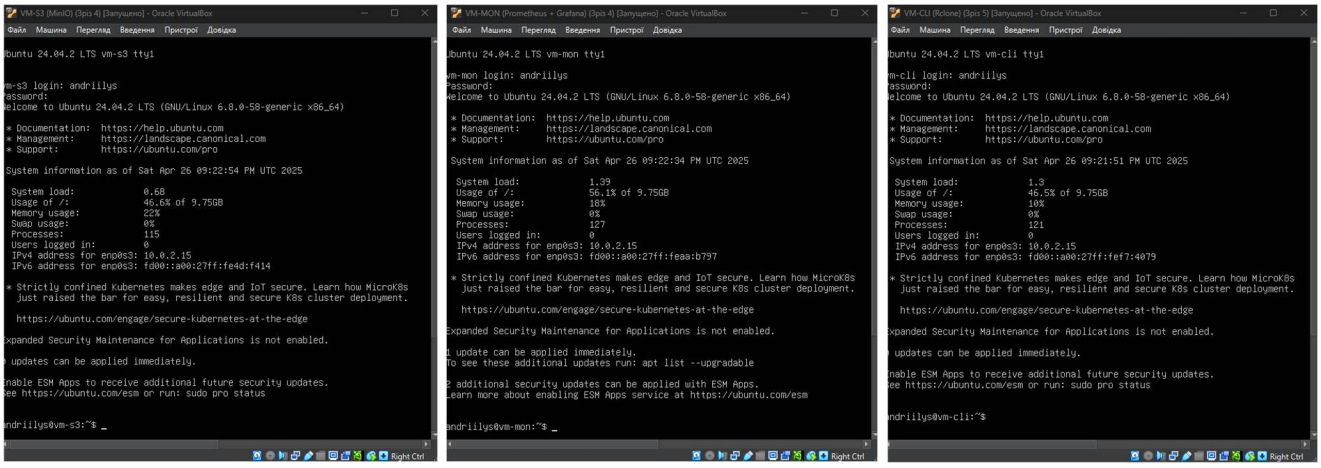


Рисунок 2.14 – Побудоване віртуалізоване середовище тривузлової моделі мікроархітектури з розподілом на сховище, моніторинг та клієнтський вузол (VM-S3, VM-MON, VM-CLI)

2.4 Реалізація та налаштування базового функціоналу комплексу

Реалізація та налаштування базового функціоналу програмного комплексу передбачає поетапну організацію мінімального середовища для зберігання та моніторингу об’єктних даних із використанням MinIO, Prometheus і Grafana. Розгортання відбувається у віртуальному середовищі VirtualBox згідно з попередньо визначеною архітектурою, а процес побудови охоплює весь цикл — від встановлення компонентів до перевірки їх взаємодії через візуалізацію метрик.

Встановлення та налаштування MinIO. Підготовка починається з налаштування середовища для MinIO, що функціонує на віртуальній машині VMS-3 під керуванням Ubuntu Server. Після запуску машини необхідно оновити всі системні пакети для забезпечення сумісності та стабільності подальших налаштувань. Оновлення виконується командою "sudo apt update && sudo apt upgrade -y", що дозволяє автоматизувати процес без втручання користувача (див. рис. 2.15) [10, 22, 27].

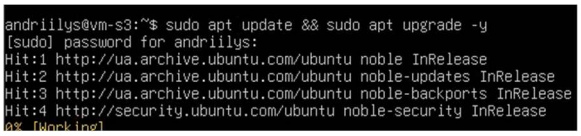


Рисунок 2.15 – Оновлення всіх системних пакетів на вузлі VM-S3

Для розміщення двох основних компонентів MinIO — виконуваного файлу та сховища — створюються відповідні каталоги "/usr/local/bin/minio" і "/data" (див. рис. 2.16).



Рисунок 2.16 – Створення та прив’язка каталогів для MinIO на вузлі VM-S3

Далі відбувається завантаження останньої стабільної версії MinIO з офіційного джерела через "wget", після чого файл отримує права на виконання "chmod +x minio" та переміщується до відповідного каталогу. З міркувань безпеки створюється окремий користувач "minio-user" без права входу в систему, а власником каталогу даних призначається саме цей користувач (див. рис. 2.17) [10, 27].

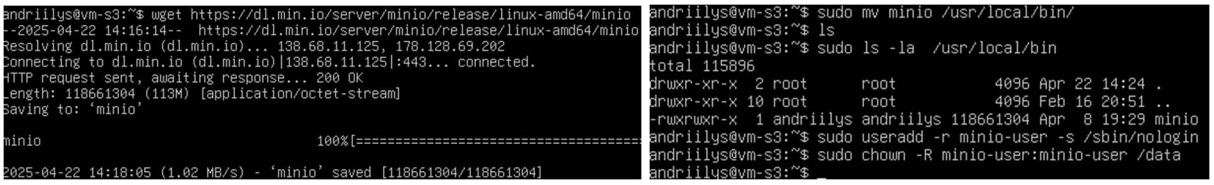


Рисунок 2.17 – Завантаження та розгортання MinIO на вузлі VM-S3

Конфігурація сервісу реалізується через створення системного юніт-файлу "minio.service", який містить необхідні змінні середовища для адміністрування, зокрема "MINIO_ROOT_USER" та "MINIO_ROOT_PASSWORD" (див. рис. 2.18).

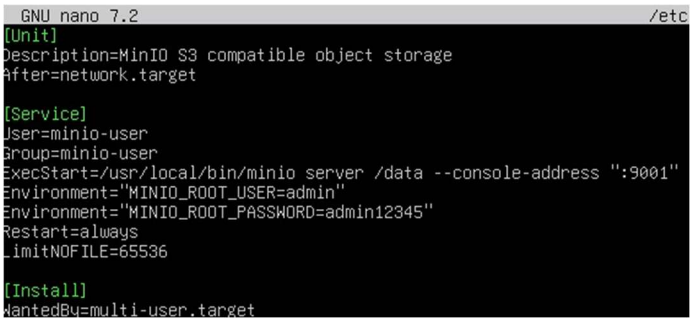


Рисунок 2.18 – Створення системного юніт-файлу "minio.service" на вузлі VM-S3

Після збереження цього файлу він активується командами "sudo systemctl daemon-reexec", "sudo systemctl enable minio" та "sudo systemctl start minio" (див. рис. 2.19).

```
andriilys@vm-s3:~$ sudo systemctl daemon-reload
andriilys@vm-s3:~$ sudo systemctl start minio
andriilys@vm-s3:~$ sudo systemctl status minio
● minio.service - MinIO S3 compatible object storage
     Loaded: loaded (/etc/systemd/system/minio.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-04-22 14:38:59 UTC; 39s ago
   Main PID: 1209 (minio)
      Tasks: 7 (limit: 2272)
     Memory: 219.0M (peak: 219.4M)
        CPU: 847ms
     CGroup: /system.slice/minio.service
             └─1209 /usr/local/bin/minio server /data --console-address :9001

Apr 22 14:38:59 vm-s3 systemd[1]: Started minio.service.
Apr 22 14:39:00 vm-s3 minio[1209]: MinIO Object Storage Server
Apr 22 14:39:00 vm-s3 minio[1209]: Copyright: 2015-2025 MinIO, Inc.
Apr 22 14:39:00 vm-s3 minio[1209]: License: GNU AGPLv3 - https://www.gnu.org/licenses/agpl-3.0.html
Apr 22 14:39:00 vm-s3 minio[1209]: Version: RELEASE.2025-04-08T15-41-24Z (go1.24.2 linux/amd64)
Apr 22 14:39:00 vm-s3 minio[1209]: API: http://192.168.56.107:9000  http://10.0.2.15:9000  http://192.168.100.10:9000  http://127.0.0.1:9000
Apr 22 14:39:00 vm-s3 minio[1209]: WebUI: http://192.168.56.107:9001 http://10.0.2.15:9001 http://192.168.100.10:9001 http://127.0.0.1:9001
Apr 22 14:39:00 vm-s3 minio[1209]: Docs: https://docs.min.io
andriilys@vm-s3:~$
```

Рисунок 2.19 – Запуск та стан сервісу "minio.service" на вузлі VM-S3

Успішний запуск сервісу дозволяє отримати доступ до веб-інтерфейсу адміністрування за портом 9001, а сам сервіс MinIO доступний за стандартним портом 9000 (див. рис. 2.20) [10, 27].

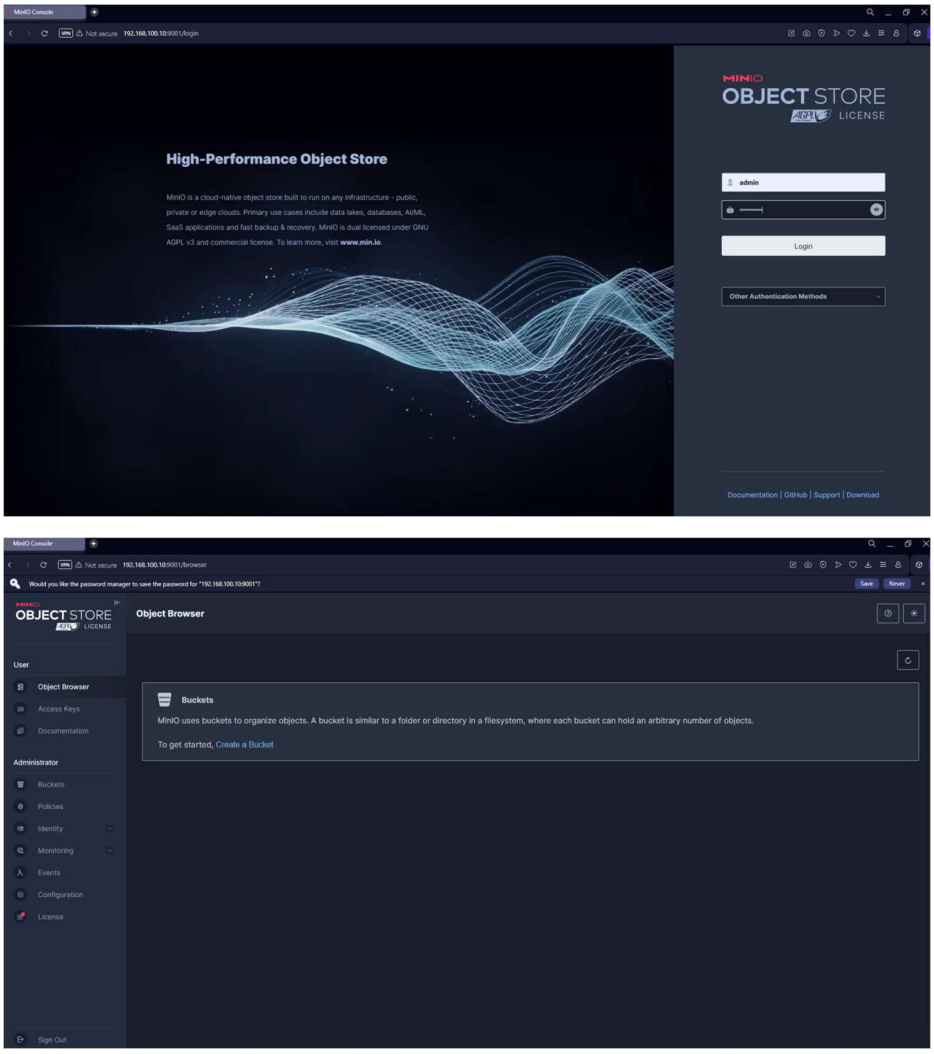


Рисунок 2.20 – Робота локального хмарного об’єктного сховища MinIO зі вузла VM-S3

Інтеграція MinIO з Prometheus. Інтеграція MinIO з Prometheus починається з активації можливості збору метрик, які MinIO здатен генерувати автоматично у форматі, сумісному з Prometheus. Ці метрики доступні за стандартною HTTP-адресою сервера, на якому запущено MinIO, і використовуються для спостереження за станом об’єктного сховища в реальному часі. Щоб Prometheus зміг отримувати ці дані, спочатку необхідно налаштувати його інфраструктуру на віртуальній машині, призначеній для моніторингу [8, 10].

Перед інсталяцією Prometheus виконується оновлення системних пакетів за допомогою "sudo apt update && sudo apt upgrade -y", що дозволяє уникнути потенційних конфліктів під час встановлення (див. рис. 2.21).

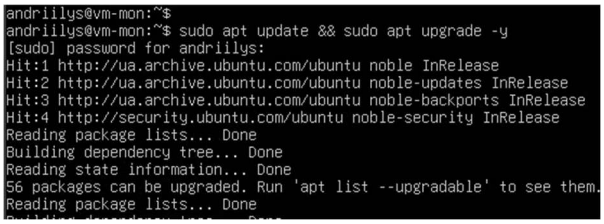


Рисунок 2.21 – Оновлення всіх системних пакетів на вузлі VM-MON

Після цього сам Prometheus встановлюється через стандартну пакетну систему "sudo apt install prometheus -y", а його конфігураційний файл "prometheus.yml" підлягає редагуванню для додавання параметрів збору метрик з MinIO (див. рис. 2.22) [8, 27].

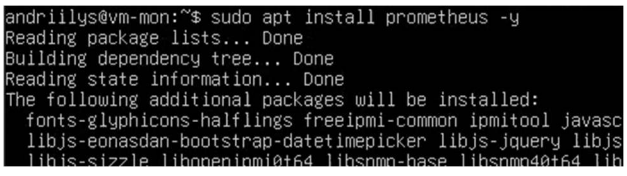


Рисунок 2.22 – Встановлення інструменту моніторингу Prometheus на вузлі VM-MON

У конфігураційний файл додається новий блок "scrape_configs", який спершу формується у вузлі VM-S3 за допомогою mc, для пройдення аунтентифікації, що у

разі вводу базових параметрів без потрібних кодів, Prometheus не бачить метрик, а лише помилку 400, оскільки не має доступу до MinIO (див. рис. 2.23).

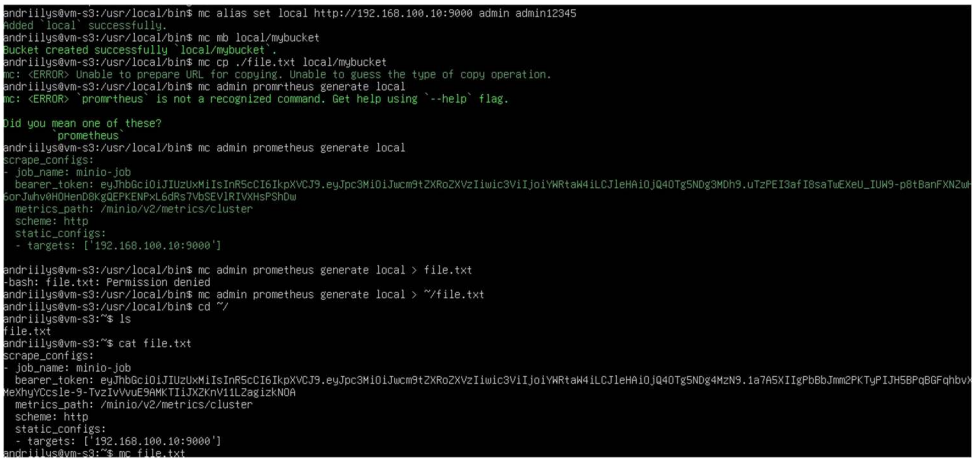


Рисунок 2.23 – Генерація даних для аунтентифікації у MinIO для збору метрик через Prometheus на вузлі VM-MON

Всередині блоку зазначаються статичні IP-адреси у полі targets, що відповідають розгортанню MinIO у локальній мережі. Таким чином, Prometheus знає, з якого вузла очікувати телеметричні дані. Наприклад, адреса 192.168.100.10:9000 використовується як основна точка доступу до метрик MinIO (див. рис. 2.24) [8, 10].

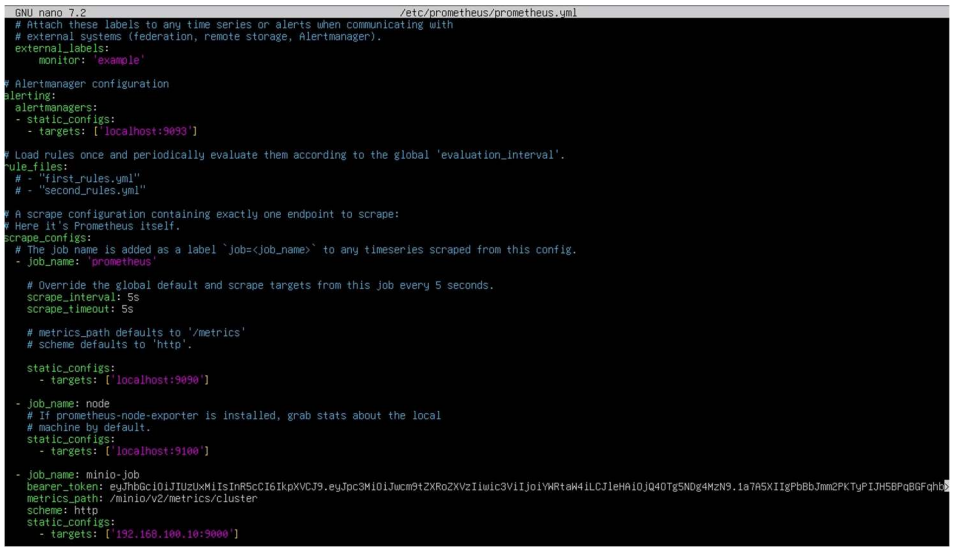


Рисунок 2.24 – Налаштування конфігураційного файлу "prometheus.yml" на вузлі VM-MON

Після збереження змін у конфігурації здійснюється перезапуск служби командою "sudo systemctl restart prometheus", що дозволяє застосувати нові параметри без зупинки всієї системи моніторингу (див. рис. 2.25).

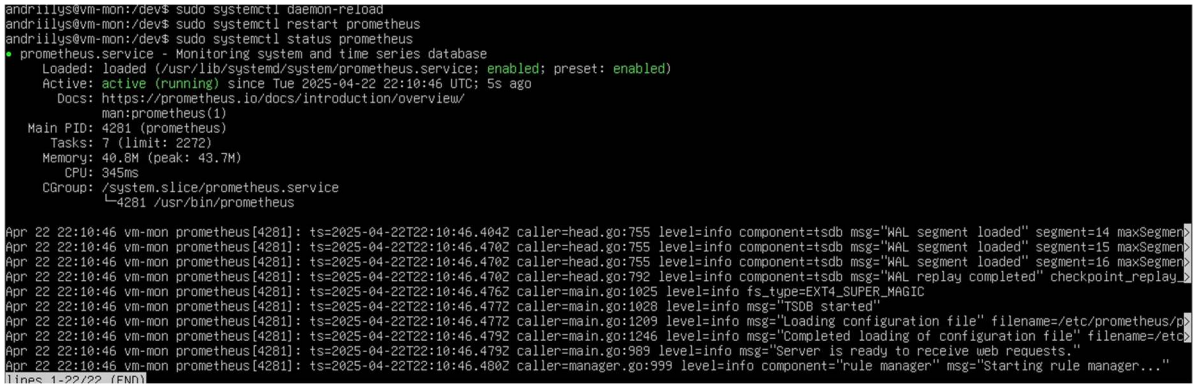


Рисунок 2.25 – Перезапуск та перевірка стану сервіса "prometheus" на вузлі VM-MON

Через кілька секунд після запуску оновленої служби в інтерфейсі Prometheus за адресою IP-сервера на порту 9090 у вкладці "Targets" з’являється новий запис із назвою "minio". Він сигналізує про те, що підключення до джерела метрик відбулося успішно, а система отримує актуальні дані зі сховища MinIO для подальшого аналізу та візуалізації (див. рис. 2.26) [8, 10, 27].

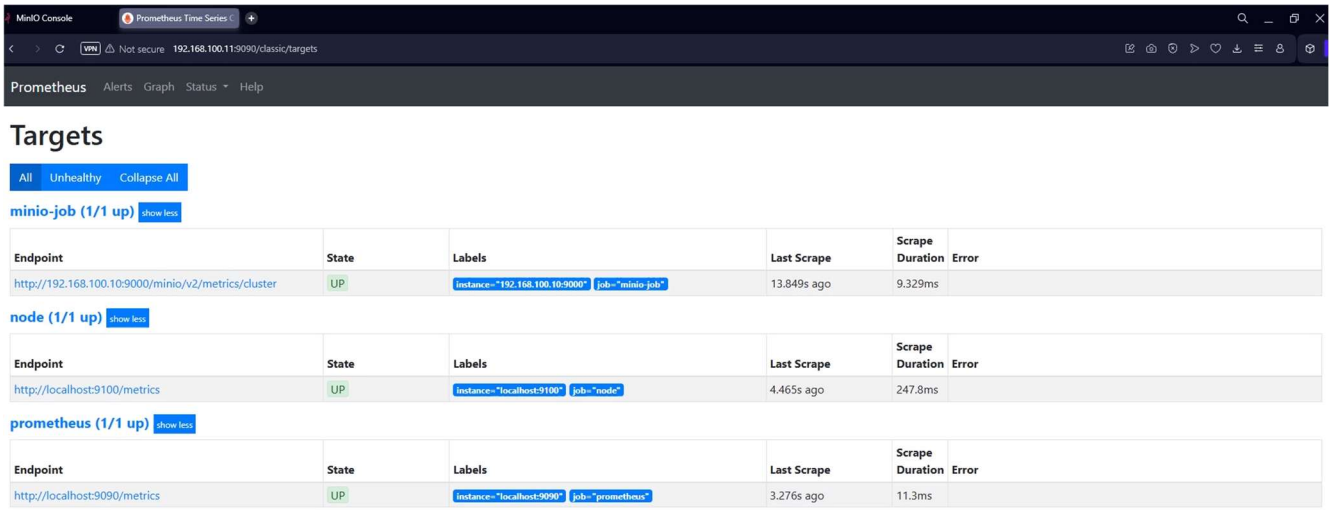


Рисунок 2.26 – Робота інструмента для збору метрик Prometheus зі вузла VM-MON

Розгортання Grafana та налаштування візуалізації. Після налаштування Prometheus для збору метрик з MinIO наступним кроком стає розгортання системи візуалізації, зокрема встановлення та конфігурація Grafana. Її використання дозволяє представити телеметричні дані у зрозумілому графічному форматі, що суттєво спрощує моніторинг стану сховища. Встановлення Grafana розпочинається з підготовки системи, де додаються необхідні компоненти та ключі репозиторіїв. Після цього оновлюється список пакетів і виконується безпосереднє встановлення Grafana за допомогою "APT" (див. рис. 2.27) [9-10, 27].

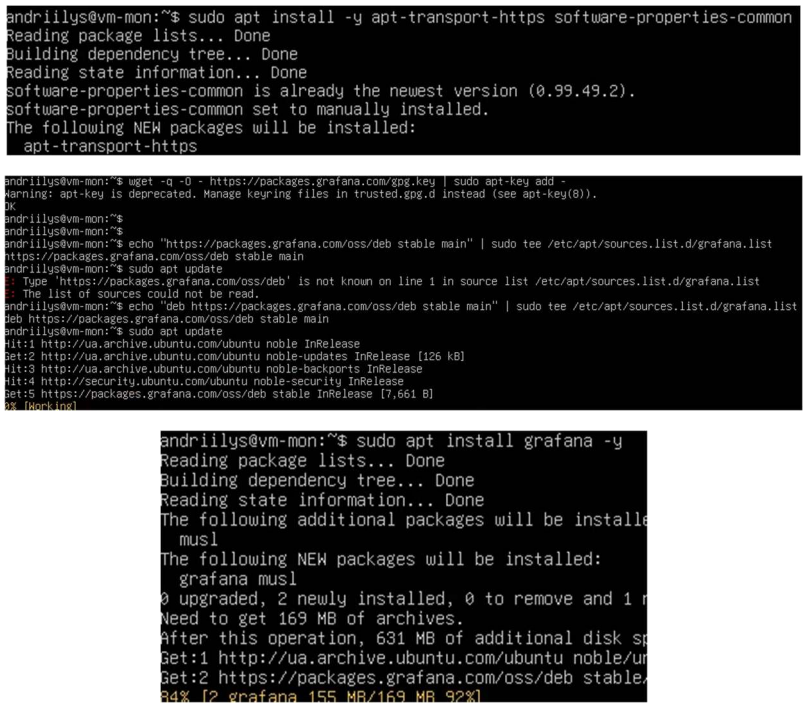


Рисунок 2.27 – Ручне оновлення списку пакетів шляхом додавання Grafana на вузлі VM-MON

Після інсталяції необхідно активувати службу командою "sudo systemctl enable grafana-server" і запустити її через "sudo systemctl start grafana-server" (див. рис. 2.28).

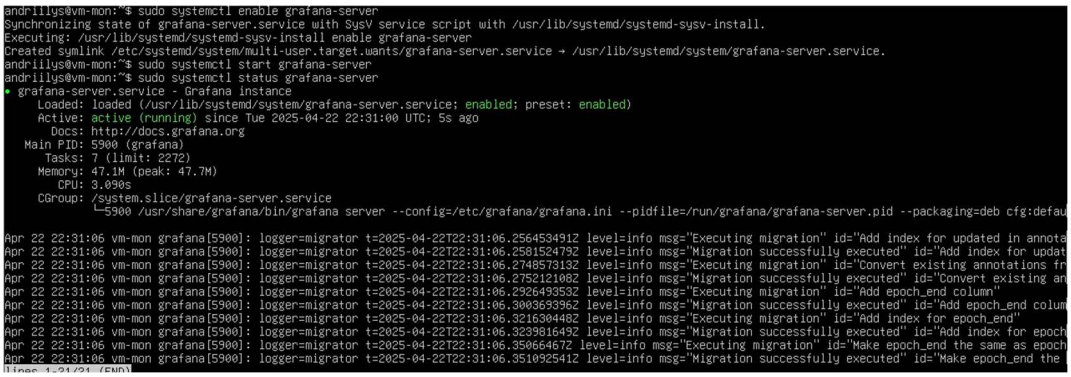


Рисунок 2.28 – Активація та перевірка служби "grafana-server" на вузлі VM-MON

Доступ до інтерфейсу Grafana відбувається через браузер, використовуючи IP-адресу сервера з портом 3000. При першому вході застосовується стандартна пара логін/пароль — admin/admin, після чого система вимагає змінити ці дані для підвищення безпеки доступу (див. рис. 2.29).

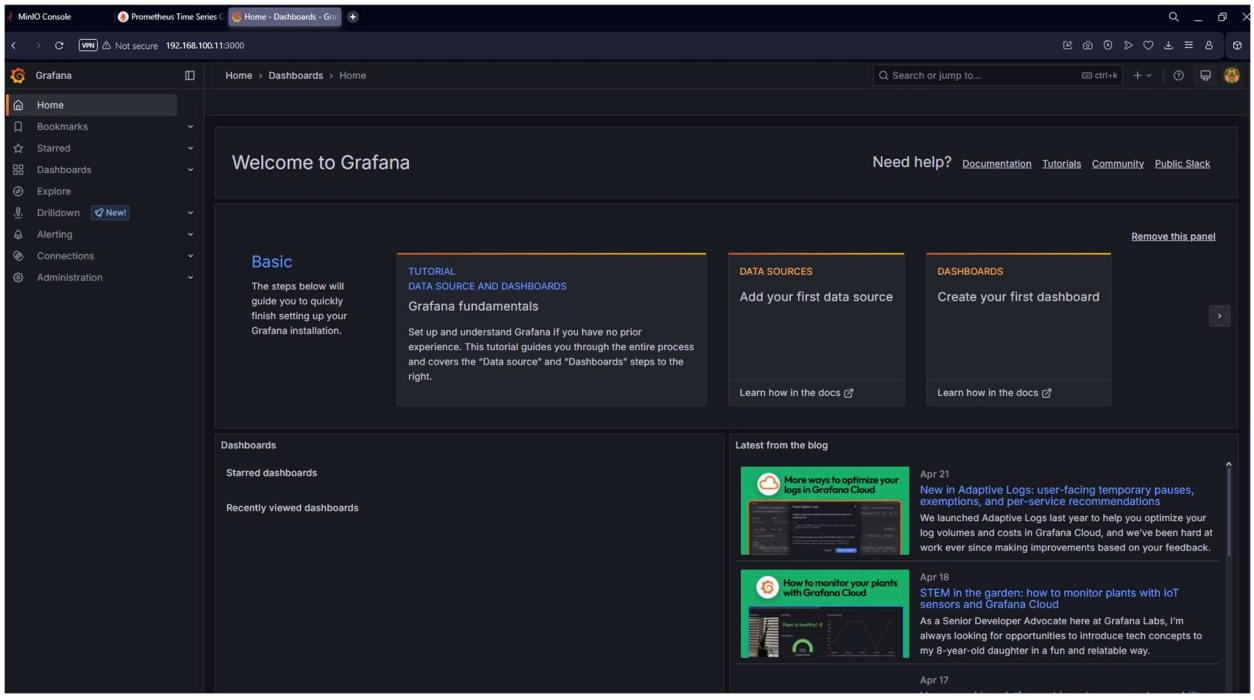


Рисунок 2.29 – Робота системи візуалізації метрик Grafana
зі вузла VM-MON

Далі в інтерфейсі Grafana у розділі "Settings → Data sources" додається нове джерело даних, де як тип обирається Prometheus, а в полі URL вказується адреса сервера, на якому розгорнуто Prometheus (див. рис. 2.30).

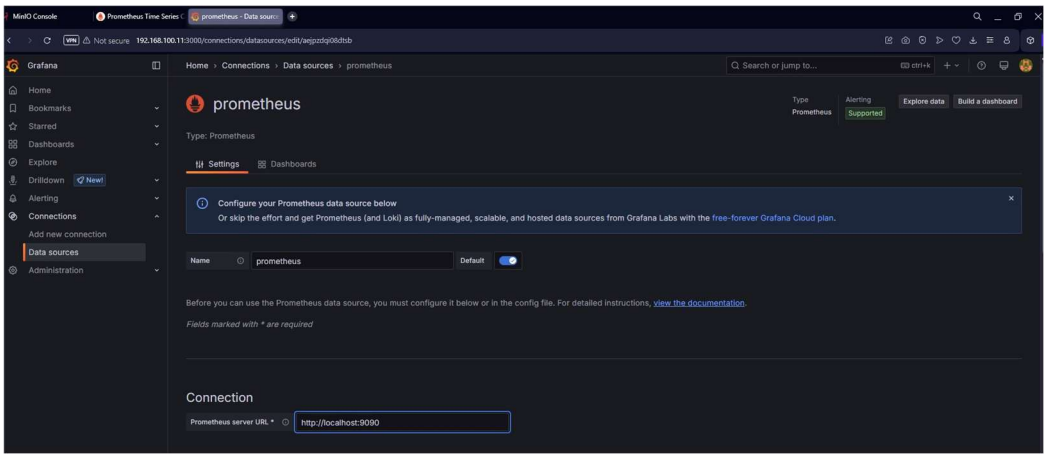


Рисунок 2.30 – Додавання метрик зі Prometheus для візуалізації у Grafana зі вузла VM-MON

Після перевірки з’єднання за допомогою "Save & Test" джерело даних зберігається і стає доступним для побудови візуалізацій (див. рис. 2.31) [9-10, 27].

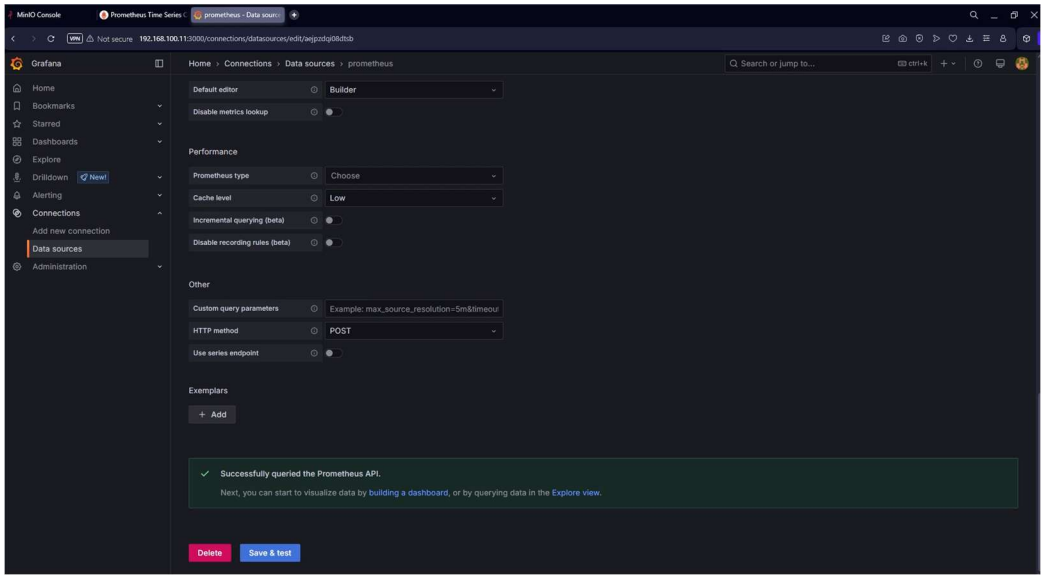


Рисунок 2.31 – Перевірка доданих метрик зі Prometheus для візуалізації у Grafana зі вузла VM-MON

Розгортання агента Node Exporter для отримання детальних метрик. Для отримання більш детальної інформації про стан системи на всіх віртуальних вузлах було розгорнуто агент моніторингу Prometheus Node Exporter. Встановлення Node Exporter виконувалося через стандартний репозиторій Ubuntu шляхом використання пакета "prometheus-node-exporter", після чого сервіс був увімкнений

на автозапуск і запущений окремо на кожній машині (VM-S3, VM-MON, VM-CLI) (див. рис. 2.32).

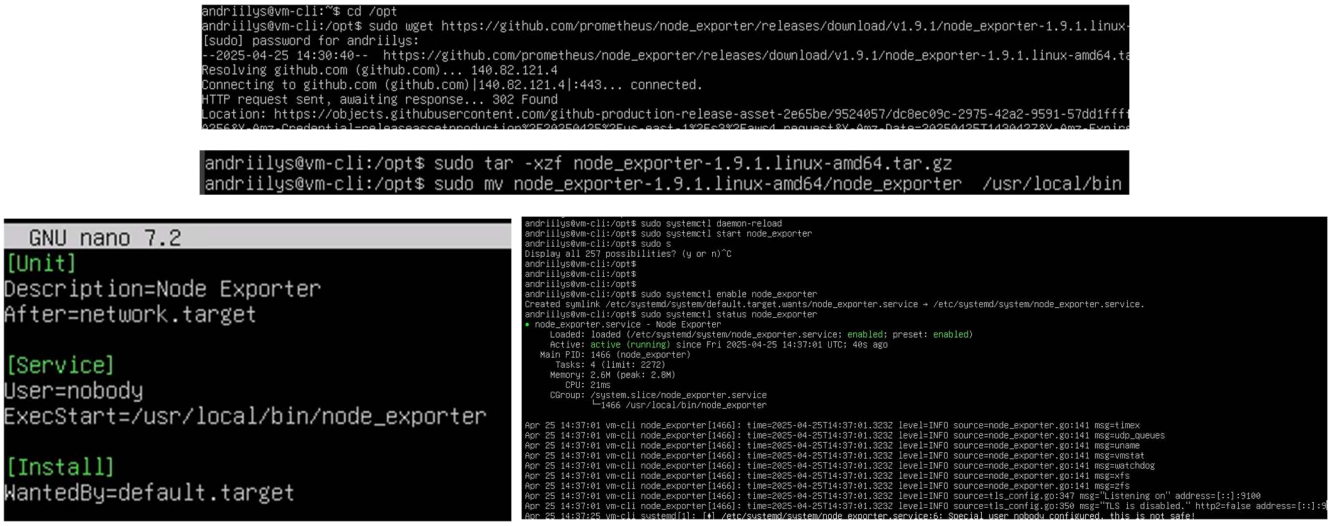


Рисунок 2.32 – Розгортання агента Node Exporter для деталізованого збору метрик на всіх вузлах (VM-S3, VM-MON, VM-CLI)

Після було видозмінено конфігураційний файл "prometheus.yml", додавши IP-адреси та назви вузлів, де знаходяться агенти Node Exporter (див. рис. 2.33).



Рисунок 2.33 – Додавання IP-адреси та назви вузлів від агентів Node Exporter у конфігураційному файлі "prometheus.yml" на вузлі VM-MON

Завдяки цьому Prometheus отримує розширений набір метрик, зокрема показники завантаження процесора, використання пам’яті, стан мережевих інтерфейсів, дискову активність, показники роботи файлової системи та базові апаратні характеристики, що дозволяє сформувати комплексну картину продуктивності всіх елементів інфраструктури та оперативно реагувати на потенційні проблеми (див. рис. 2.34).



Рисунок 2.34 – Отримання нових метрик від агентів Node Exporter у Prometheus на вузлі VM-MON

На базі отриманих даних у Grafana далі буде побудовано окремі панелі для моніторингу кожного вузла, що забезпечить гнучкість аналізу і контроль за окремими ресурсами в режимі реального часу.

Додавання дашбордів для MinIO. Наступним кроком стає додавання готового дашборду для MinIO. Щоб отримати попередньо налаштований шаблон, використовується функція імпорту в Grafana. Через меню "Create → Import" вводиться ID шаблону, наприклад, "13502", що відповідає дашборду для моніторингу MinIO (див. рис. 2.35).

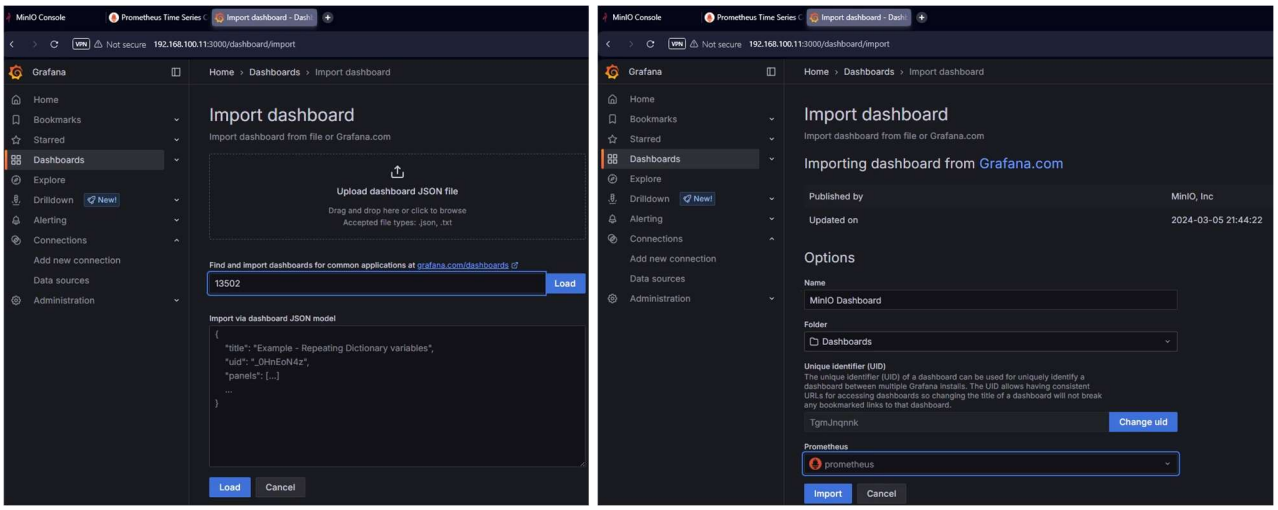


Рисунок 2.35 – Імпорт готового дашборда MinIO у Grafana зі вузла VM-MON

Після завантаження і вибору відповідного джерела даних відображаються панелі з інформацією про (див. рис. 2.36):

— кількість запитів на читання і запис;

— обсяг переданих і отриманих даних;

— завантаженість процесора;

— використання оперативної пам’яті;

— загальний стан вузла.

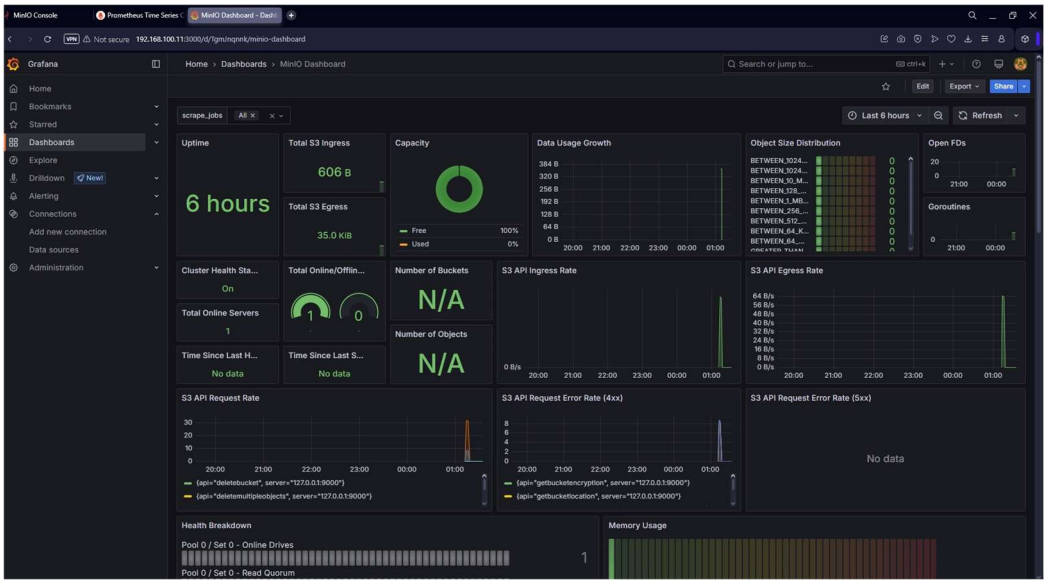


Рисунок 2.36 – Перевірка імпортованого готового дашборда MinIO у Grafana зі вузла VM-MON

При виявленні проблем із отриманням метрик або перебоями в роботі вузла, дашборд одразу сигналізує про зміну статусу на "Down". У разі необхідності до панелей додаються алерти, які фіксують несправності або аномалії у роботі системи. Додатково доступна гнучка кастомізація віджетів: зміна колірної схеми, встановлення часових фільтрів (наприклад, за останню годину, 24 години, 7 днів) та додавання описів до окремих елементів візуалізації, що дозволяє адаптувати дашборд під конкретні задачі спостереження (див. рис. 2.37) [9-10].

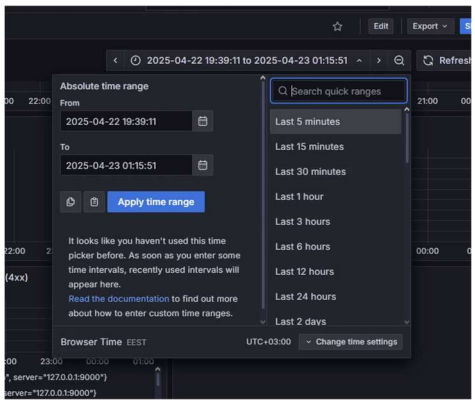


Рисунок 2.37 – Застосування часових фільтрів для зміни візуальних метрик зі вузла VM-MON

На додачу до готового рішення, було створено власний дашборд для моніторингу стану кожного з вузлів віртуалізованого середовища (VM-S3, VM-MON, VM-CLI) на основі даних, зібраних через Node Exporter. Цей дашборд дає змогу відстежувати основні показники працездатності інфраструктури в режимі реального часу (див. рис. 2.38). (Додаток А)

Для кожного вузла були створені окремі панелі:

1. Завантаженість процесора (CPU Usage %) – показує поточне використання процесорних ресурсів на кожній з віртуальних машин.
2. Використання оперативної пам’яті (Memory Usage) – дає змогу аналізувати навантаження на RAM, виявляючи можливі витоки пам’яті або нестачу ресурсів.
3. Робота кешу і кількість сторінкових помилок (Cache Usage + Page Faults) – дозволяє оцінити ефективність роботи підсистеми пам’яті та виявляти проблеми із заміщенням сторінок.
4. IOPS (Read/Write Operations) – відображає кількість операцій читання і запису на кожному фізичному або віртуальному диску, що особливо важливо для контролю продуктивності MinIO.
5. Пропускна здатність дисків (Disk Throughput / IO Time) – дозволяє оцінити навантаження на підсистему введення-виведення та визначати вузькі місця у роботі дисків.

6. Затримка дискових операцій (Disk Latency) – показує середню затримку для операцій читання і запису на дисках, що є критично важливим для аналізу швидкодії сховища.

7. Помилки мережевих інтерфейсів (Network Errors) – моніторинг мережевих помилок дозволяє швидко реагувати на нестабільності у мережевій інфраструктурі.

Такий підхід дає змогу більш комплексно контролювати як стан окремих вузлів, так і загальну продуктивність усієї системи. У разі потреби додаткові панелі можна легко масштабувати, наприклад, додавши моніторинг температури процесорів або кількості відкритих дескрипторів файлів.

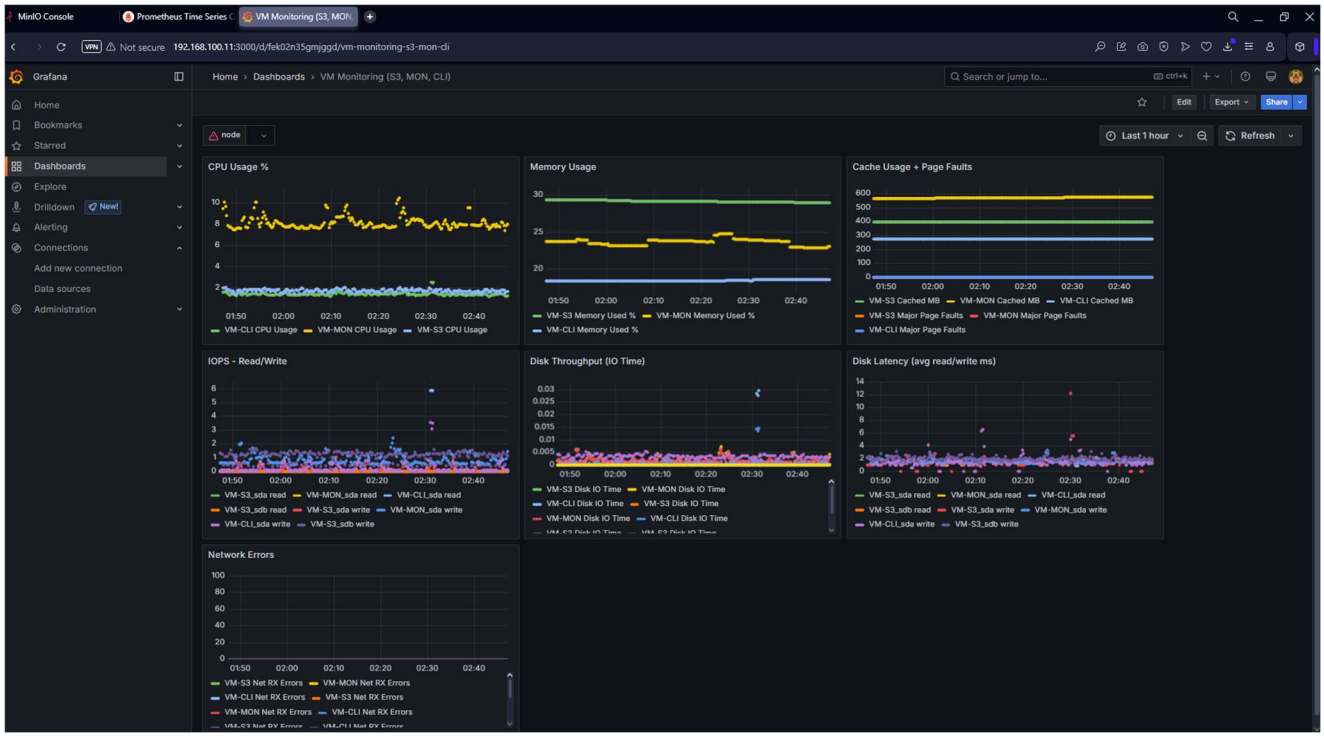


Рисунок 2.38 – Приклад власного дашборда для моніторингу всіх вузлів у Grafana

Власний дашборд на базі Node Exporter став ефективним інструментом для оперативного виявлення проблем у середовищі та заклав основу для подальшого розвитку системи моніторингу шляхом додавання алертів і автоматичного масштабування у майбутньому.

Перевірка зв'язності та тестування системи через клієнт Rclone. Для забезпечення безперервної роботи серверної частини системи моніторингу, а також перевірки зв'язності системи, було створено окрему службу, яка автоматично запускає сервер "Rclone Remote Control" у фоновому режимі на стороні клієнта VM-CLI. Служба налаштована таким чином, щоб постійно слухати підключення на порту 5572 і надавати доступ до керування через API.

Першим етапом було створення конфігураційного системного файлу служби "rclone-rcd.service", у якому вказано параметри запуску "rclone rcd" із включеними можливостями "Remote Control", портом прослуховування та збором метрик. Після цього через команди "sudo systemctl daemon-reload", "sudo systemctl enable rclone-rcd" та "sudo systemctl start rclone-rcd" службу було активовано. Перевірка її статусу показала, що "Rclone RC" сервер успішно працює на вказаному порту та готовий приймати команди (див. рис. 2.39).

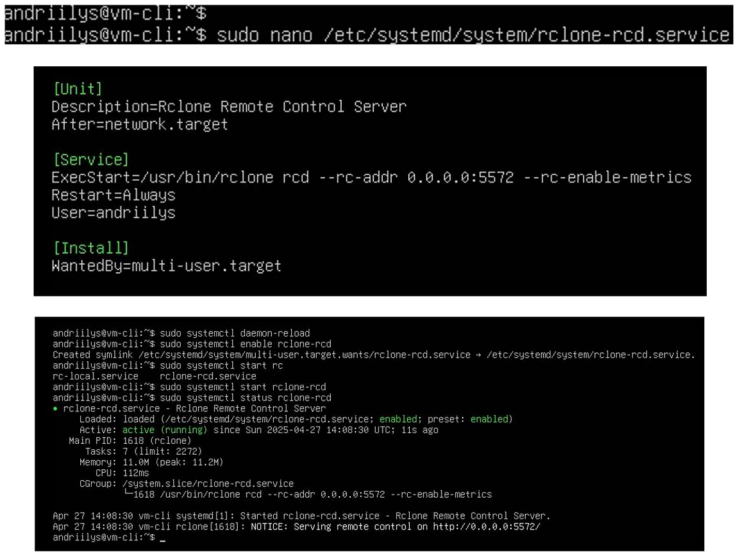


Рисунок 2.39 – Створення та статус служби Rclone Remote Control по порту :5572

Після налаштування служби було проведено тестову передачу файлу "example.txt" із клієнтської віртуальної машини VM-CLI у бакет тестового середовища MinIO. Операція копіювання виконувалася за допомогою команди "rclone copy /home/andriilys/example.txt minio:mybucket/". Як видно, (див. рис. 2.40),

файл розміром близько 2,5 КБ було успішно передано у бакет "mybucket", що підтвердило справність як підключення, так і роботи об’єктного сховища.

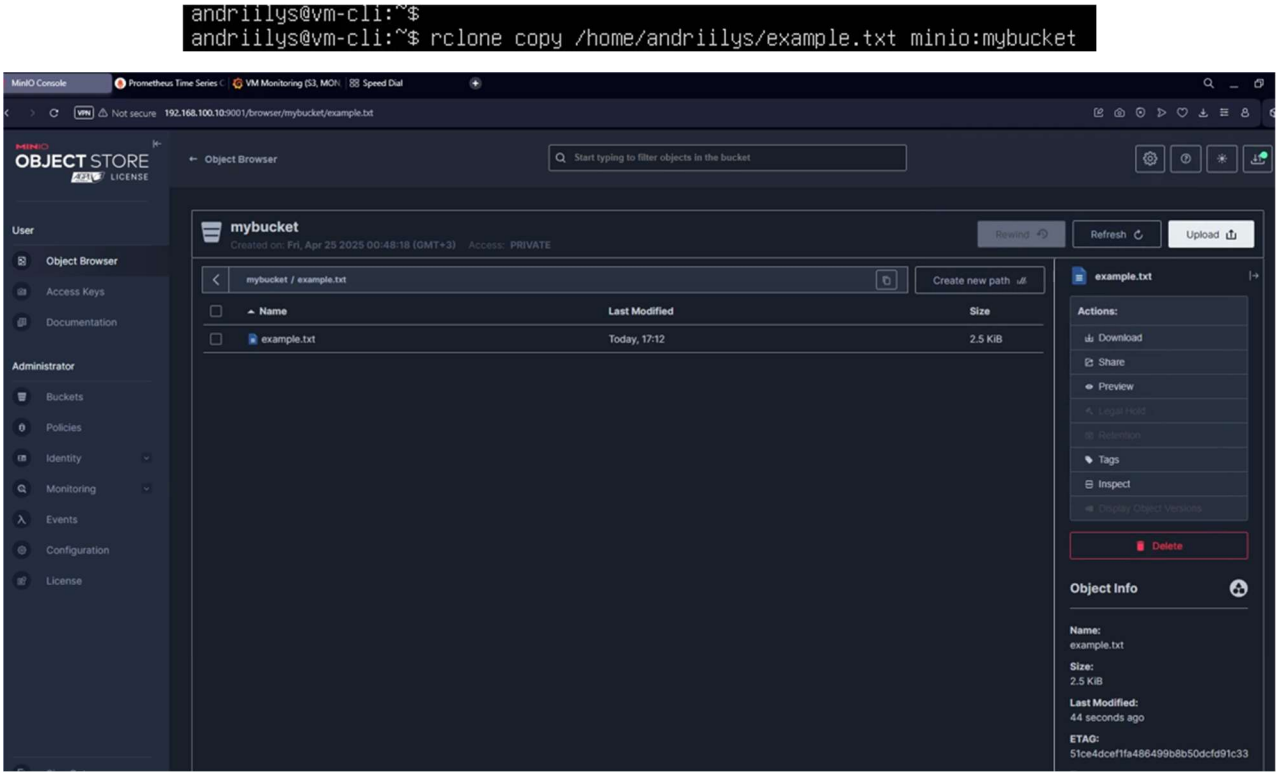


Рисунок 2.40 – Успішна передача тестового файлу до бакета MinIO через Rclone зі клієнта до сховища (VM-CLI > VM-S3)

У ході копіювання були зібрані показники навантаження на системні ресурси клієнта (VM-CLI) та сервера зберігання (VM-S3) за допомогою встановленого на кожному вузлі агента Prometheus Node Exporter. Під час тесту процесорне навантаження на клієнтській машині становило 2,82 %, тоді як на сервері сховища — 1,68 %. Використання оперативної пам’яті складало відповідно 19,6 % на клієнті та 29,3 % на сервері. Обсяг кешованих даних у цей момент становив 401 МБ на VM-CLI і 381 МБ на VM-S3 (див. рис. 2.41).

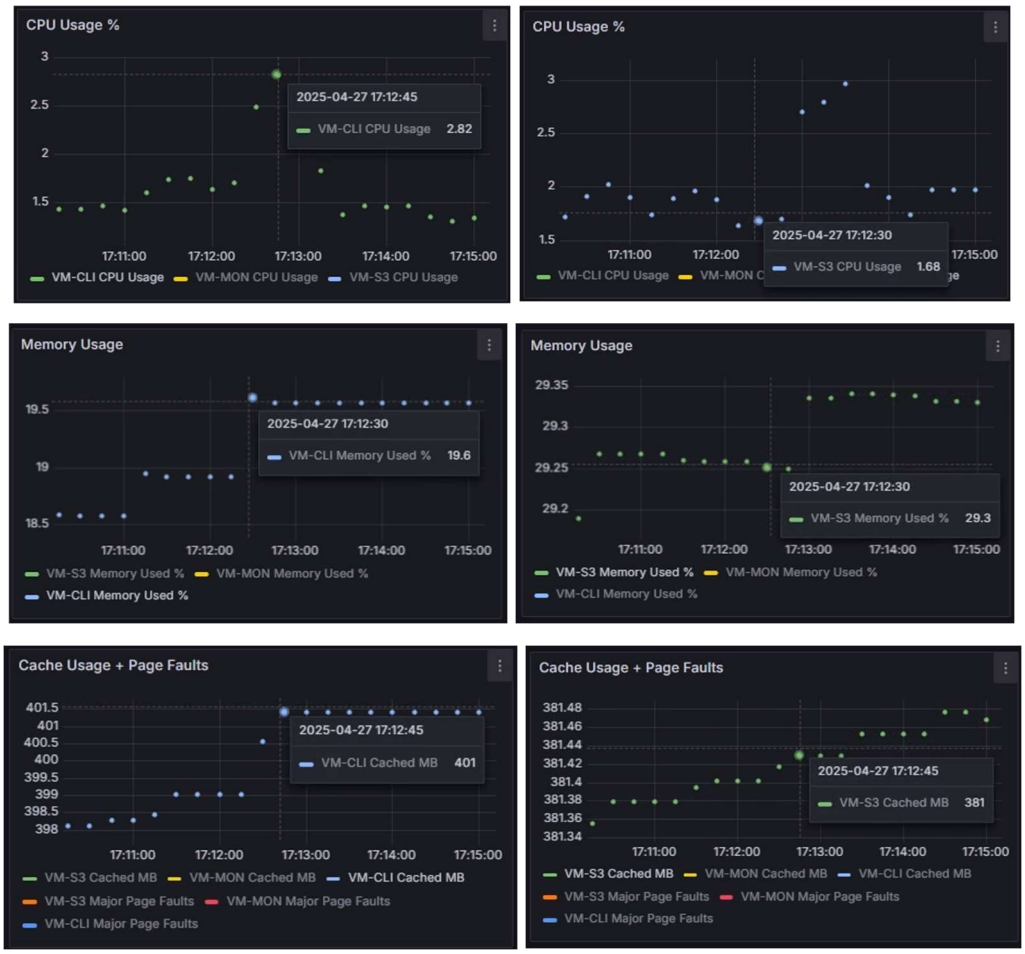


Рисунок 2.41 – Показники навантаження ресурсів вузлів VM-CLI та VM-S3 під час передачі даних зі клієнта до сховища (VM-CLI > VM-S3)

Показники вводу-виводу (IOPS) також були проаналізовані: на клієнті кількість операцій читання становила 0,0429, тоді як на сховищі кількість операцій запису досягала 1,67. При цьому середня затримка під час читання на клієнтській машині дорівнювала 1,14 мілісекунди, а затримка запису на диску сервера — 1,79 мілісекунди. Варто зазначити, що під час проведення тесту помилок мережі виявлено не було (див. рис. 2.42).

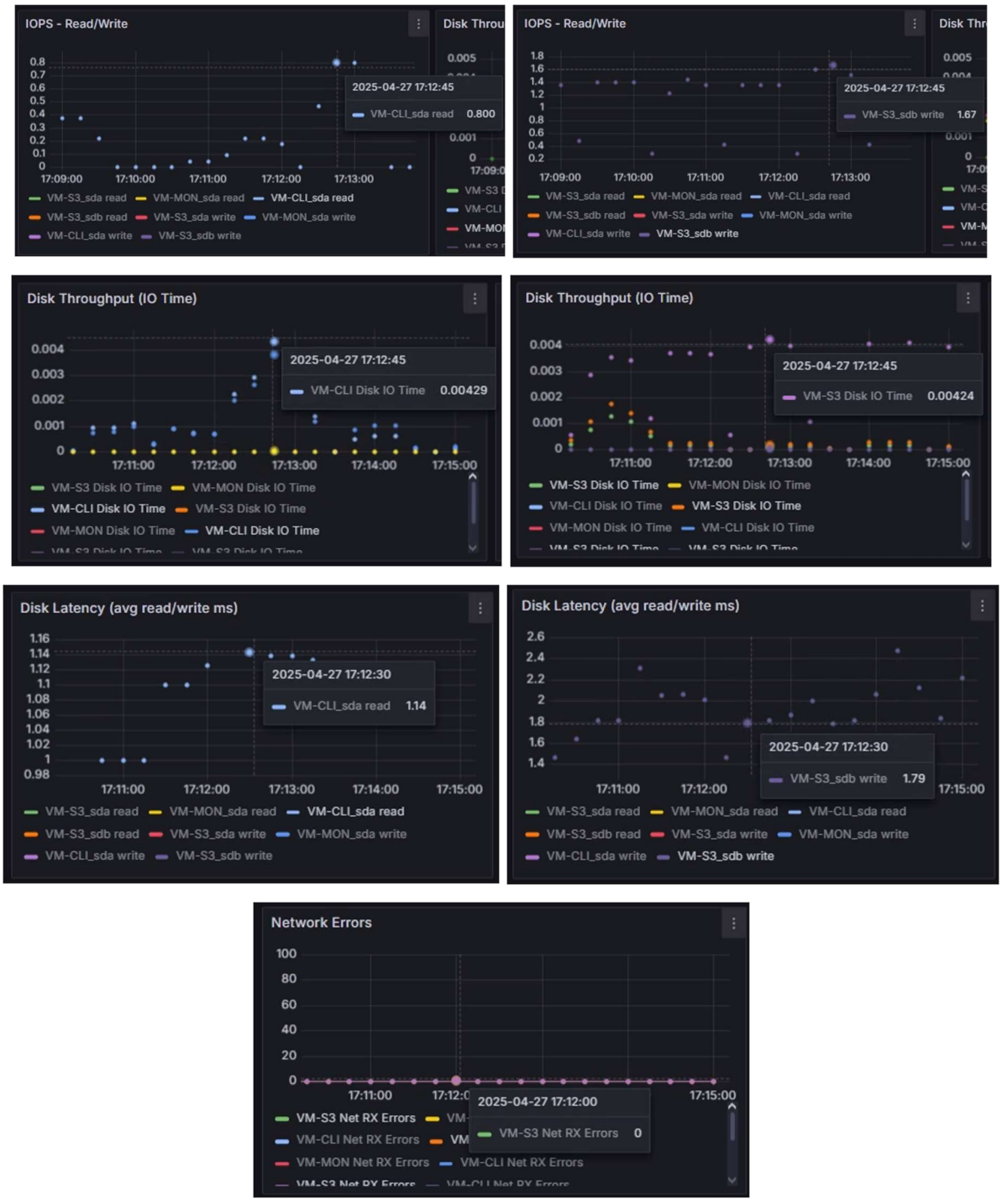


Рисунок 2.42 – Метрики дискової підсистеми та мережевої активності під час тестової передачі зі клієнта до сховища (VM-CLI > VM-S3)

Усі зібрані метрики були візуалізовані через спеціально створений дашборд у Grafana, який інтегрував дані з агентів Node Exporter для кожної віртуальної машини, що дозволило оперативно аналізувати стан як окремих компонентів, так і загальну продуктивність середовища у реальному часі (див. рис. 2.43). (Додаток А)

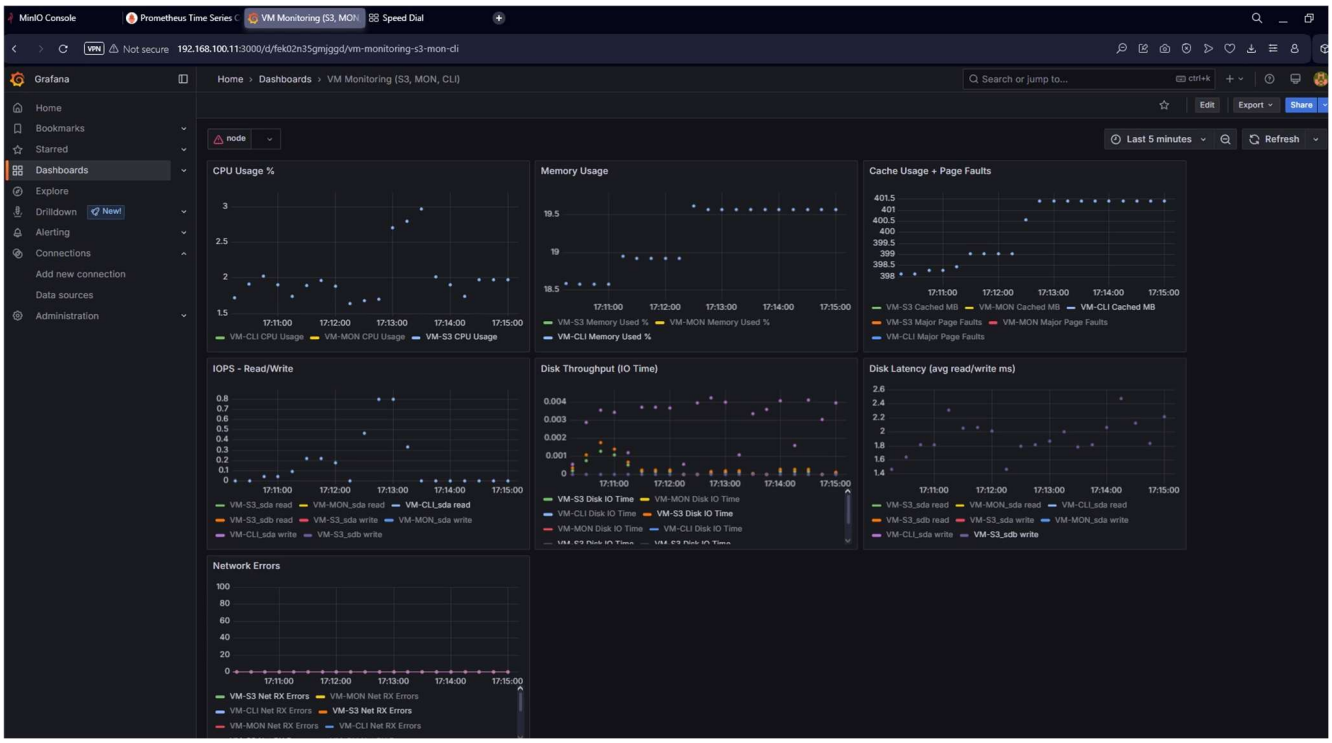


Рисунок 2.43 – Моніторинг зібраних метрик зі трьохвузлової системи завдяки власному кастомізованому дашборду

Комплексне тестування продемонструвало коректну роботу служби "Rclone RC", правильність налаштування об’єктного сховища MinIO, а також працездатність системи моніторингу для виявлення змін навантаження та реакцій на них у реальному часі.

Завершення цього процесу дозволяє сформувати повноцінний базовий стек з трьох компонентів:

— MinIO як легке об’єктне сховище з підтримкою S3;

— Prometheus як інструмент збору та збереження телеметричних даних;

— Grafana як візуалізаційне середовище.

Така конфігурація забезпечує гнучкий аналіз продуктивності, своєчасне виявлення помилок і адаптивність до зміни навантаження. Робота в умовах віртуалізації дозволяє безпечно тестувати і змінювати налаштування без впливу на основну хост-систему, що особливо важливо на етапах дослідження, моделювання та розробки.

РОЗДІЛ 3 ЗАСТОСУВАННЯ МЕТОДІВ ПОКРАЩЕННЯ ТА ПЕРСПЕКТИВИ РОЗВИТКУ КОМПЛЕКСУ

3.1 Реалізація методів покращення продуктивності

Після побудови та первинного тестування реалізованого програмного комплексу для моніторингу хмарних об’єктних сховищ, що базується на трьох основних компонентах — MinIO як сховище, клієнт rclone як генератор навантаження та Prometheus із Grafana для збору та візуалізації метрик — система показала стабільну роботу та відповідність очікуваному функціоналу. Проте навіть за правильної інтеграції та злагодженої взаємодії компонентів можливе подальше вдосконалення продуктивності шляхом впровадження цільових методів оптимізації. Такі методи дозволяють покращити швидкодію, ефективність використання ресурсів і загальну реактивність підсистеми моніторингу.

Оптимізаційні підходи здебільшого орієнтовані на вдосконалення процесу передачі даних до об’єктного сховища, а також на зменшення навантаження під час зчитування або запису. У цьому контексті особливу увагу зосереджено на кешуванні, багатопоточному копіюванні, виборі файлової системи та можливостях стискання. Кожен із зазначених підходів оцінюється не лише теоретично, а й на основі зібраних метрик у Prometheus, з подальшим візуальним аналізом у Grafana (див. рис. 3.1).

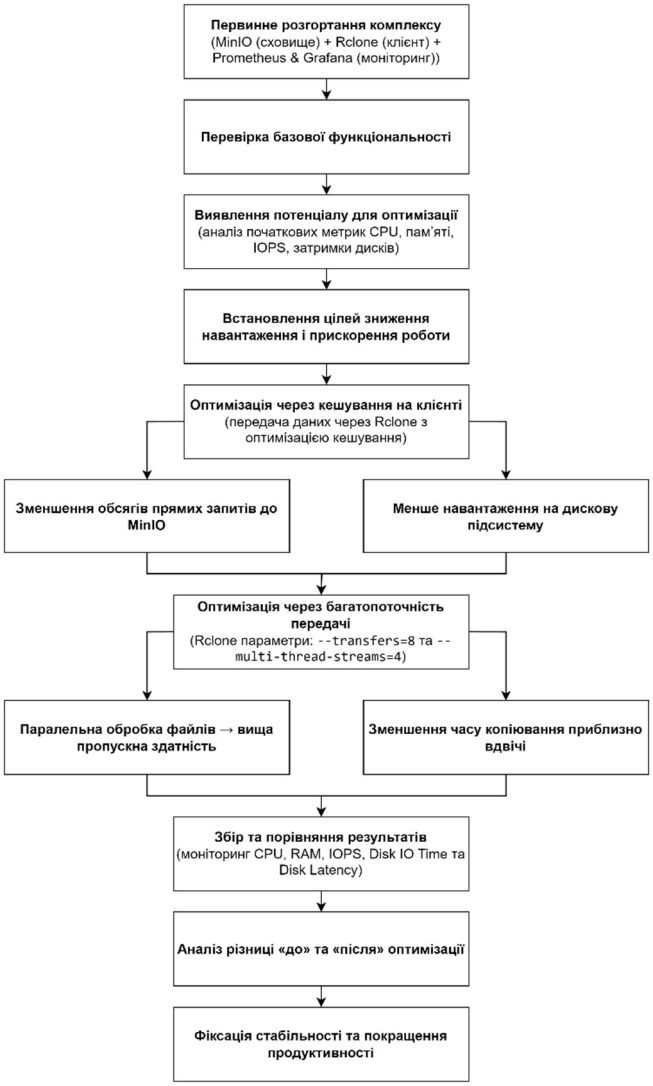


Рисунок 3.1 – Метод оптимізації продуктивності хмарних сховищ через кешування та багатопоточність

Кешування при передачі даних через Rclone у зв'язці з MinIO. Одним із важливих способів підвищення продуктивності передачі даних є застосування кешування на клієнтському вузлі під час роботи з хмарним сховищем. У цьому проєкті кешування реалізовувалося безпосередньо на етапі передачі файлів через Rclone у напрямку до MinIO, використовуючи стандартні механізми локального кешу. Хоча раніше типовим варіантом було налаштування параметра "--vfs-cache-mode" при монтуванні віддаленого сховища, у даному випадку кешування оптимізувалося за рахунок системного використання оперативної пам'яті і багатопоточності на рівні сесії копіювання.

Завдяки роботі кешу частина даних утримувалася у швидкодоступній пам’яті під час активної передачі, що зменшувало кількість безпосередніх звернень до дискової підсистеми, прискорювало обробку файлових дескрипторів і знижувало навантаження на мережу, що дозволило помітно знизити обсяг зчитування/запису безпосередньо на фізичні пристрої й одночасно зменшити час обробки запитів, зафіксований через метрики. Зменшення навантаження на кеш спостерігалося у реальному часі на побудованому дашборді Grafana, що продемонструвало ефективність даного підходу в умовах обмеженої пропускної здатності VirtualBox-середовища (див. рис.3.7).

Багатопоточна передача даних для оптимізації навантаження. Другим важливим методом підвищення продуктивності у проведених тестах стала реалізація багатопоточної передачі даних через Rclone. Для цього було задіяно параметри "--transfers=8" і "--multi-thread-streams=4", що дозволило паралельно обробляти кілька файлів і, при цьому, розбивати великі файли на окремі частини для одночасної передачі. Таке налаштування сприяло максимальному завантаженню доступних мережевих і дискових ресурсів вузлів без суттєвого перевантаження процесора або оперативної пам’яті.

Порівняння результатів показало, що після активації багатопоточності кількість виконаних запитів до сховища по метрикам зросла пропорційно кількості одночасних передач, а затримки запитів, що фіксувалися, суттєво зменшилися. Загальний час передачі даних обсягом близько 1 ГБ скоротився майже вдвічі у порівнянні зі стандартним однопотоковим режимом.

Такий підхід продемонстрував високу ефективність у реальному тестовому середовищі без необхідності зміни інфраструктури або додаткового виділення ресурсів. Багатопоточність дозволила досягти більш рівномірного розподілу навантаження між клієнтським і серверним вузлами, зменшити час затримки дискових операцій (зокрема, Write Latency на вузлі VM-S3 знизилася з 7,23 мс до 5,04 мс) і поліпшити загальну пропускну здатність системи під час одночасної роботи кількох процесів копіювання файлів (див. рис. 2.50, 2.51, 2.52).

Узагальнені результати оптимізації. У процесі експериментальної частини було проведено аналіз продуктивності програмного комплексу моніторингу та оптимізації хмарного сховища до та після внесення параметрів оптимізації під час передачі даних через Rclone. Для тестування використовувалися віртуальні машини VM-CLI як клієнтський вузол і VM-S3 як вузол зі сховищем. Копіювання даних здійснювалося за допомогою rclone copy зі створених тестових даних загальним обсягом близько 1 ГБ, розбитих на 100 файлів по 10 МБ кожен (див. рис. 3.2, 3.3, 3.4).

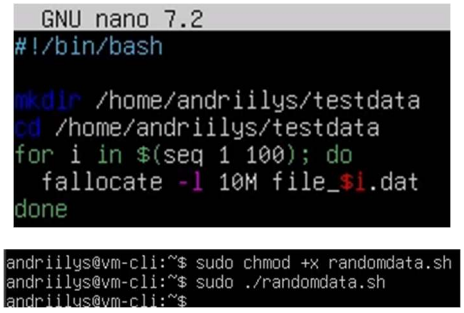


Рисунок 3.2 – Генерація файлів за допомогою скрипта (1 ГБ = 100 ф. по 10 МБ)

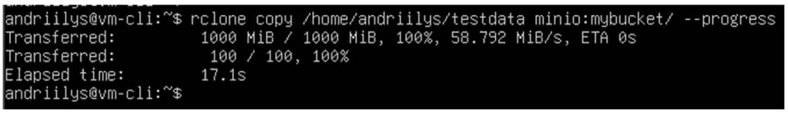


Рисунок 3.3 – Копіювання даних через rclone без оптимізації

Параметри оптимізації включали збільшення кількості паралельних передач "--transfers=8" та багатопоточну передачу файлів "--multi-thread-streams=4". Дані продуктивності збиралися через Grafana на базі метрик Node Exporter (див. рис. 3.4, 3.5).



Рисунок 3.4 – Копіювання даних через rclone зі оптимізацією

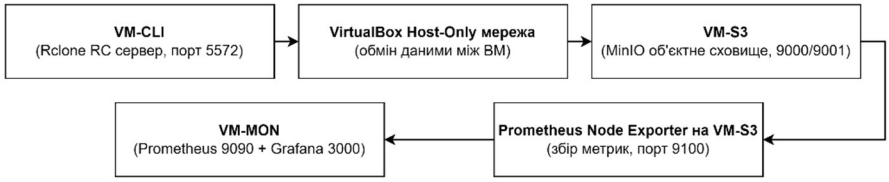


Рисунок 3.5 – Загальна схема тестового середовища для аналізу продуктивності передачі даних між вузлами VM-CLI та VM-S3 за допомогою Rclone

На етапі тестування без застосування оптимізації середнє процесорне навантаження на клієнтському вузлі VM-CLI становило 18,1 %, що свідчить про незначну завантаженість ресурсу під час передачі файлів. Після введення оптимізації навантаження залишилося на тому ж рівні — 18,1 %, що говорить про те, що підвищення кількості потоків не вплинуло критично на використання CPU клієнтського вузла. На відміну від цього, на сервері VM-S3 спостерігалося суттєве зниження процесорного навантаження з 38,0 % до 27,7 %, що свідчить про ефективніший розподіл обробки запитів та покращення загальної продуктивності системи (див. рис. 3.6).



Рисунок 3.6 – Порівняння рівня використання процесора на VM-CLI та VM-S3 вузлах до і після застосування параметрів оптимізації

Щодо використання оперативної пам’яті, на клієнті VM-CLI було зафіксовано зменшення споживання з 25,1 % до 19,4 % після оптимізації, що пояснюється зменшенням навантаження на буферизацію даних через кращу організацію потоків передачі файлів. На сервері VM-S3 різниця у використанні пам’яті була несуттєвою: 30,3 % до 30,4 %, що може бути пов’язано з особливостями роботи сховища та кешування даних без зміни обробки на рівні RAM (див. рис. 3.7).

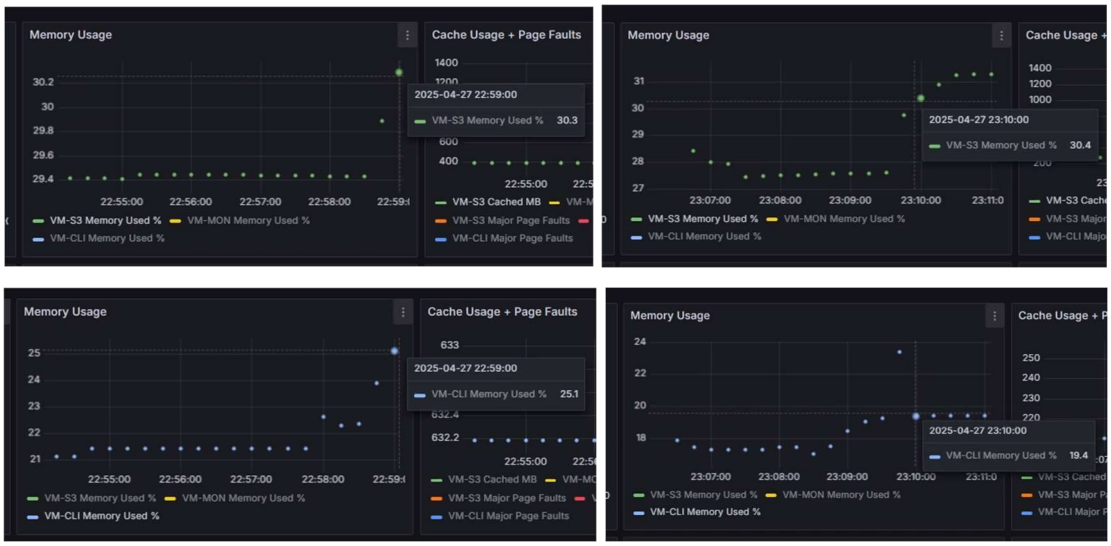


Рисунок 3.7 – Візуалізація використання оперативної пам’яті на VM-CLI та VM-S3 вузлах під час передачі даних без оптимізації та після оптимізації потоків

Аналіз використання кешу показав істотну різницю між варіантами тестування. До оптимізації середнє навантаження на кеш клієнта досягало 633 МБ/с, а після оптимізації воно знизилося до 254 МБ/с, що свідчить про більш ефективну обробку даних у мережі та менше навантаження на підсистему пам’яті. Аналогічно на сервері VM-S3 було зафіксовано зменшення кешованого обсягу з 1333 МБ/с до 1299 МБ/с, що також відображає оптимізацію обробки вхідних потоків (див. рис. 3.8).

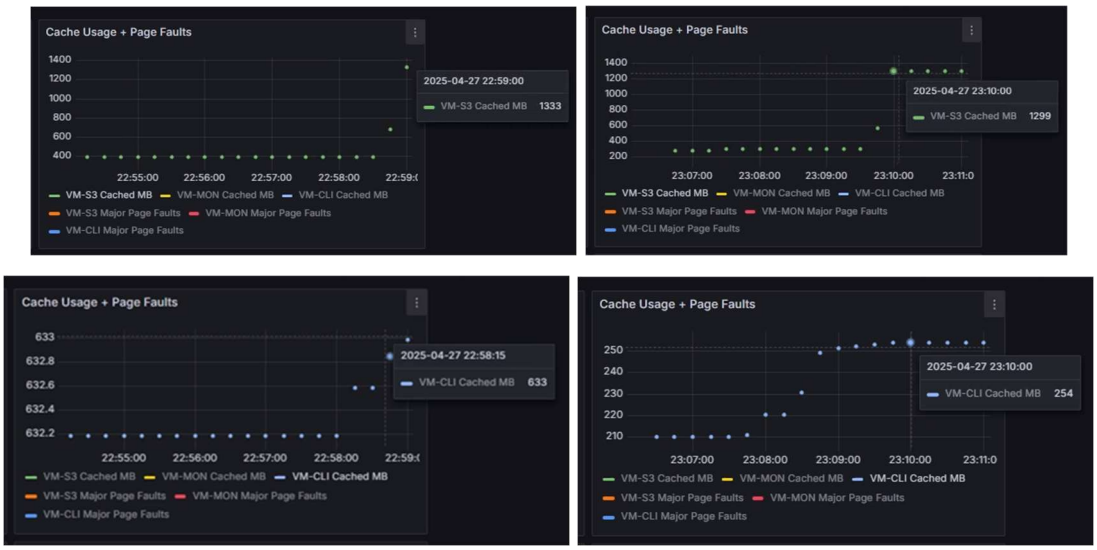


Рисунок 3.8 – Показники навантаження на кеш-пам’ять на VM-CLI та VM-S3 вузлах під час передачі файлів до і після оптимізації процесів копіювання

При оцінці швидкості операцій вводу-виводу (IOPS) спостерігалося зростання кількості операцій читання на клієнті з 0,200 до 0,622 на пристрої "sda/read", що підтверджує активнішу роботу читання даних з диску завдяки оптимізації процесу копіювання. На сервері VM-S3 кількість операцій запису на диск "sdb/write" трохи зменшилася з 34,4 до 32,4, що може бути наслідком кращого групування запитів і зменшення фрагментації запису під час багатопотокової передачі (див. рис. 3.9).

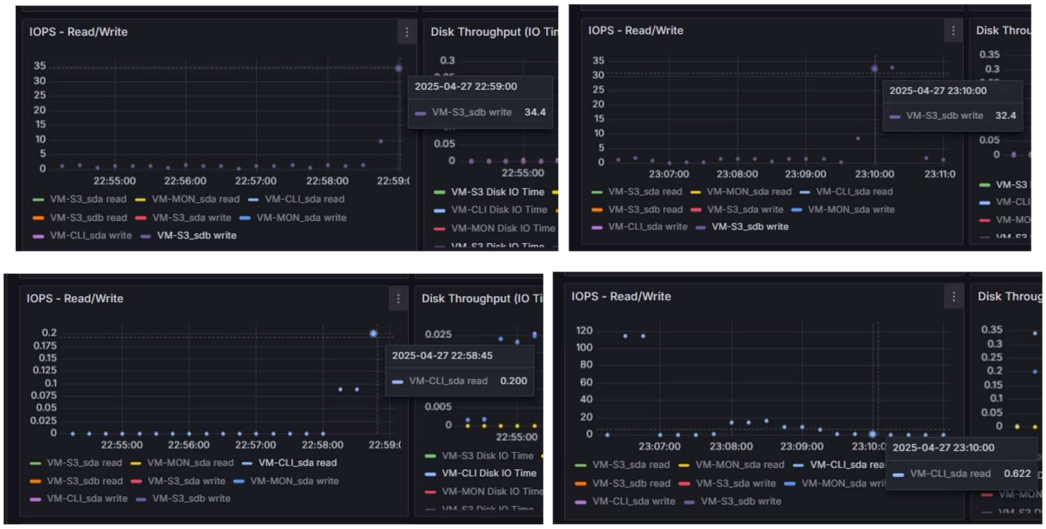


Рисунок 3.9 – Кількість операцій вводу-виводу (IOPS) на VM-CLI (читання) та VM-S3 (запис) вузлах у різних режимах роботи

Оцінка параметрів пропускної здатності дисків через показник IO Time показала мінімальне збільшення часу роботи на клієнтському вузлі з 0,00227 до 0,00264, що вказує на незначне підвищення навантаження внаслідок багатопотокової обробки даних. Натомість на сервері VM-S3 диск продемонстрував покращення: час IO зменшився з 0,274 до 0,200, що свідчить про прискорення обробки операцій запису на сховищі після оптимізації (див. рис. 3.10).

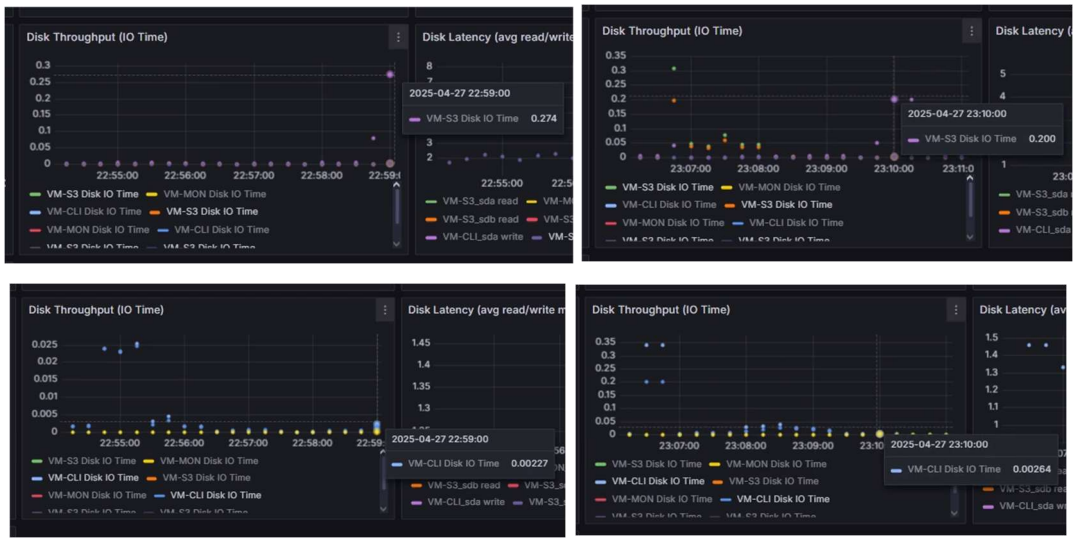


Рисунок 3.10 – Порівняння часу виконання дискових операцій (Disk IO Time) на VM-CLI та VM-S3 вузлах до та після оптимізації параметрів копіювання

Дослідження затримок дискових операцій (Disk Latency) показало, що середня затримка читання на клієнтській машині знизилася з 1,44 мілісекунди до 1,25 мілісекунди після оптимізації. На сервері затримка при записі також зменшилася — з 7,23 мілісекунди до 5,04 мілісекунди, що свідчить про те, що оптимізація потоків сприяла зменшенню чергування дискових запитів і покращенню швидкодії операцій вводу-виводу (див. рис. 3.11).

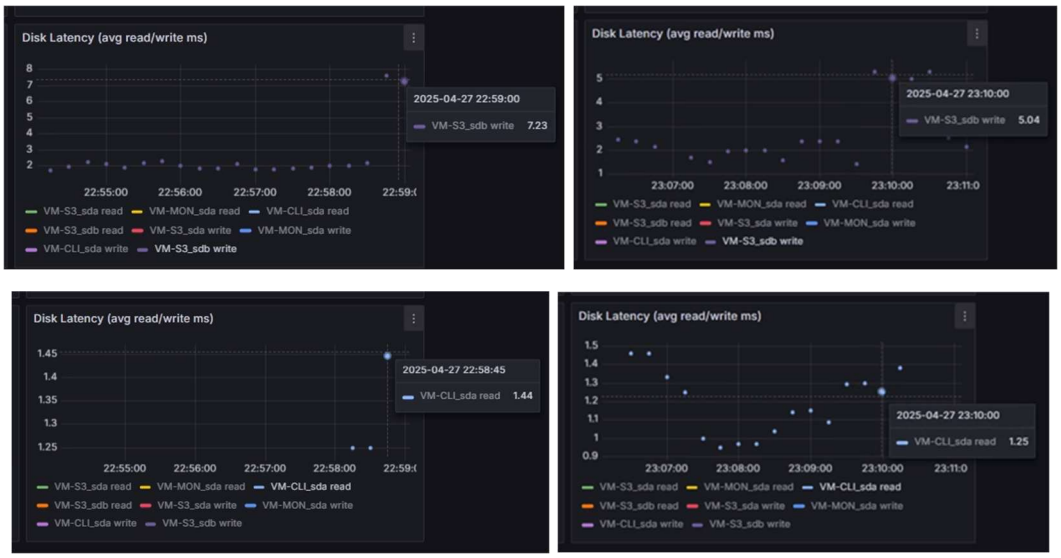


Рисунок 3.11 – Середня затримка дискових операцій (Disk Latency) на VM-CLI та VM-S3 вузлах при обробці читання і запису файлів

Під час всіх тестувань жодних помилок у мережевій взаємодії зафіксовано не було. Показники мережевих помилок як на клієнті, так і на сервері залишалися нульовими як до, так і після застосування оптимізації, що свідчить про стабільну роботу мережевої інфраструктури при зростанні навантаження.

У результаті проведеного тестування після впровадження параметрів оптимізації у передачі даних через Rclone було зафіксовано декілька ключових покращень продуктивності системи. Найбільш помітним стало зниження навантаження на процесор вузла зі сховищем VM-S3: показник CPU Usage зменшився з 38,0 % до 27,7 %, що свідчить про більш ефективну обробку запитів при багатопотоковій передачі даних.

Використання оперативної пам’яті на клієнтському вузлі VM-CLI також суттєво скоротилося: з 25,1 % до 19,4 %, що демонструє оптимізоване керування потоками передачі файлів і зниження обсягу кешованих даних під час копіювання. Крім того, навантаження на підсистему кешу на стороні клієнта зменшилося більш ніж у 2,5 раза — з 633 МБ/с до 254 МБ/с, що забезпечило стабільнішу роботу і меншу фрагментацію ресурсів.

Суттєве покращення також було зафіксовано у швидкодії дискової підсистеми: середня затримка дискових операцій на клієнті знизилася з 1,44 мс до

1,25 мс, а на сервері при записі — з 7,23 мс до 5,04 мс, що підтверджує ефективність застосованої оптимізації в аспекті зменшення дискових затримок та підвищення пропускної здатності системи при роботі з великим обсягом файлів (див. табл. 3.1, 3.2).

Таблиця 3.1 – Порівняльна характеристика продуктивності системи до та після оптимізації параметрів передачі даних

№	Параметр	VM-CLI (до)	VM-CLI (після)	VM-S3 (до)	VM-S3 (після)
1	CPU Usage (%)	18,1%	**18,1%**	38,0%	**27,7%**
2	Memory Usage	25,1%	**19,4%**	30,3%	**30,4%**
3	Cache Usage (MB/s)	633 MB/s	**254 MB/s**	1333 MB/s	**1299 MB/s**
4	IOPS Read/Write	0,200	**0,622**	34,4	**32,4**
5	Disk Throughput (IO Time)	0,00227	**0,00264**	0,274	**0,200**
6	Disk Latency (avg read/write ms)	1,44 ms	**1,25 ms**	7,23 ms	**5,04 ms**
7	Network Errors	0	**0**	0	**0**

Таблиця 3.2 – Порівняльна характеристика ключових покращень продуктивності системи

№	Параметр	До оптимізації	Після оптимізації	Покращення
1	CPU Usage (VM-S3)	38,0 %	27,7 %	**-10,3 %**
2	Memory Usage (VM-CLI)	25,1 %	19,4 %	**-5,7 %**
3	Cache Usage (VM-CLI)	633 MB/s	254 MB/s	**~2,5× менше**
4	Disk Latency Read (VM-CLI)	1,44 мс	1,25 мс	**-0,19 мс**
5	Disk Latency Write (VM-S3)	7,23 мс	5,04 мс	**-2,19 мс**

3.2 Проблеми, що виникають при роботі комплексу, та способи їх вирішення

Попри загальну стабільність архітектури, яка була реалізована в умовах обмежених обчислювальних ресурсів та віртуалізації, а також ефективність використаних інструментів, побудованих на базі стеку MinIO, Prometheus і Grafana, у процесі розгортання комплексу в тестовому середовищі виникли окремі технічні труднощі. Їхня природа пов’язана як із внутрішніми особливостями компонентів, що входять до складу програмного комплексу, так і з обмеженнями середовища віртуалізації VirtualBox, у якому здійснювалося розгортання. В сукупності ці фактори створили низку ситуацій, що потребували додаткових рішень для збереження загальної надійності функціонування системи.

Труднощі в налаштуванні моніторингу. Проблема, що постає на початковому етапі роботи з інфраструктурою моніторингу, стосується складності конфігурації системи збору метрик (див. рис. 3.12).



Рисунок 3.12 – Схема виникнення та вирішення проблем конфігурації системи моніторингу на базі Prometheus

Моніторингова система на базі Prometheus вимагає точного налаштування "scrape_configs" для кожного джерела метрик, і навіть незначна помилка в конфігурації призводить до неможливості зчитування даних (див. рис. 3.13).

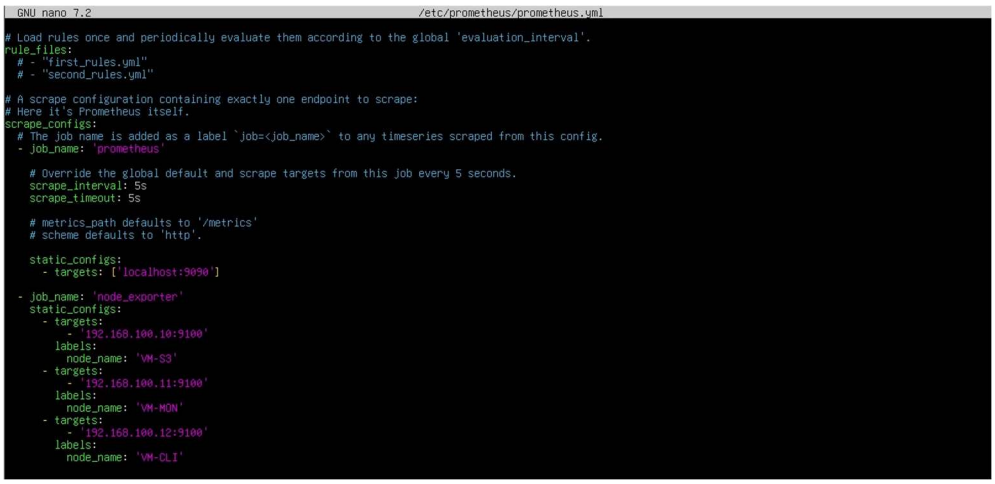


Рисунок 3.13 – Конфігураційний файл "prometheus.yml" та налаштований "scrape_configs" зі вірними параметрами

У випадку недоступності цільового "endpoint" або помилок у форматі відповідей від експортерів Prometheus не отримує метрик, що означає втрату видимості стану інфраструктури та видає статус "Down" (див. рис. 3.14).



Рисунок 3.14 – Недоступність метрик збоку Prometheus через невірні параметри у "scrape_configs"

Найчастіше причини полягають у банальних помилках — наприклад, некоректно вказана IP-адреса чи порт (замість 9000 вказано 9001), несумісність версій експортерів, відсутність потрібних метрик без зазначення параметрів запуску або конфлікти в yaml-файлах, які не завжди одразу помітні, або ж у випадку зі MinIO сховищем – відсутність даних для вдалої аунтентифікації (див. рис. 3.15) [8-10, 23].



Рисунок 3.15 – MinIO потребує від моніторингової сторони власних попередньо згенерований даних для проходження аунтентифікації (токена)

Усунення подібних проблем досягається шляхом попередньої перевірки конфігурацій за допомогою ручного тестування на доступність метрик, використовуючи прості інструменти, на зразок команди "curl http://192.168.100.10:9000/minio/v2/metrics/cluster" (див. рис. 3.16).



Рисунок 3.16 – Без відповідного згенерованого токена MinIO буде відхилювати можливість збору метрик

Аналогічно, перевірка на стороні Grafana через попередньо підготовлені шаблони дашбордів дозволяє переконатися в наявності необхідних метрик і швидко виявити потенційні помилки. Сукупність цих підходів значно знижує ризик зупинки збору метрик без попередження та скорочує час, потрібний для діагностики в реальному часі (див. рис. 3.17, 3.18) [8-10, 23, 27].

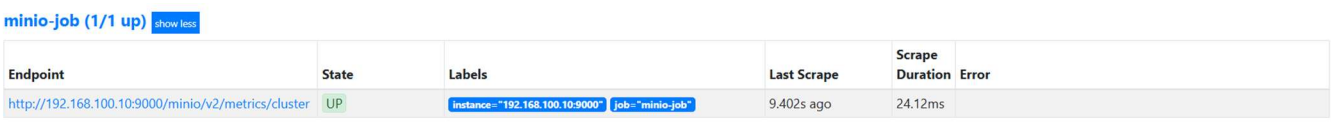


Рисунок 3.17 – У разі введення правильних параметрів "scrape_configs" у конфігураційному файлі "prometheus.yml" здійснюється корректний збір метрик (успішна валідація метрик в Prometheus)

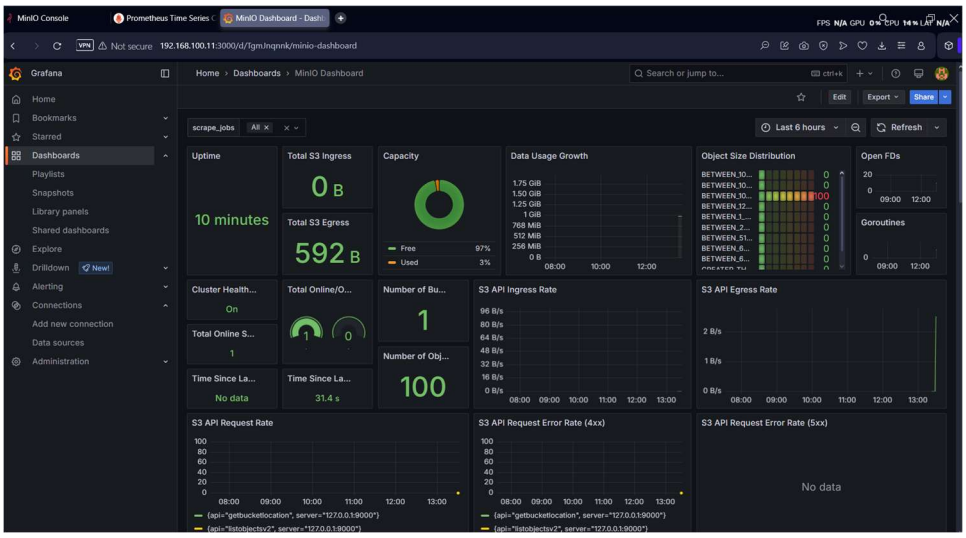


Рисунок 3.18 – У разі введення правильних параметрів "scrape_configs" у конфігураційному файлі "prometheus.yml" здійснюється корректний збір метрик (успішна візуалізація метрик Prometheus через Grafana)

Обмеження ресурсів у VirtualBox. Ще один виклик стосується роботи в умовах обмежених апаратних ресурсів, що характерно для тестового середовища на базі VirtualBox. Віртуалізація другого типу, яка виконується поверх операційної системи Windows, хоч і надає гнучкість у розгортанні інфраструктури, суттєво обмежується характеристиками хост-машини. При одночасному функціонуванні трьох віртуальних машин — сховища, клієнта та моніторингового вузла — виникає навантаження на оперативну пам’ять, дискову підсистему та процесор, що позначається на стабільності й швидкодії (див. рис. 3.19) [22].



Рисунок 3.19 – Схема оптимізації роботи віртуального середовища VirtualBox в умовах обмежених ресурсів

У таких умовах спостерігаються затримки у відповідях з боку об’єктного сховища MinIO, пропуски у циклах збору метрик Prometheus і порушення відображення графіків у Grafana. Паралельна робота хостової ОС і віртуалізованих систем призводить до надмірного використання оперативної пам’яті, активного звернення до swap-файлу та загального уповільнення процесів. Оскільки проєкт базувався на домашній системі, де одночасно виконувалося кілька ресурсомістких задач, ефективність потребувала додаткових заходів оптимізації [8-10, 23, 27].

Одним із рішень стала заміна образів віртуальних машин на більш легкі — зокрема, використання Ubuntu Server замість десктопної версії, що дозволило зменшити споживання оперативної пам’яті завдяки відсутності графічного середовища (див. рис. 3.20) [8, 22, 27].

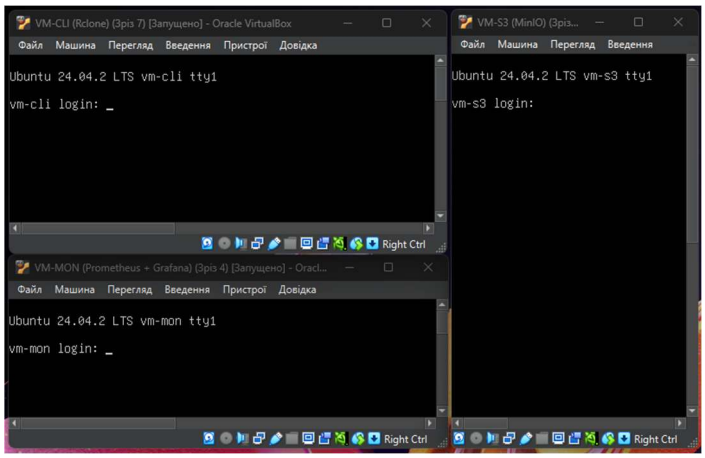


Рисунок 3.20 – Використання Ubuntu Server для зменшення споживання ресурсів

Іншим кроком стала консолідація функцій у рамках однієї віртуальної машини — Grafana і Prometheus, що дозволило відмовитися від ще однієї окремої віртуальної машини. Кешування даних за допомогою rclone на клієнтському боці зменшило кількість запитів до MinIO [8-9, 22-23, 27].

У результаті цих заходів вдалося забезпечити стабільну роботу трьох віртуальних машин при наявності лише 8 ГБ оперативної пам’яті на хості. Продуктивність залишалася достатньою для виконання основних задач, зокрема тестування, моніторингу та відображення ключових метрик у режимі реального часу.

Нестабільність мережевого з'єднання між ВМ. Нестабільність мережевого з'єднання між віртуальними машинами залишається поширеною проблемою, яка проявляється під час використання таких режимів, як "Internal Network" або "Host-Only Adapter" у середовищі VirtualBox. Після перезавантаження машин або оновлення драйверів інтерфейсів часто змінюється IP-адреса, що унеможливлює стабільне з'єднання між окремими вузлами. У результаті Prometheus втрачає можливість зчитування метрик із MinIO через зміну адреси, а rclone отримує помилку з типу "Connection refused" через відсутність відповіді від попереднього інтерфейсу. Водночас Grafana не здатна відобразити графіки, оскільки джерело метрик втрачається, що призводить до повного порушення процесу візуалізації (див. рис. 3.21) [8-10, 22-23].

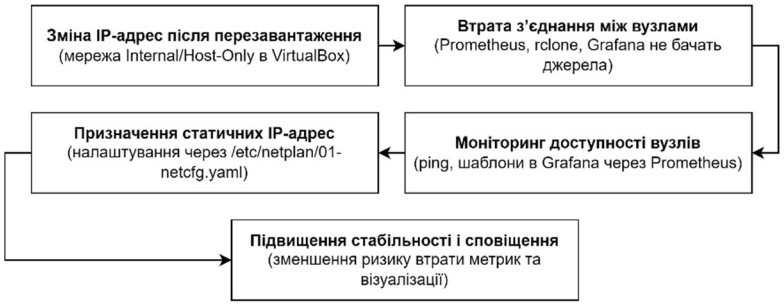


Рисунок 3.21 – Схема забезпечення стабільної взаємодії віртуальних машин шляхом фіксації IP-адрес

Для забезпечення стабільної взаємодії між вузлами доцільно призначити статичні IP-адреси кожній машині за допомогою конфігураційного файлу "/etc/netplan/01-netcfg.yaml". Наприклад, мережевий інтерфейс "enp0s3" може отримати адресу "192.168.100.10/24-' із фіксованими параметрами шлюзу та DNS. Такий підхід дозволяє уникнути автоматичних змін IP-адрес після перезавантаження. Для виявлення недоступних вузлів варто застосовувати періодичний "ping" або ж використовувати відповідні шаблони в Grafana для моніторингу стану вузлів через Prometheus. У результаті зменшується кількість збоїв у роботі служб, підвищується надійність взаємодії між компонентами, а

автоматичне сповіщення про недоступність прискорює виявлення збоїв (див. рис. 3.22) [8-10, 27].

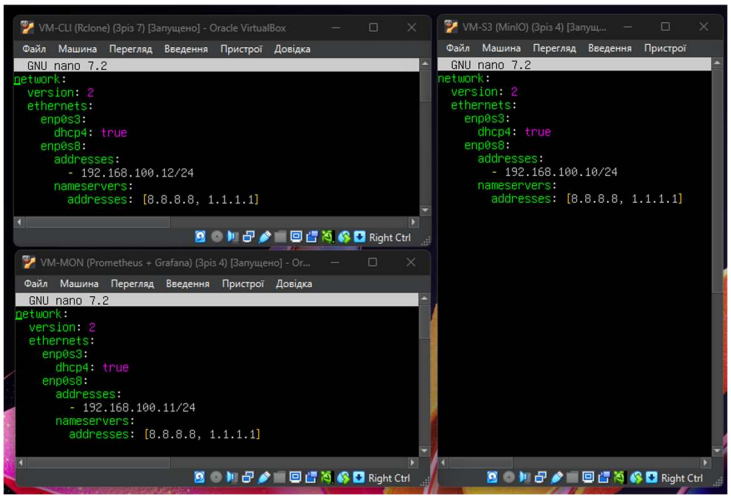


Рисунок 3.22 – Ручне налаштування статичних IP-адресів для взаємної роботи вузлів у одній незмінній підмережі

Конфлікти портів між сервісами. Проблема конфліктів портів між сервісами є ще одним технічним бар'єром під час розгортання моніторингових систем у середовищі віртуалізації. Більшість основних компонентів, зокрема MinIO, Grafana, Prometheus та Node Exporter, працюють на фіксованих портах: 9000, 9001, 9090, 9100 і 3000 відповідно. У ситуаціях, коли машини дублюються або перезапускаються, наприклад під час клонування, можуть виникати конфлікти, що призводить до неможливості запуску сервісу або його недоступності ззовні (див. рис. 3.23) [8-10, 23].



Рисунок 3.23 – Схема уникнення конфліктів портів сервісів у тестовій інфраструктурі моніторингу

Зменшити ймовірність конфлікту можна шляхом прив’язки сервісів до конкретного інтерфейсу за допомогою параметра "--address", наприклад "127.0.0.1:9000", що забезпечує ізоляцію сервісу в межах локального вузла. Додатково можна керувати портами через системні «юніти», у яких явно вказуються допустимі адреси для прослуховування. Доцільною є попередня перевірка зайнятих портів за допомогою "sudo lsof -i -P -n | grep LISTEN", що дозволяє уникнути запуску з помилками. У разі необхідності централізованого доступу та гнучкості у маршрутизації портів можна використовувати Haproxy або nginx у ролі фронтенда, які забезпечують проксування запитів на відповідні порти. У підсумку така стратегія дозволяє уникнути конфліктів під час запуску віртуальних машин і забезпечує стабільний доступ до веб-інтерфейсів сервісів (див. рис. 3.24) [8-10, 27].

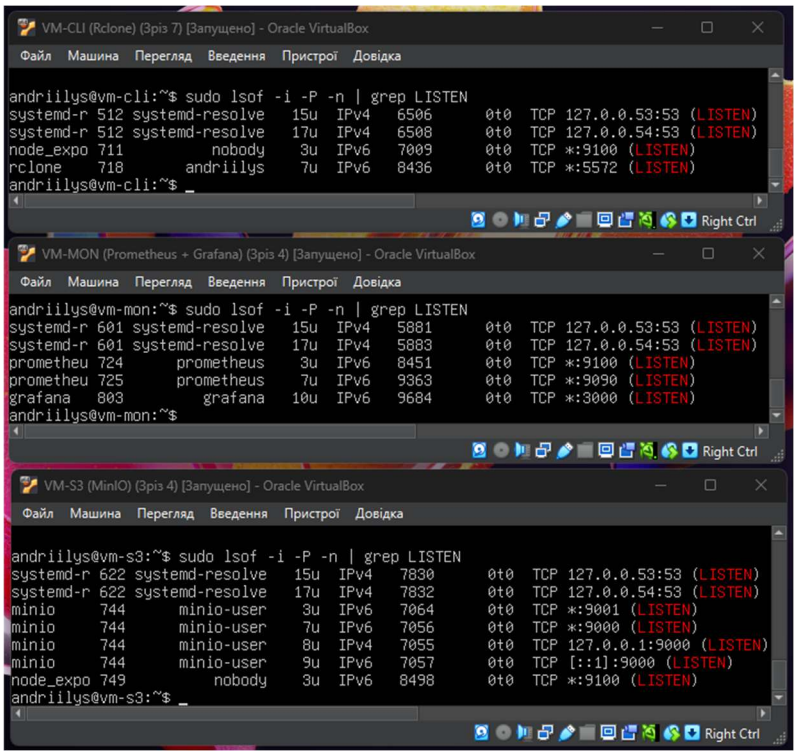


Рисунок 3.24 – Перевірка зайнятих портів для уникання конфліктів

3.3 Перспективи масштабування системи в умовах реального IT-середовища

Реалізована система моніторингу та оптимізації хмарного об’єктного сховища на базі MinIO, Prometheus та Grafana демонструє стабільну ефективність у середовищі з обмеженими ресурсами, що дозволяє оцінити її придатність для базового функціоналу. Водночас інтеграція такого рішення в повноцінне IT-середовище з підвищеними вимогами до надійності, безперервної доступності та автоматизованого резервного копіювання потребує подальшого розвитку, спрямованого на формування архітектури, готової до використання на рівні продакшн-інфраструктури. Перспективним напрямом залишається трансформація архітектури системи у бік гнучких, масштабованих рішень, здатних відповідати умовам сучасних IT-викликів.

Перехід на контейнеризацію (Docker, Kubernetes). Поточна реалізація побудована на трьох окремих віртуальних машинах, кожна з яких виконує власну роль. Такий підхід добре працює у середовищах локального тестування, однак значно обмежує можливості масштабування та адаптації до змінних навантажень, що є критичним у реальному середовищі. З огляду на це доцільним стає перехід до використання контейнеризації, зокрема з опорою на Docker або Kubernetes. Завдяки Docker можливо оперативно запускати, оновлювати та масштабувати окремі сервіси системи, створюючи конфігурації у вигляді єдиного YAML-файлу, що включає всі основні компоненти — MinIO, Prometheus, Grafana, Node Exporter. Такий підхід знижує споживання ресурсів у порівнянні з повноцінними віртуальними машинами та спрощує процеси розгортання й відновлення [8-10, 17].

Використання Kubernetes відкриває додаткові можливості, включаючи автоматичне масштабування як у вертикальному, так і в горизонтальному напрямках, механізми самовідновлення після збоїв та збереження стану за допомогою StatefulSets. Наприклад, розгортання MinIO як stateful застосунку з PVC

(Persistent Volume Claim) дозволяє забезпечити збереження об’єктів навіть після перезапусків контейнерів. Prometheus може бути встановлений із використанням Helm-чарту Prometheus Operator, що додає до екосистеми централізоване управління конфігураціями, моніторингом та алертами. Grafana з persistent storage та інтеграцією Alertmanager створює повноцінне середовище для візуалізації та сповіщення, тоді як Ingress Controller забезпечує зовнішній доступ до сервісів за межами кластеру [8-10, 17].

У поєднанні ці інструменти дозволяють сформувати гнучку, самовідновлювану систему, готову до роботи у продуктивному середовищі з урахуванням сучасних підходів cloud-native архітектури. Такий розвиток рішення забезпечує не лише стабільність, але й адаптивність до зростаючих вимог масштабування та інтеграції.

Інтеграція з публічними S3-сумісними хмарами. Інтеграція з публічними S3-сумісними хмарами може бути реалізована у межах гібридної моделі зберігання, що характерна для корпоративного середовища, де частина даних зберігається локально — зокрема, конфіденційна або критична для безпеки інформація — тоді як інша частина розміщується у хмарі. Використання сервісів на кшталт AWS S3, Wasabi або Backblaze B2 дозволяє побудувати сценарії, де локальне сховище на базі MinIO синхронізується з віддаленим хмарним об’єктним сховищем за допомогою утиліт на зразок "mc mirror" або "rclone", що забезпечує автоматичне дублювання важливих об’єктів та знижує ризики втрати даних. Підтримка режиму "Gateway" у MinIO дозволяє системі працювати як проксі до оригінального S3-ресурсу з можливістю зберігати гарячі дані у локальному кеші, що, у свою чергу, значно скорочує час доступу до часто використовуваних об’єктів [2, 10, 14, 23].

Подібний підхід також відкриває перспективи мультихмарної інтеграції, коли декілька S3-сумісних endpoint-ів об’єднуються в єдину логіку доступу та зберігання, дозволяючи дублювати дані у різних географічних регіонах для підвищення надійності та доступності. Наприклад, конфігурація може здійснюватися через встановлення відповідних псевдонімів та дзеркалювання локального сегменту на зовнішнє сховище командою типу "mc alias set aws

https://s3.amazonaws.com ACCESS_KEY SECRET_KEY" з наступним "mc mirror localbucket aws/backupbucket", що створює автоматизовану модель передачі резервних копій у хмару [1-2].

Застосування для резервного копіювання. Застосування MinIO як повноцінного сховища резервного копіювання передбачає його інтеграцію як S3-бекенду для популярних систем, таких як BorgBackup, Proxmox Backup Server або Veeam через підтримку S3 Target, що робить можливим створення ефективного репозиторію для віртуальних машин і образів. Крім того, Nextcloud може використовувати MinIO як зовнішнє сховище для користувацьких даних, забезпечуючи прозорий обмін із мінімальними затримками та гнучке масштабування. У рамках резервного копіювання це дозволяє впровадити багаторівневу стратегію зберігання — наприклад, розділення за частотою створення копій: щоденні, щотижневі, щомісячні; додавання тегів для класифікації об’єктів за критичністю та створення окремих бакетів під різні типи даних. У результаті формується система, яка підтримує [3-4, 27]:

— версійність файлів;

— зберігання снапшотів;

— оптимізоване управління життєвим циклом.

Реалізація алертів та автоматичного реагування. У поточній тестовій інфраструктурі зібрані метрики наразі візуалізуються через дашборди Grafana, однак у реальних умовах експлуатації цього недостатньо. Для забезпечення стабільної роботи необхідно впровадити активну систему сповіщень, яка дозволяє своєчасно реагувати на аномалії — наприклад, перевищення порогових значень, падіння вузлів або зростання латентності. Алертинг може бути реалізований безпосередньо через Grafana шляхом створення правил для конкретних панелей, а також за рахунок налаштування інтеграцій з каналами сповіщення, такими як Slack, Telegram або електронна пошта. Існує можливість додати затримку перевірки умови алерту протягом визначеного інтервалу (наприклад, 5 хвилин), щоб уникнути хибних спрацювань у разі короткочасних флуктуацій [8-9].

Альтернативним або доповнюючим варіантом є використання Alertmanager у складі екосистеми Prometheus. Дане рішення дозволяє централізовано керувати алертами, групувати повідомлення за типами інцидентів та маршрутизувати їх до потрібних каналів зв’язку [8-10].

Забезпечення високої доступності (HA). У контексті забезпечення високої доступності (HA) критичною є здатність системи зберігати працездатність у разі відмови окремих компонентів. Зокрема, дані повинні залишатися доступними, а система моніторингу — автоматично повідомляти про інцидент. Для реалізації такої стратегії у MinIO доцільно використовувати "distributed mode", розгортаючи його на декількох вузлах (наприклад, 4 або 8) з увімкненим "erasure coding", що дозволяє відновити дані навіть у разі втрати частини дисків [10].

Необхідним компонентом є також сервіс автоматичного виявлення (Service Discovery) або DNS, що забезпечує зв’язність між вузлами. У системі моніторингу варто впровадити відмовостійкий Prometheus у парному запуску з режимом "standby" або реплікацією даних за допомогою Remote Write, з можливістю спільного зберігання серій у зовнішній базі даних — наприклад, PostgreSQL або ClickHouse — через Grafana Mimir або Cortex. Додаткове навантаження рівномірно розподіляється через балансувальник (як-от nginx або HAProxy), який також може дублювати точки входу до API MinIO у разі потреби [8-10, 14].

Для підвищення стійкості системи зберігання рекомендовано також впровадити резервне копіювання конфігурацій MinIO за допомогою "mc admin backup", що дає змогу швидко відновити налаштування після збою, або у процесі переносу системи на нове обладнання [10, 23].

Інтеграція в корпоративну інфраструктуру. Тестова варіація комплексу, що демонструє базову працездатність системи, при наявності обґрунтованої потреби може бути адаптована для використання в масштабованій корпоративній інфраструктурі. Така система здатна виконувати кілька функцій одночасно: слугувати локальним сховищем для мікросервісів, зокрема для зберігання артефактів CI/CD; виступати середовищем для зберігання критичних даних; реалізовувати функції моніторингу як частина централізованої DevOps-платформи;

а також застосовуватись як тренувальна платформа для фахівців, що опановують хмарні технології. Значні інтеграційні можливості системи дозволяють підключення до внутрішніх сервісів аутентифікації через LDAP або Active Directory, організацію API-запитів з корпоративних застосунків, а також імпорт логів та метрик через стек Elasticsearch або Splunk, що розширює функціональність у бік аналітики та аудиту [17-20].

Архітектурна модель, що лежить в основі такої інтеграції, має вигляд наскрізної взаємодії сервісів та користувачів через захищений доступ (див. рис. 3.25) [8-10, 14, 17, 23].

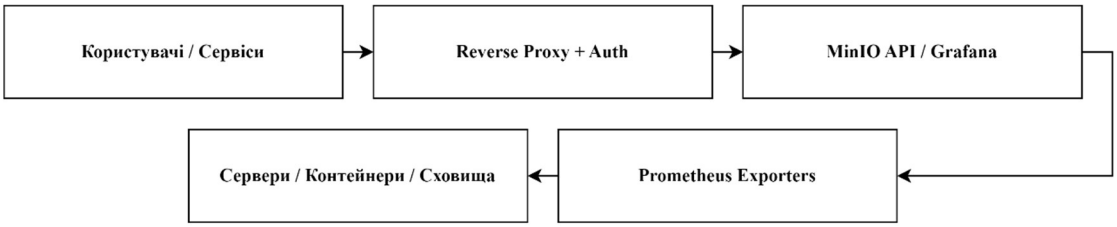


Рисунок 3.25 – Удосконалена потенційна модель для повноцінного масштабування в межах реального IT-середовища

Завдяки цій архітектурі система набуває потенціалу для повноцінного масштабування в межах реального IT-середовища. Сучасний відкритий стек інструментів — MinIO, Prometheus, Grafana — дозволяє еволюцію платформи зі статусу навчального стенду до стабільної, промислової складової корпоративної інфраструктури. Застосування контейнеризації, розширеного алертингу, багаторівневого зберігання, автоматизації резервного копіювання та реалізація високої доступності створює контрольовану, адаптивну систему, яка готова до роботи в умовах зростаючого навантаження, зміни вимог і необхідності забезпечення безперервного доступу до критичних сервісів.

ВИСНОВКИ

У межах роботи було реалізовано програмний комплекс для моніторингу та оптимізації хмарного об’єктного сховища, що базувався на архітектурі «клієнт – сховище – моніторинг – панель керування». Основу рішення склали об’єктне сховище MinIO, система збору метрик Prometheus і платформа візуалізації Grafana. Середовище для розгортання складалося з окремих віртуальних машин у VirtualBox, що дозволило імітувати реальну розподілену інфраструктуру з обмеженими ресурсами, забезпечити тестування сценаріїв навантаження та налаштувати повноцінну взаємодію між вузлами.

У процесі побудови комплексу особлива увага приділялася правильному добору інструментів за критеріями відкритості стандартів, сумісності й мінімізації вимог до ресурсів. Кожен компонент був налаштований окремо, із забезпеченням інтеграції через відкриті API. За допомогою Grafana було створено набір дашбордів для моніторингу основних показників роботи системи, таких як кількість запитів, використання ресурсів і стан доступності сервісів. Prometheus забезпечив автоматичний збір метрик і зберігання історичних даних для подальшого аналізу.

Покращення продуктивності системи досягнуто через впровадження кешування на клієнтському боці та паралельної передачі даних за допомогою Rclone. Найбільш відчутні зміни зафіксовано на вузлі зі сховищем VM-S3 — зниження навантаження на процесор із 38,0 % до 27,7 % і зменшення затримки запису на диск із 7,23 мс до 5,04 мс. На клієнті VM-CLI також відбулося зниження використання оперативної пам’яті та обсягу кешу зі 633 MB/s до 254 MB/s.

Подальший розвиток комплексу передбачає можливість інтеграції контейнеризації (Docker/Kubernetes), впровадження автоматичного резервного копіювання, алертингу та розподіленої архітектури для підвищення надійності та масштабованості рішення у корпоративних середовищах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is cloud storage? - cloud storage explained - AWS. Amazon Web Services, Inc. URL: https://aws.amazon.com/what-is/cloud-storage/#:~:text=There%20are%20three%20main%20cloud,file%20storage,%20and%20block%20storage. (дата звернення: 24.03.2025).
2. Kranz G., Lelii S. What is cloud storage provider? | Definition from TechTarget. Search Storage. URL: https://www.techtarget.com/searchstorage/definition/cloud-storage-provider (дата звернення: 24.03.2025).
3. Optimizing data storage performance in hybrid cloud environments. Storware. URL: https://storware.eu/blog/optimizing-data-storage-performance-in-hybrid-cloud-environments/ (дата звернення: 25.03.2025).
4. 5 common performance problems in cloud computing - liveaction. LiveAction. URL: https://www.liveaction.com/resources/blog-post/5-common-performance-problems-in-cloud-computing/ (дата звернення: 25.03.2025).
5. 10 key cloud performance metrics to monitor - Site24x7. Site24x7. URL: https://www.site24x7.com/learn/key-cloud-performance-monitoring-metrics.html (дата звернення: 26.03.2025).
6. Optimizing cloud performance testing: metrics to track & techniques. NZO Cloud. URL: https://nzocloud.com/blog/cloud-performance-testing/ (дата звернення: 27.03.2025).
7. Metrics of measurement: essential analytics for cloud computing - binadox. Binadox. URL: https://www.binadox.com/blog/metrics-of-measurement-essential-analytics-for-cloud-computing/ (дата звернення: 28.03.2025).
8. Overview. Prometheus. Prometheus - Monitoring system & time series database. URL: https://prometheus.io/docs/introduction/overview/ (дата звернення: 30.03.2025).
9. Grafana Cloud documentation. Grafana Labs. URL: https://grafana.com/docs/grafana-cloud/ (дата звернення: 01.04.2025).
10. MinIO object storage for linux. MinIO. S3 Compatible Storage for AI. URL: https://min.io/docs/minio/linux/index.html (дата звернення: 02.04.2025).
11. What is agent based monitoring? - key components and benefits. Motadata. URL: https://www.motadata.com/it-glossary/agent-based-monitoring/ (дата звернення: 04.04.2025).
12. What is agentless monitoring? - IT glossary - solarwinds. SolarWinds. URL: https://www.solarwinds.com/resources/it-glossary/agentless-monitoring (дата звернення: 04.04.2025).
13. What Is SNMP? Why Do We Need SNMP? - Huawei. Huawei. URL: https://info.support.huawei.com/info-finder/encyclopedia/en/SNMP.html (дата звернення: 07.04.2025).

14. IBM app connect enterprise 11.0.0. IBM - United States. URL: https://www.ibm.com/docs/en/app-connect/11.0.0?topic=api-monitoring-by-using-administration-rest (дата звернення: 08.04.2025).

15. Cloud monitoring API. Google Cloud. URL: https://cloud.google.com/monitoring/api/ref_v3/rest (дата звернення: 23.04.2025).

16. Capacity optimization strategies: maximizing storage efficiency. DataCore Software. URL: https://www.datacore.com/glossary/capacity-optimization/ (дата звернення: 12.04.2025).

17. Virtualization vs containerization. Scale Computing. URL: https://www.scalecomputing.com/resources/whats-the-difference-between-virtualization-vs-containerization (дата звернення: 13.04.2025).

18. IBM. What is auto scaling?. IBM - United States. URL: https://www.ibm.com/think/topics/autoscaling#:~:text=Auto%20scaling%20allows%20a%20cloud,time%20for%20more%20valuable%20tasks. (дата звернення: 15.04.2025).

19. Data lifecycle management - cloud adoption framework. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/scenarios/cloud-scale-analytics/govern-lifecycle (дата звернення: 17.04.2025).

20. Caching content with cloud CDN. CloudNetDevOps. URL: https://pradeepgadde.com/blog/cloud/2024/05/01/gcp-cloud-cdn.html (дата звернення: 19.04.2025).

21. Aram. Noisy neighbor problem in multi-tenant systems explained briefly. Medium. URL: https://zerofilter.medium.com/noisy-neighbor-problem-in-multi-tenant-systems-explained-briefly-3788ae5e9d5b (дата звернення: 21.04.2025).

22. Documentation – Oracle VirtualBox. Oracle VirtualBox. URL: https://www.virtualbox.org/wiki/Documentation (дата звернення: 22.04.2025).

23. Documentation. Rclone. URL: https://rclone.org/docs/ (дата звернення: 24.04.2025).

24. 4 open source alternatives to amazon S3. Open Source Alternatives To Proprietary Software. URL: https://opensourcealternative.to/alternativesto/amazon-s3 (дата звернення: 25.04.2025).

25. Top 10 prometheus alternatives in 2025 [includes open-source]. Uptrace. URL: https://uptrace.dev/comparisons/prometheus-alternatives (дата звернення: 28.04.2025).

26. Top 11 grafana alternatives [comparison 2025]. Uptrace. URL: https://uptrace.dev/comparisons/grafana-alternatives (дата звернення: 02.05.2025).

27. Ubuntu Server documentation. Ubuntu Server. URL: https://documentation.ubuntu.com/server/ (date of access: 05.05.2025).

28. Лисенко А.Г., Торошанко Я.І. Моніторинг продуктивності хмарних сервісів із застосуванням аналітики Big Data. «Сучасний стан та перспективи розвитку IoT» : VI Науково-техн. конф., м.Київ, 15 квітня 2025 р. (подано до публікації).

29. Лисенко А.Г., Торошанко Я.І. Моніторинг та оптимізація хмарних сховищ у програмному комплексі. «Застосування програмного забезпечення в ІКТ» : Всеукр. науково-техн. конф., м.Київ, 24 квітня 2025 р. С.484-487.

Додаток А

Фрагмент JSON-конфігурації дашборда програмного комплексу моніторингу та оптимізації продуктивності хмарного сховища з трьохвузловою архітектурою

```
{
  "__inputs": [],
  "__requires": [],
  "annotations": {
    "list": []
  },
  "editable": true,
  "gnetId": null,
  "graphTooltip": 1,
  "id": null,
  "iteration": 1657124697381,
  "panels": [
    {
      "type": "graph",
      "title": "IOPS - Read/Write",
      "targets": [
        {
          "expr": "rate(node_disk_reads_completed_total{instance=\"$node\",
device=~\"sda|sdb\"}[1m])",
          "legendFormat": "{{device}} read",
          "refId": "A"
        },
        {
          "expr": "rate(node_disk_writes_completed_total{instance=\"$node\",
device=~\"sda|sdb\"}[1m])",
          "legendFormat": "{{device}} write",
          "refId": "B"
        }
      ],
      "gridPos": { "x": 0, "y": 0, "w": 12, "h": 8 }
    },
    {
      "type": "graph",
      "title": "Disk Latency (avg read/write ms)",
      "targets": [
        {
          "expr": "(rate(node_disk_read_time_seconds_total{instance=\"$node\",
device=~\"sda|sdb\"}[1m]) /
rate(node_disk_reads_completed_total{instance=\"$node\",
device=~\"sda|sdb\"}[1m])) * 1000",
          "legendFormat": "{{device}} read",
          "refId": "A"
        },
        {
          "expr": "(rate(node_disk_write_time_seconds_total{instance=\"$node\",
device=~\"sda|sdb\"}[1m]) /
rate(node_disk_writes_completed_total{instance=\"$node\",
device=~\"sda|sdb\"}[1m])) * 1000",
          "legendFormat": "{{device}} write",
          "refId": "B"
        }
      ],
      "gridPos": { "x": 12, "y": 0, "w": 12, "h": 8 }
    },
    {
      "type": "graph",
```

Продовження додатка А

```
      "title": "CPU Usage %",
      "targets": [
        {
          "expr": "100 - (avg by (instance)
(rate(node_cpu_seconds_total{mode=\"idle\",instance=\"$node\"}[1m])) * 100)",
          "legendFormat": "CPU Usage",
          "refId": "A"
        }
      ],
      "gridPos": { "x": 0, "y": 8, "w": 12, "h": 8 }
    },
    {
      "type": "graph",
      "title": "Memory Usage",
      "targets": [
        {
          "expr": "(node_memory_MemTotal_bytes{instance=\"$node\"} -
node_memory_MemAvailable_bytes{instance=\"$node\"}) /
node_memory_MemTotal_bytes{instance=\"$node\"} * 100",
          "legendFormat": "Memory Used %",
          "refId": "A"
        }
      ],
      "gridPos": { "x": 12, "y": 8, "w": 12, "h": 8 }
    },
    {
      "type": "graph",
      "title": "Cache Usage + Page Faults",
      "targets": [
        {
          "expr": "node_memory_Cached_bytes{instance=\"$node\"} / 1024 / 1024",
          "legendFormat": "Cached MB",
          "refId": "A"
        },
        {
          "expr": "rate(node_vmstat_pgmajfault{instance=\"$node\"}[1m])",
          "legendFormat": "Major Page Faults",
          "refId": "B"
        }
      ],
      "gridPos": { "x": 0, "y": 16, "w": 12, "h": 8 }
    },
    {
      "type": "graph",
      "title": "Disk/Net Errors",
      "targets": [
        {
          "expr":
"rate(node_disk_io_time_weighted_seconds_total{instance=\"$node\"}[1m])",
          "legendFormat": "Disk IO Time",
          "refId": "A"
        },
        {
          "expr": "rate(node_network_receive_errs_total{instance=\"$node\"}[1m])",
          "legendFormat": "Net RX Errors",
          "refId": "B"
        },
        {
          "expr":
"rate(node_network_transmit_errs_total{instance=\"$node\"}[1m])",
          "legendFormat": "Net TX Errors",
```

Продовження додатка А


```
        "refId": "C"
      }
    ],
    "gridPos": { "x": 12, "y": 16, "w": 12, "h": 8 }
  }
],
"templating": {
  "list": [
    {
      "name": "node",
      "type": "query",
      "datasource": "Prometheus",
      "refresh": 1,
      "query": "label_values(node_cpu_seconds_total, instance)",
      "includeAll": false,
      "multi": false,
      "sort": 1
    }
  ]
},
"title": "VM Monitoring (S3, MON, CLI)",
"version": 1
```

Додаток Б

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

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



КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавра

НА ТЕМУ: «ПРОГРАМНИЙ КОМПЛЕКС ДЛЯ МОНІТОРИНГУ ТА ОПТИМІЗАЦІЇ ПРОДУКТИВНОСТІ ХМАРНИХ СХОВИЩ»

Виконав студент групи КІД-42
Лисенко Андрій Григорович
Керівник дипломної роботи:
Торошанко Ярослав Іванович,
канд. техн. наук, доц., с.н.с.

Київ – 2025

Мета, об’єкт, предмет дослідження

Мета роботи – оптимізація продуктивності хмарних сховищ за рахунок використання методів моніторингу, візуалізації метрик і автоматизації управління на основі відкритих інструментів

Об’єкт дослідження – процеси функціонування та продуктивності хмарних сховищ у віртуалізованому середовищі

Предмет дослідження – інструменти моніторингу, автоматизації та аналітики, що забезпечують стабільність і ефективність хмарних сховищ

Актуальність роботи

У роботі розглянуто підходи до побудови програмного комплексу для моніторингу та оптимізації хмарних об’єктних сховищ із використанням віртуалізації та відкритого програмного забезпечення. Проаналізовано типи хмарних сховищ, основні метрики продуктивності, а також методи збору й візуалізації даних за допомогою Prometheus, Grafana та допоміжних агентів моніторингу.

Основну увагу зосереджено на реалізації системи у тестовому середовищі на базі VirtualBox із трьома вузлами: клієнтським, сервером об’єктного сховища та вузлом моніторингу. Детально описано процеси встановлення MinIO, налаштування збору метрик Prometheus, побудови кастомних дашбордів Grafana та перевірки зв'язності через Rclone. Проведено експериментальну оцінку продуктивності передачі даних до та після оптимізації параметрів багатопоточності та кешування, що дозволило зафіксувати суттєве покращення ефективності роботи системи.

Проблеми продуктивності у хмарних середовищах зберігання

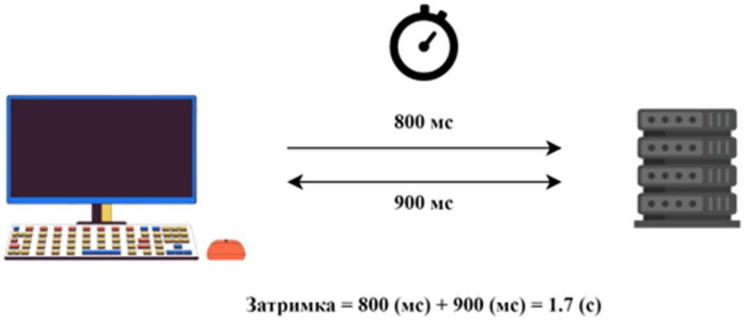


Рис. 1.7 – Проблема високої затримки доступу до даних



Рис. 1.8 – Проблема обмеженої пропускної здатності у обміні даними

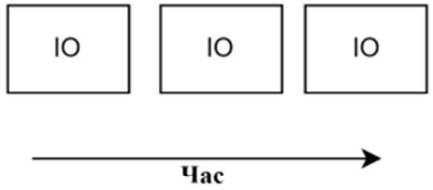


Рис. 1.9 – Проблема навантаженості на дискову систему (IOPS)

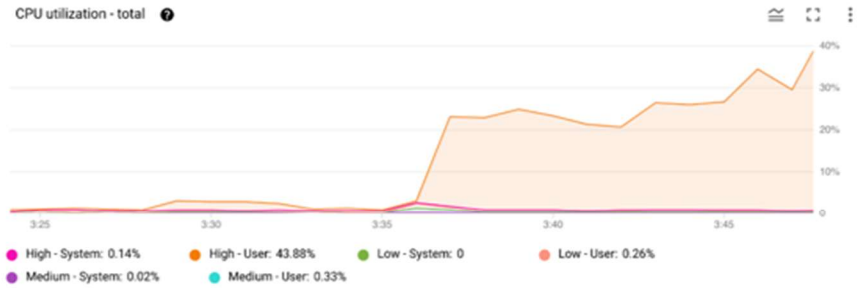


Рис. 1.10 – Проблема високого навантаження на процесор (CPU)

Архітектура програмного комплексу моніторингу та оптимізації

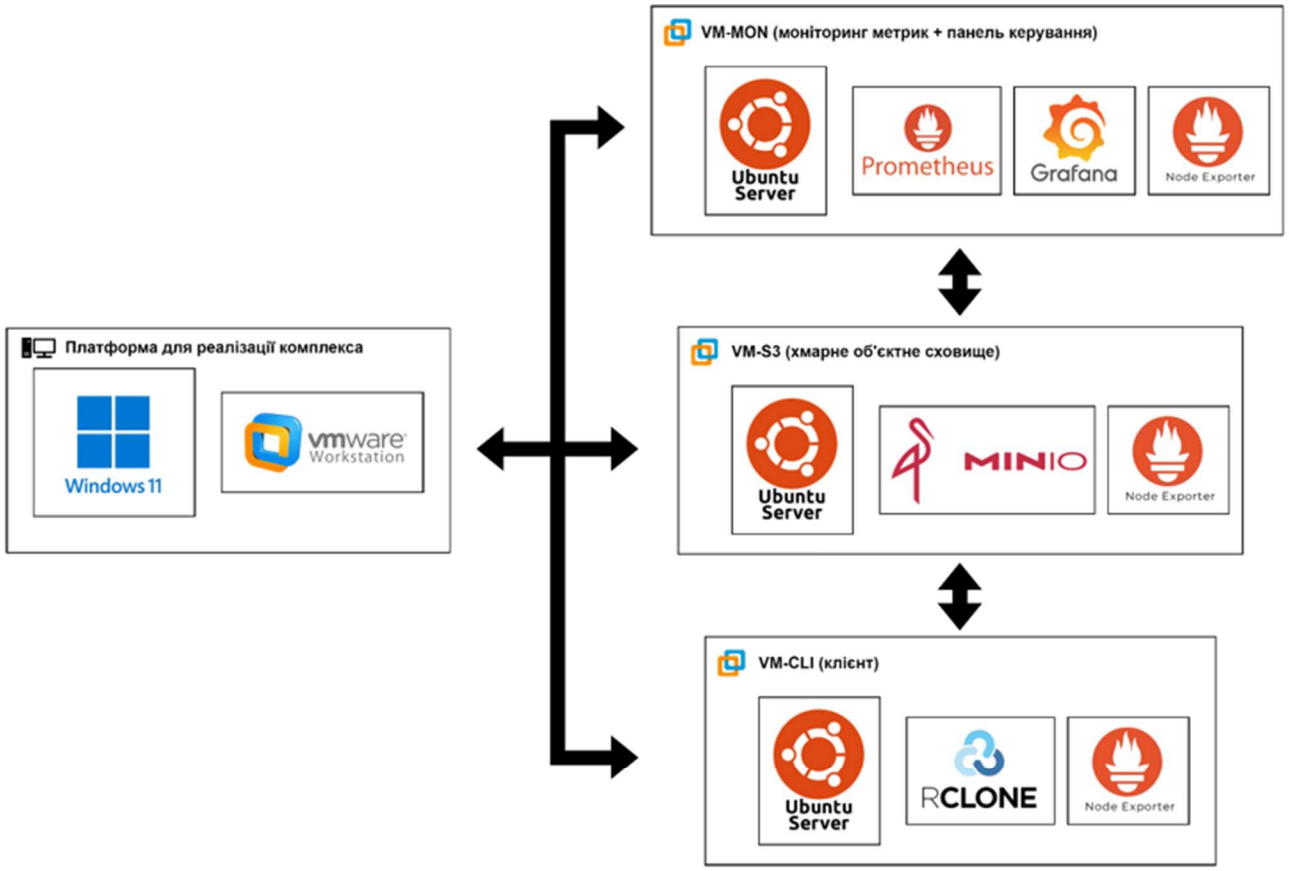


Рис. 2.1 – Архітектура програмного комплексу для моніторингу та оптимізації продуктивності хмарного сховища

5

Побудова віртуалізованого середовища на базі VirtualBox



Рис. 2.9 – Загальні налаштування віртуального вузла VM-S3

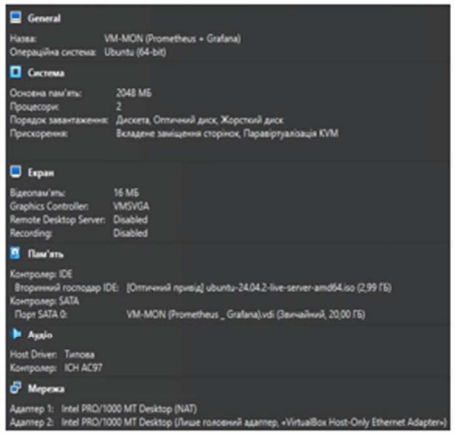


Рис. 2.10 – Загальні налаштування віртуального вузла VM-MON



Рис. 2.11 – Загальні налаштування віртуального вузла VM-CLI

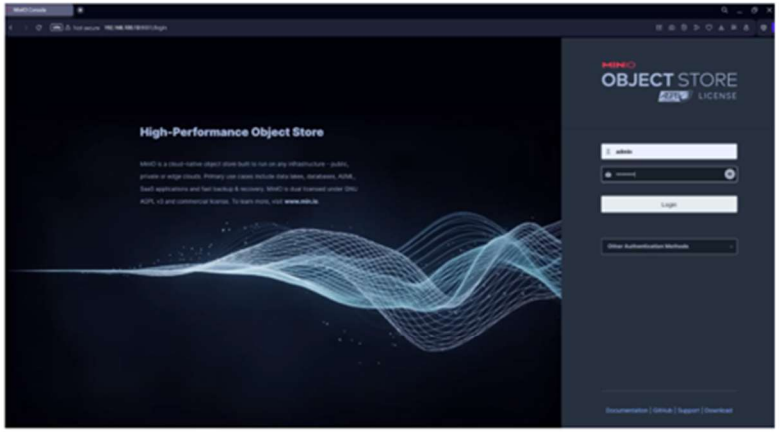


Рис. 2.5 – Процес налаштування віртуального середовища для програмного комплексу по моніторингу та оптимізації продуктивності хмарного сховища



Рис. 2.13 – Кожен вузол налаштований на статичні IP-адреси для надійної взаємної роботи

Реалізація базового функціоналу програмного комплексу



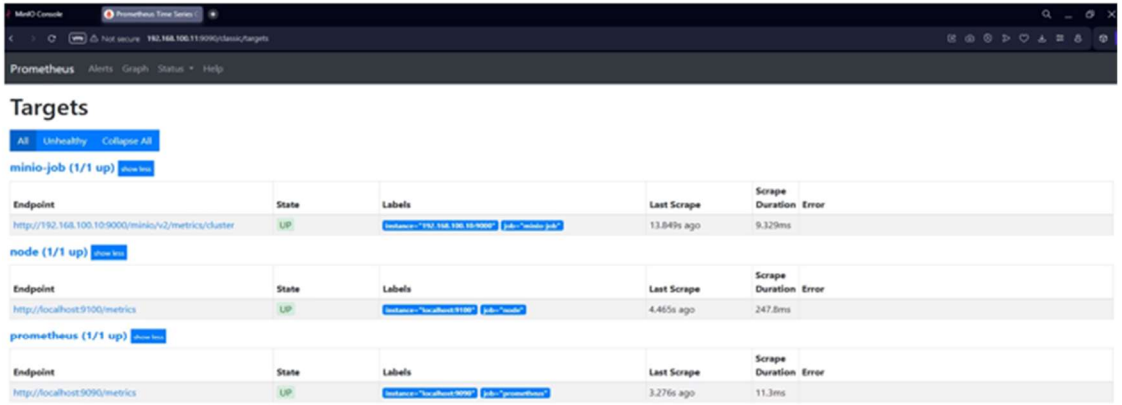


Рис. 2.26 – Робота інструмента для збору метрик Prometheus зі вузла VM-MON

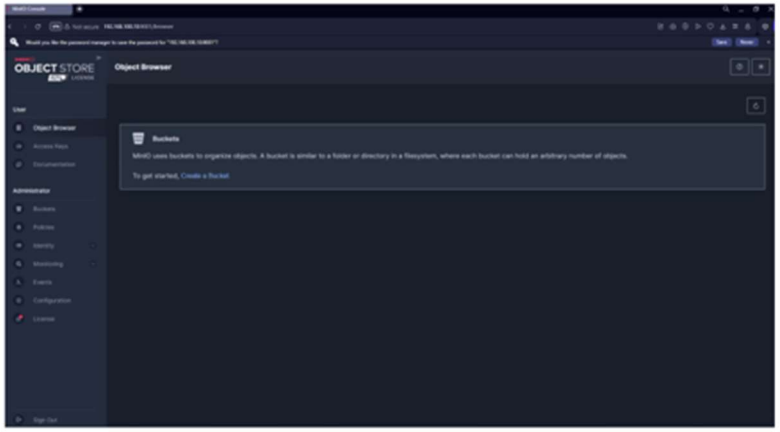


Рис. 2.20 – Робота локального хмарного об'єктного сховища MinIO зі вузла VM-S3

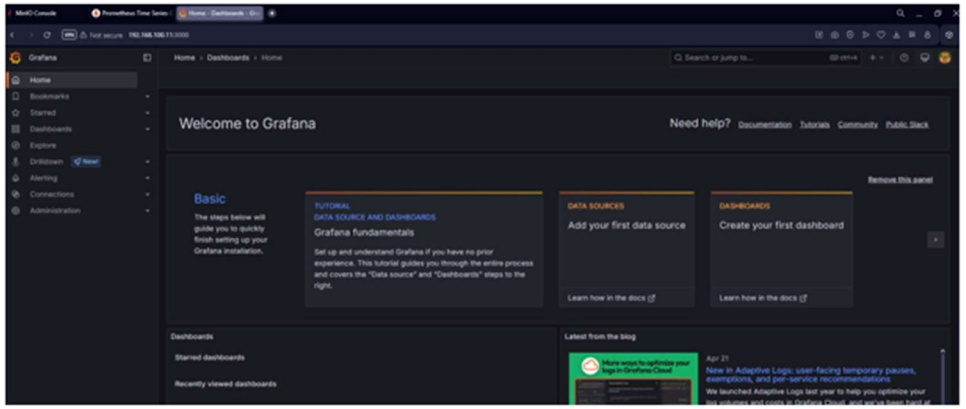


Рис. 2.29 – Робота системи візуалізації метрик Grafana зі вузла VM-MON

Встановлення та налаштування Rclone Remote Control для управління передачею

```
andriilys@vm-cli:~$
andriilys@vm-cli:~$ sudo nano /etc/systemd/system/rclone-rcd.service
```

```
[Unit]
Description=Rclone Remote Control Server
After=network.target

[Service]
ExecStart=/usr/bin/rclone rcd --rc-addr 0.0.0.0:5572 --rc-enable-metrics
Restart=Always
User=andriilys

[Install]
WantedBy=multi-user.target
```

```
andriilys@vm-cli:~$ sudo systemctl daemon-reload
andriilys@vm-cli:~$ sudo systemctl enable rclone-rcd
Created symlink /etc/systemd/system/multi-user.target.wants/rclone-rcd.service → /etc/systemd/system/rclone-rcd.service.
andriilys@vm-cli:~$ sudo systemctl start rc
rc-local.service    rclone-rcd.service
andriilys@vm-cli:~$ sudo systemctl start rclone-rcd
andriilys@vm-cli:~$ sudo systemctl status rclone-rcd
● rclone-rcd.service - Rclone Remote Control Server
     Loaded: loaded (/etc/systemd/system/rclone-rcd.service; enabled; preset: enabled)
     Active: active (running) since Sun 2025-04-27 14:08:30 UTC; 11s ago
   Main PID: 1618 (rclone)
      Tasks: 7 (limit: 2272)
     Memory: 11.0M (peak: 11.2M)
        CPU: 112ms
     CGroup: /system.slice/rclone-rcd.service
             └─1618 /usr/bin/rclone rcd --rc-addr 0.0.0.0:5572 --rc-enable-metrics

Apr 27 14:08:30 vm-cli systemd[1]: Started rclone-rcd.service - Rclone Remote Control Server.
Apr 27 14:08:30 vm-cli rclone[1618]: NOTICE: Serving remote control on http://0.0.0.0:5572/
andriilys@vm-cli:~$ _
```

Рис. 2.39 — Створення та статус служби Rclone Remote Control по порту :5572

```
andriilys@vm-cli:~$
andriilys@vm-cli:~$ rclone copy /home/andriilys/example.txt minio:mybucket
```

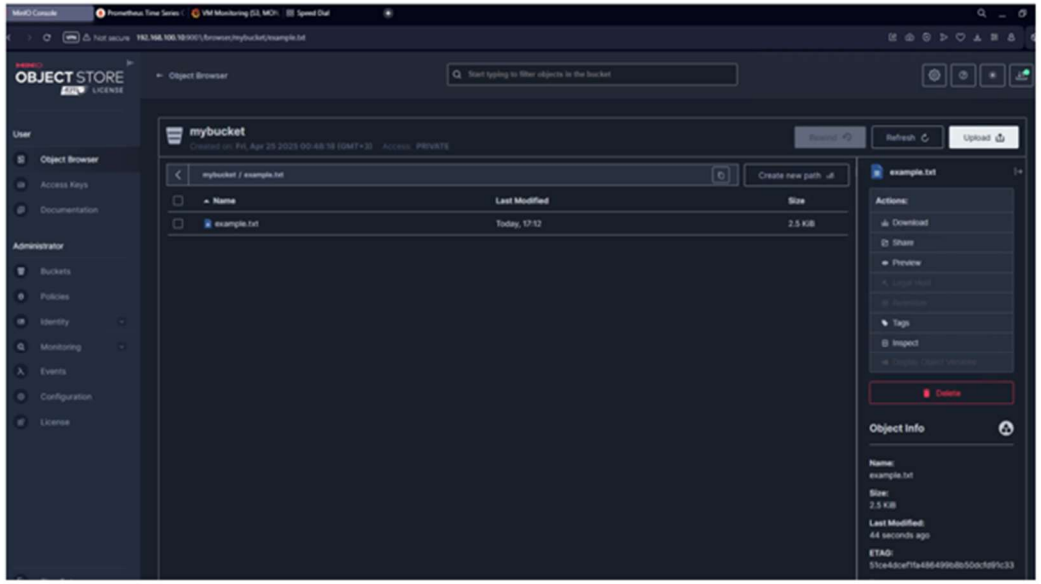


Рис. 2.40 — Успішна передача тестового файлу до бакета MinIO через Rclone зі клієнта до сховища (VM-CLI > VM-S3)

Побудова готових дашбордів для візуалізації метрик на базі Grafana

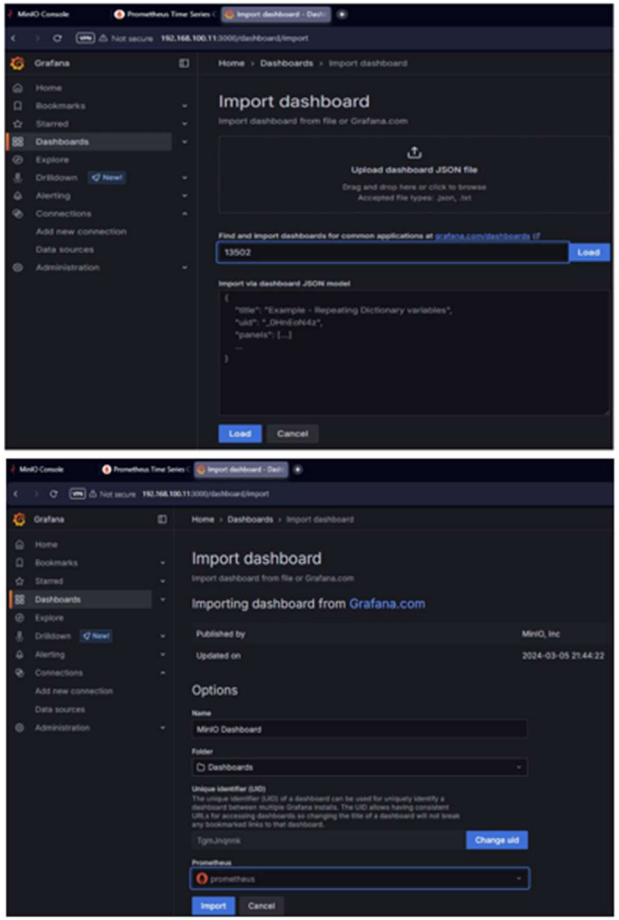


Рис. 2.35 – Імпорт готового дашборда MinIO у Grafana зі вузла VM-MON

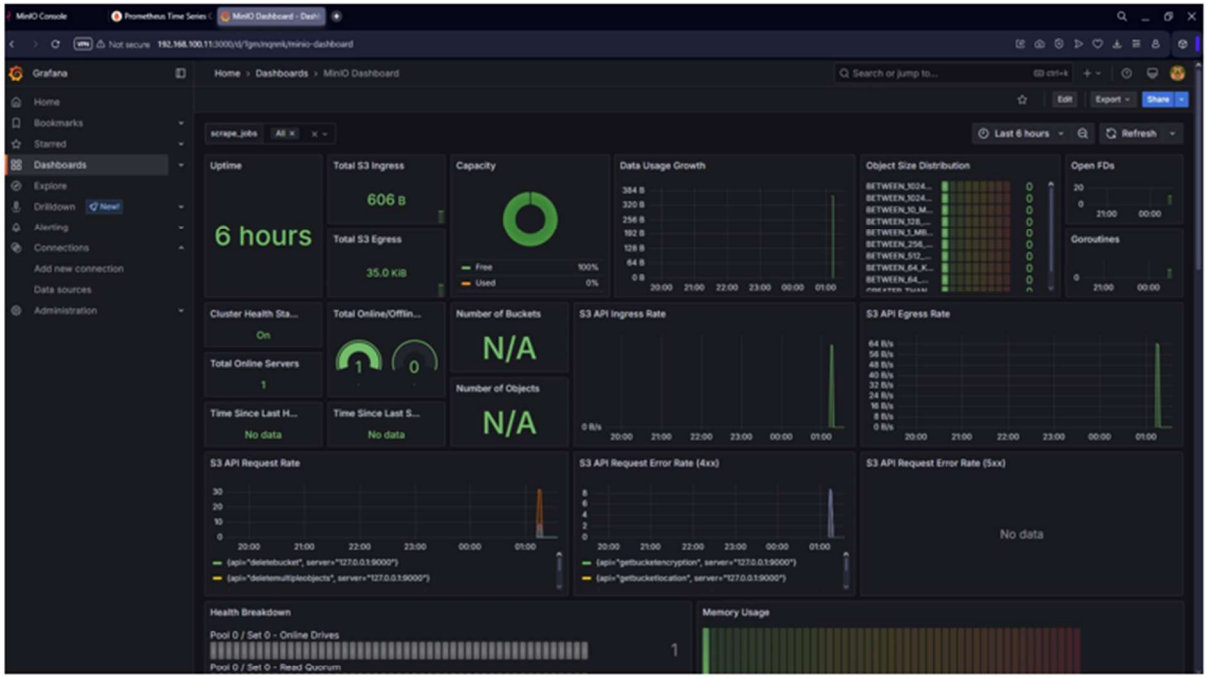


Рис. 2.36 – Перевірка імпортованого готового дашборда MinIO у Grafana зі вузла VM-MON

Встановлення та налаштування агентів Prometheus – Node Exporter (для деталізованого моніторингу всіх вузлів)

```
andriilys@vm-cli:~$ cd /opt
andriilys@vm-cli:/opt$ sudo wget https://github.com/prometheus/node_exporter/releases/download/v1.9.1/node_exporter-1.9.1.linux-
[sudo] password for andriilys:
--2025-04-25 14:30:40--  https://github.com/prometheus/node_exporter/releases/download/v1.9.1/node_exporter-1.9.1.linux-amd64.t
Resolving github.com (github.com)... 140.82.121.4
Connecting to github.com (github.com)|140.82.121.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/9524057/dc8ec09c-2975-42a2-9591-57dd1fff
```

```
andriilys@vm-cli:/opt$ sudo tar -xzf node_exporter-1.9.1.linux-amd64.tar.gz
andriilys@vm-cli:/opt$ sudo mv node_exporter-1.9.1.linux-amd64/node_exporter  /usr/local/bin
```

```
  GNU nano 7.2
[Unit]
Description=Node Exporter
After=network.target

[Service]
User=nobody
ExecStart=/usr/local/bin/node_exporter

[Install]
WantedBy=default.target
```

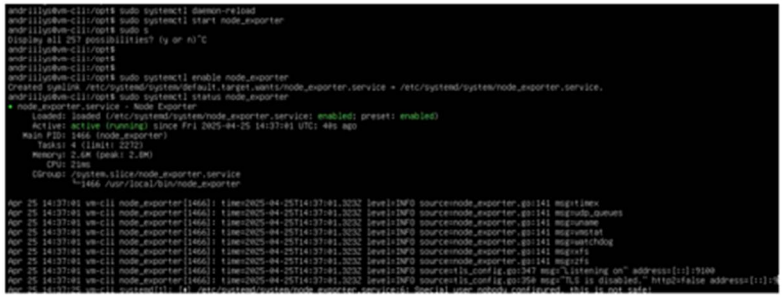


Рис. 2.32 – Розгортання агента Node Exporter для деталізованого збору метрик на всіх вузлах (VM-S3, VM-MON, VM-CLI)

```
- job_name: 'node_exporter'
  static_configs:
    - targets:
        - '192.168.100.10:9100'
      labels:
        node_name: 'VM-S3'
    - targets:
        - '192.168.100.11:9100'
      labels:
        node_name: 'VM-MON'
    - targets:
        - '192.168.100.12:9100'
      labels:
        node_name: 'VM-CLI'
```

Рис. 2.33 – Додавання IP-адреси та назви вузлів від агентів Node Exporter у конфігураційному файлі "prometheus.yml" на вузлі VM-MON

node_exporter (3/3 up) show less

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.100.10:9100/metrics	UP	instance="192.168.100.10:9100" job="node_exporter" node_name="VM-S3"	2.928s ago	27.39ms	
http://192.168.100.11:9100/metrics	UP	instance="192.168.100.11:9100" job="node_exporter" node_name="VM-MON"	11.2s ago	266.2ms	
http://192.168.100.12:9100/metrics	UP	instance="192.168.100.12:9100" job="node_exporter" node_name="VM-CLI"	4.726s ago	25.37ms	

Рис. 2.34 – Отримання нових метрик від агентів Node Exporter у Prometheus на вузлі VM-MON

Побудова кастомного дашборда для візуалізації метрик на базі Grafana

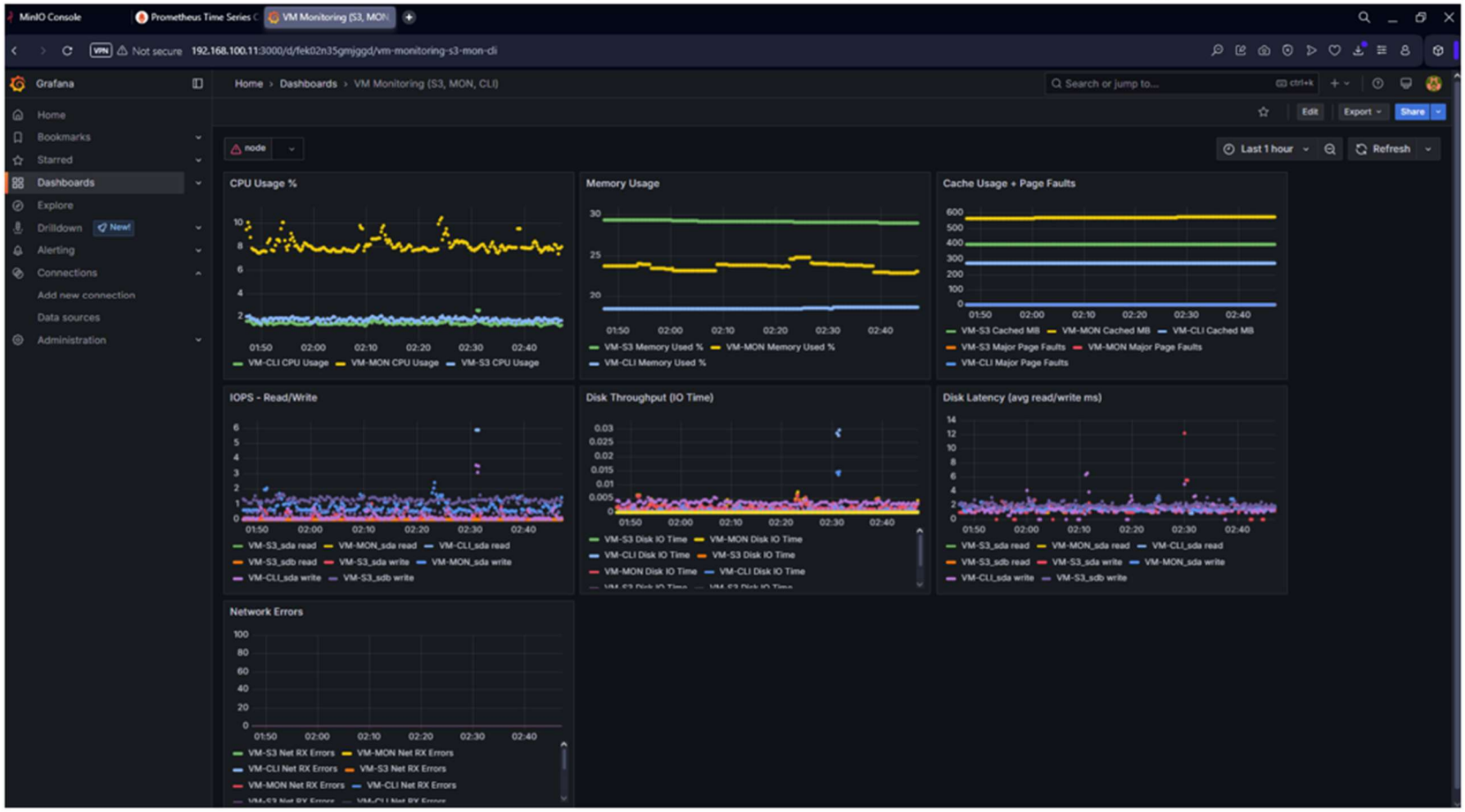


Рис. 2.38 – Приклад власного дашборда для моніторингу всіх вузлів у Grafana

Реалізація методів оптимізації продуктивності

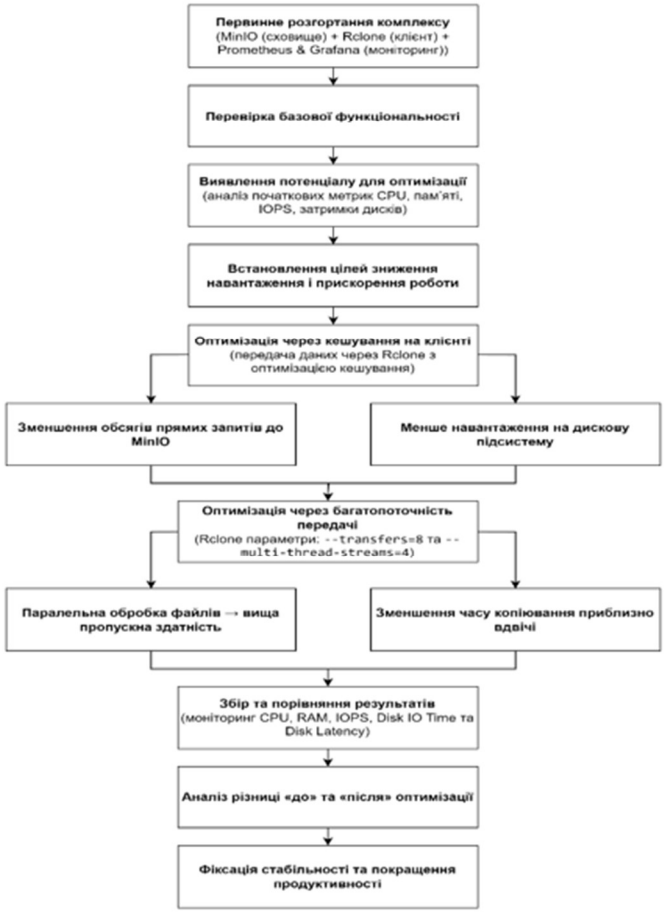


Рис. 3.1 – Метод оптимізації продуктивності хмарних сховищ через кешування та багатопоточність

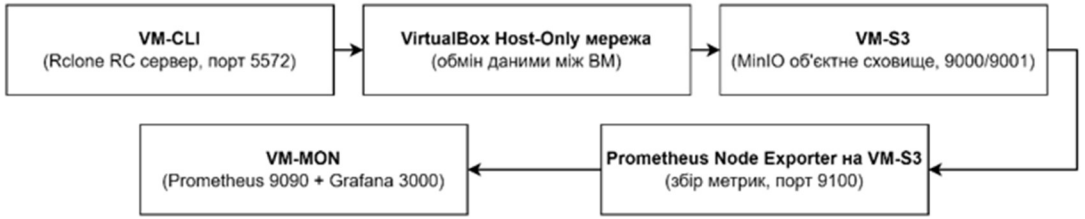


Рис. 3.5 – Загальна схема тестового середовища для аналізу продуктивності передачі даних між вузлами VM-CLI та VM-S3 за допомогою Rclone

Порівняння продуктивності до та після оптимізації



Рис. 3.6 – Порівняння рівня використання процесора на VM-CLI та VM-S3 вузлах до і після застосування параметрів оптимізації

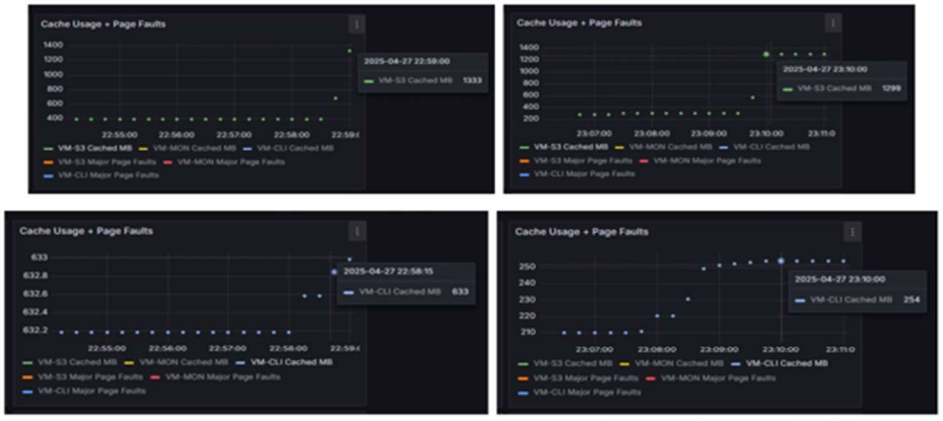


Рис. 3.8 – Показники навантаження на кеш-пам’ять на VM-CLI та VM-S3 вузлах під час передачі файлів до і після оптимізації процесів копіювання

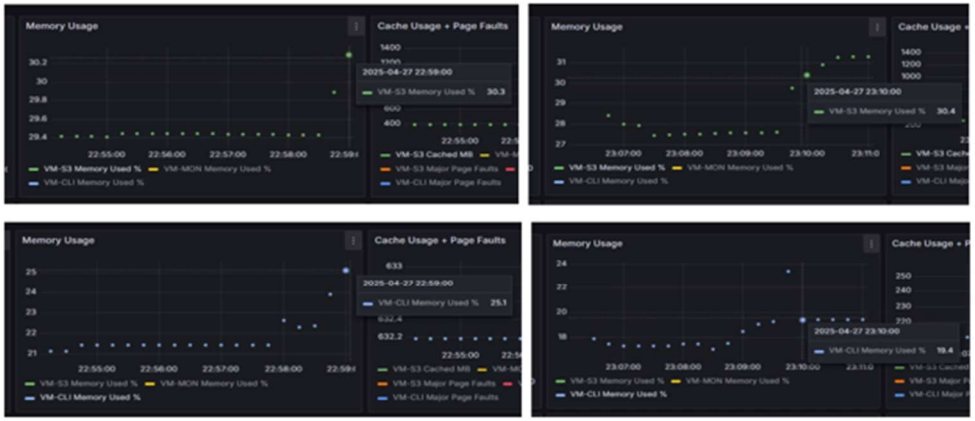


Рис. 3.7 – Візуалізація використання оперативної пам’яті на VM-CLI та VM-S3 вузлах під час передачі даних без оптимізації та після оптимізації потоків



Рис. 3.9 – Кількість операцій вводу-виводу (IOPS) на VM-CLI (читання) та VM-S3 (запис) вузлах у різних режимах роботи

Порівняння продуктивності до та після оптимізації

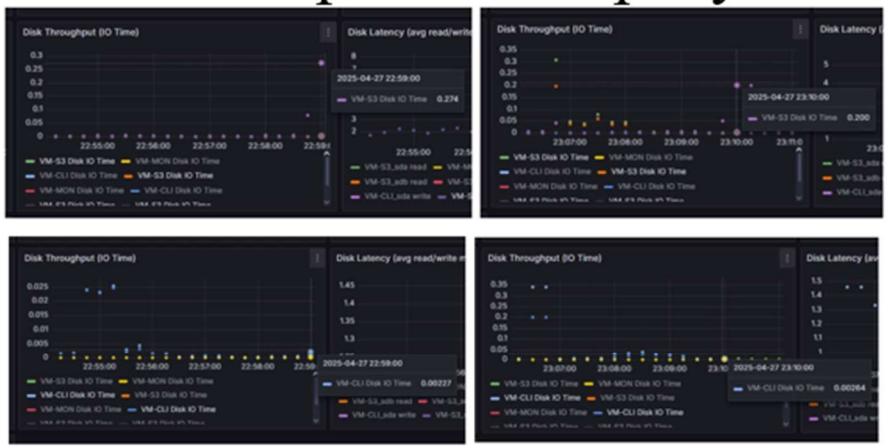


Рис. 3.10 – Порівняння часу виконання дискових операцій (Disk IO Time) на VM-CLI та VM-S3 вузлах до та після оптимізації параметрів копіювання

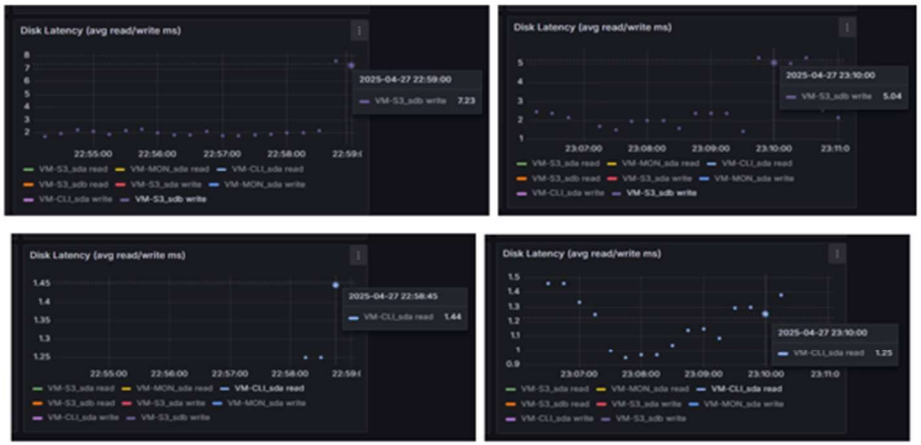


Рис. 3.11 – Середня затримка дискових операцій (Disk Latency) на VM-CLI та VM-S3 вузлах при обробці читання і запису файлів

```
andriilys@vm-cli:~$ rclone copy /home/andriilys/testdata minio:mybucket/ --progress
Transferred:          1000 MiB / 1000 MiB, 100%, 58.792 MiB/s, ETA 0s
Transferred:            100 / 100, 100%
Elapsed time:        17.1s
andriilys@vm-cli:~$
```

Рис. 3.3 – Копіювання даних через rclone без оптимізації

```
andriilys@vm-cli:~$ rclone copy /home/andriilys/testdata minio:mybucket/ --transfers=8 --multi-thread-streams=4 --progress
Transferred:          1000 MiB / 1000 MiB, 100%, 83.104 MiB/s, ETA 0s
Transferred:            100 / 100, 100%
Elapsed time:        12.1s
andriilys@vm-cli:~$
```

Рис. 3.4 – Копіювання даних через rclone зі оптимізацією

Результати тестування і підсумки експериментів

Табл. 3.1 – Порівняльна характеристика продуктивності системи до та після оптимізації параметрів передачі даних

№	Параметр	VM-CLI (до)	VM-CLI (після)	VM-S3 (до)	VM-S3 (після)
1	CPU Usage (%)	18,1%	**18,1%**	38,0%	**27,7%**
2	Memory Usage	25,1%	**19,4%**	30,3%	**30,4%**
3	Cache Usage (MB/s)	633 MB/s	**254 MB/s**	1333 MB/s	**1299 MB/s**
4	IOPS Read/Write	0,200	**0,622**	34,4	**32,4**
5	Disk Throughput (IO Time)	0,00227	**0,00264**	0,274	**0,200**
6	Disk Latency (avg read/write ms)	1,44 ms	**1,25 ms**	7,23 ms	**5,04 ms**
7	Network Errors	0	**0**	0	**0**

Табл. 3.2 – Порівняльна характеристика ключових покращень продуктивності системи

№	Параметр	До оптимізації	Після оптимізації	Покращення
1	CPU Usage (VM-S3)	38,0 %	27,7 %	**-10,3 %**
2	Memory Usage (VM-CLI)	25,1 %	19,4 %	**-5,7 %**
3	Cache Usage (VM-CLI)	633 MB/s	254 MB/s	**~2,5× менше**
4	Disk Latency Read (VM-CLI)	1,44 мс	1,25 мс	**-0,19 мс**
5	Disk Latency Write (VM-S3)	7,23 мс	5,04 мс	**-2,19 мс**

Висновки

У межах роботи було реалізовано програмний комплекс для моніторингу та оптимізації хмарного об’єктного сховища на базі архітектури «клієнт – сховище – моніторинг – панель керування», використовуючи MinIO, Prometheus і Grafana. Система була розгорнута на окремих віртуальних машинах у VirtualBox, що дозволило імітувати роботу в умовах обмежених ресурсів, провести тестування сценаріїв навантаження та налаштувати інтеграцію через відкриті протоколи і API. В межах роботи створено дашборди для моніторингу запитів, використання ресурсів і стану вузлів, що забезпечило оперативний контроль за продуктивністю інфраструктури.

Під час оптимізації було впроваджено кешування та багатопоточну передачу через Rclone, що дозволило знизити навантаження на процесор вузла VM-S3 з 38,0 % до 27,7 %, скоротити затримку запису з 7,23 мс до 5,04 мс та зменшити використання кешу на клієнті VM-CLI із 633 МБ/с до 254 МБ/с. Подальші перспективи розвитку комплексу включають інтеграцію контейнеризації, автоматичне резервне копіювання, алертинг і розширення до розподіленої архітектури з високою доступністю.

Публікації та апробація роботи

1. Лисенко А.Г., Торошанко Я.І. Моніторинг продуктивності хмарних сервісів із застосуванням аналітики Big Data. «Сучасний стан та перспективи розвитку IoT» : VI Науково-техн. конф., м.Київ, 15 квітня 2025 р. (подано до публікації).
2. Лисенко А.Г., Торошанко Я.І. Моніторинг та оптимізація хмарних сховищ у програмному комплексі. «Застосування програмного забезпечення в ІКТ» : Всеукр. науково-техн. конф., м.Київ, 24 квітня 2025 р. С.484-487.