МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

другого (магістерського) рівня вищої освіти

на тему:

«ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ДЛЯ ЗБЕРІГАННЯ ТА ОБМІНУ ФАЙЛАМИ ВЕБ-СЕРВІСІВ»

Виконав: здобувач 6 курсу групи Іт-61
спеціальності 126 «Інформаційні системи
та технології» Іваськевич Ю.А.
(Прізвище та ініціали)

Керівник: Желєзняк А.М.
(Прізвище та ініціали)

ДУБЛЯНИ-2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ПРИРОДОКОРИСТУВАННЯ
ФАКУЛЬТЕТ МЕХАНІКИ, ЕНЕРГЕТИКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Рівень вищої освіти другий (магістерський)
Спеціальність 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
(підпис)
д.т.н., професор, Тригуба А. М.
(вч. звання, прізвище, ініціали)
“____” _____202__ року

ЗАВДАННЯ
на кваліфікаційну роботу здобувачу
Іваськевичу Юрію Андрійовичу
(прізвище, ім’я, по батькові)

1.Тема роботи: *«Дослідження технологій для зберігання та обміну файлами веб-сервісів»*
Керівник роботи к.е.н., доцент Желєзняк А.М.
затверджені наказом по університету № 133 / к - с від 28.04.2023 р
2. Строк подання студентом роботи 15.01.2024 р.
3. Вихідні дані: *вихідні дані та вимоги до веб-сервісів, опис інструментів та технологій для реалізації зберігання та обміну файлами, програмна конфігурація веб-сервісу, науково-технічна і довідкова література.*
4. Зміст кваліфікаційної роботи (перелік питань, які необхідно розробити):
Вступ
1. Аналіз стану питання в теорії та практиці та постановка завдання
2. Обґрунтування, вибір та реалізація інструментарію вирішення задачі
3. Результати вирішення задачі
4. Охорона праці та безпека в надзвичайних ситуаціях
5. Визначення ефективності
Висновки
Список використаних джерел
5. Перелік графічного матеріалу: *графічний матеріал подається у вигляді презентації*

6. Консультанти з розділів:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1, 2, 3, 5			
4			

7. Дата видачі завдання *28.04.2023 р.*

Календарний план

№ з/п	Назва етапів дипломного проекту	Терміни виконання етапів роботи	Примітка
1	*Отримання завдання. Вивчення рекомендованої літератури по темі роботи. Написання першого розділу*	*10.04.2023 – 31.05.2023*	
2	*Проектування та опис технічного завдання, обґрунтування та вибір інструментарію реалізації проекту (написання другого розділу).*	*1.06.2023 – 31.08.2023*	
3	*Програмна реалізація поставленого завдання (написання третього розділу)*	*1.09.2023 – 31.10.2023*	
4	*Написання розділу «Охорона праці та безпека у надзвичайних ситуаціях»*	*1.11.2023 – 20.11.2023*	
5	*Оцінка ефективності поставленого завдання (виконання п'ятого розділу)*	*21.11.2023 – 15.12.2023*	
6	*Завершення оформлення основної частини, написання висновків та підготовка презентаційного матеріалу*	*16.12.2023 – 31.12.2023*	
7	*Завершення роботи в цілому. Підготовка до захисту кваліфікаційної роботи*	*01.01.2024 – 15.01.2024*	

Здобувач _____ Іваськевич Ю.А.

(підпис)

Керівник роботи _____ Желєзняк А.М.

(підпис)

РЕФЕРАТ

УДК 004.774

Дослідження технологій для зберігання та обміну файлами веб-сервісів Іваськевич Ю.А. Кафедра ІТ - Дубляни, Львівський НАУ, 2024.
Кваліфікаційна робота: 55с. текст. част., 28 рис., 3 табл., 48 джерел.

Наведено особливості технологій для зберігання та обміну файлами. Проведено огляд безкоштовних, з відкритим кодом та комерційних рішень у цій сфері. Сформульовано метод виміру швидкодії наведених систем та критерії для аргументованого вибору того чи іншого продукту в залежності від вимог до реалізації та необхідного функціоналу. Об'єкти дослідження - технології та рішення для обміну та зберігання даних та файлів різного розміру та кількості.

Метою кваліфікаційної роботи є розкриття сучасних підходів та методів, які дозволяють ефективно управляти та передавати дані у великому масштабі.

Окреслено метрики та підходи до оцінки та порівняння технологій для зберігання та обміну файлами веб-сервісів. Вибрано засоби реалізації. Проведено заміри швидкодії у різних сценаріях та зібрані відповідні метрики систем, включно з хмарними рішеннями.

Подано особливості реалізації інформаційних систем з функціоналом зберігання та обміну файлами та практичне їх використання.

Розроблено заходи із охорони праці та безпеки у надзвичайних ситуаціях під час використання інформаційної системи.

Визначено показники ефективності запропонованих технологій зі зберігання та обміну файлами.

Ключові слова: веб-сервіс, технології, хмара, швидкодія, файли.

ЗМІСТ

ВСТУП

У сучасному інформаційному світі, де обсяги даних постійно зростають, важливість ефективного зберігання та обміну файлами набуває все більшого значення. Із ростом попиту на споживання контенту збільшуються і вимоги щодо його збереження, передачі та обробки, що веде за собою більші фінансові витрати. Багато компаній стараються оптимізовувати свої продукти для приросту ефективності обробки інформації, а також для кращого сервісу для кінцевих користувачів для збільшення аудиторії. Тут застосовуються різноманітні підходи для досягнення цілі, але зазвичай з ростом кількості користувачів росте й кількість інформації для зберігання та обміну.

Водночас, зростає і щільність та обсяг вільного місця на серверах, але залишається проблема у власне передачі чи копіюванні інформації з одного сервера на інших, оскільки пропускна здатність мереж росте помаліше. А також є певні фізичні ліміти як швидкість світла, що певним чином обмежує ефективність передачі даних з одного континенту на інший, оскільки будувати швидші канали значно дорожче після певного порогу швидкодії. Аналогічно, зберігання даних на серверах зазвичай не є безкоштовним для комерційних проектів, та й за передачу певного об'єму інформації доводиться платити у випадках, як наприклад скачування файлу через мобільну мережу з тарифікацією кількості використаного трафіку.

Магістерську кваліфікаційну роботу присвячено дослідженню технологій для зберігання та обміну файлами веб-сервісів. Метою є розкриття сучасних підходів та методів, які дозволяють ефективно управляти та передавати дані у великому масштабі.

У роботі буде розглянуто різноманітні способи передачі даних, та зокрема файлів різного розміру між додатками та веб-сервісами, їх порівняння з перевагами та недоліками, а також висновки. Спочатку будуть розглянуті попередні дослідження та роботи на цю або дотичні теми для аналізу стану

питання в теорії та практиці. Далі розберемо кожну технологію окремо, опишемо вибір та реалізацію інструментарію вирішення поставленої задачі, включаючи наявні хмарні технології, та власний практичний досвід використання.

В основній частині роботи наведемо результати вирішення задачі, після чого розглянемо дотичний предмет з охорони праці та безпеки у надзвичайних ситуаціях. В кінці визначимо ефективність наведених методів та сформулюємо висновок.

Як підгрунтя для цієї магістерської кваліфікаційної роботи буде також використано набуті практичні знання, отримані під час проходження дипломної практики в компанії з розробки програмного забезпечення, де отримано можливість не лише теоретично вивчити сучасні технології, але й застосувати деякі технології для зберігання та обміну файлами веб-сервісів на реальному проекті, зокрема й хмарні сервіси такі як Microsoft Azure Cloud для побудови гібридних рішень.

Під час проходження практики результати дослідження були апробовані на Міжнародній студентській конференції “Студентська молодь і науковий прогрес в АПК” та представлена доповідь на тему дослідження технологій для зберігання та обміну файлами веб-сервісів.

РОЗДІЛ 1.
АНАЛІЗ СТАНУ ПИТАННЯ В ТЕОРІЇ ТА ПРАКТИЦІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Теоретичні основи веб-сервісів та файлів як виду інформації

Веб-сервіси - це програмні засоби, які дозволяють різним системам та пристроям взаємодіяти через мережу Інтернет. Основна ідея полягає у використанні стандартних протоколів для обміну даними між різними системами. Наприклад, серед таких протоколів - HTTP, SOAP. SOAP (Simple Object Access Protocol) є одним з видів веб-сервісів, заснованим на XML і HTTP, призначений для виклику методів об'єктів або функцій на віддаленому сервері. REST (Representational State Transfer) - це архітектурний стиль, який використовує HTTP для звернення до ресурсів через методи HTTP, такі як GET, POST, PUT, DELETE і т. д. JSON-RPC та XML-RPC - це інші протоколи віддаленого виклику процедур, які використовують формати JSON або XML для виклику функцій на віддаленому сервері.

Застосування веб-сервісів широкі і різноманітні. Наприклад, вони використовуються для інтеграції різних систем, де вони забезпечують обмін даними та функціями між ними. Веб-сервіси надають можливість розробляти мультиплатформені додатки, які працюють на різних пристроях та платформах. Також їх використовують у сервісно-орієнтованих архітектурах (SOA) чи мікросервісних підходах, де різні послуги можуть бути незалежно розгорнуті та використовуватися для створення нових додатків. Наприклад, в соціальних мережах веб-сервіси дозволяють обмінюватися даними між користувачами та серверами. Вони створюють можливості для ефективного обміну інформацією через Інтернет, що робить їх потужним інструментом для взаємодії різних систем та пристроїв. Багато інформації, якою обмінюються сервіси між собою та кінцевими користувачами є файлами, наприклад відео чи аудіо контент, запити та відповіді у JSON, XML чи бінарному форматі, тощо.

Розглянемо різні наявні технології для зберігання та обміну файлами веб-сервісів, та проаналізуємо їх вивченість в теорії та практиці за матеріалами публікацій.

Загалом є декілька розповсюджених та відомих технологій для обміну файлами веб-сервісів, наприклад FTP (File Transfer Protocol) чи його захищена версія SFTP. Це стандартний протокол для передачі файлів через мережу, що використовується для завантаження та вивантаження файлів між локальним комп'ютером та сервером.

Браузери частіше використовують HTTP та його безпечну версію HTTPS для передачі файлів через мережу, зокрема між сервером та користувачем. Також існує технологія gRPC для передачі бінарних файлів як між різними сервісами так і між сервісом та кінцевим користувачем. Обидва протоколи можуть використовуватись для передачі невеликих файлів зразу, або розбиваючи їх на менші частини. Для останнього у HTTP використовується кодування multipart form data, а gRPC може застосовувати можливість транслювання (streaming) контенту. При цьому файли можуть зберігатись напряму у файловій системі сервера з веб-додатком, або окремо для розподілення.

Технологія веб сокетів (WebSocket) надає можливість встановлювати постійне з'єднання між клієнтом і сервером для швидкої передачі даних у режимі реального часу, стрімінгу даних і передачі великих об'ємів інформації (рис.1.1).

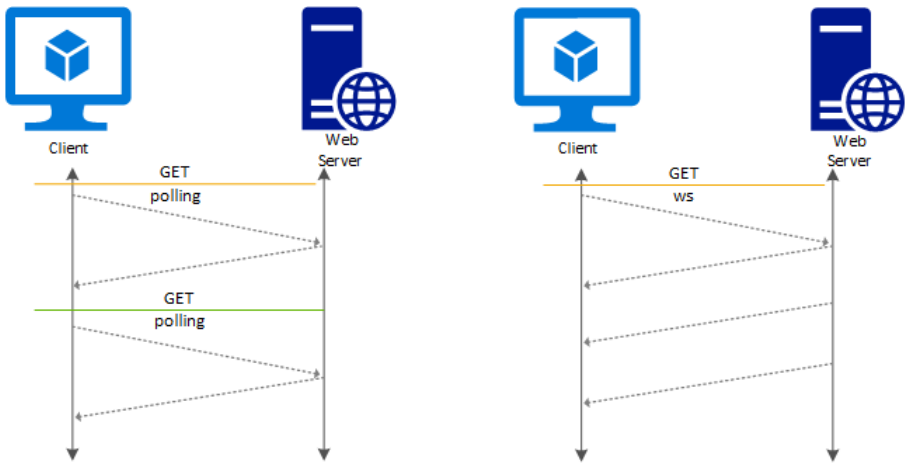

Рисунок 1.1. – Використання технології WebSocket [47]

Подібно до gRPC, в даному випадку використовується транслювання (streaming) бінарних даних, а одне постійне з'єднання дозволяє оптимізацію завантаження великої кількості файлів невеликого об'єму, якщо наприклад вони запитувались клієнтом.

Протокол WebDAV (Web Distributed Authoring and Versioning) надає можливість для спільного доступу до файлів та їхньої зміни через протокол HTTP та віддаленої роботи з файлами на сервері (рис.1.2).

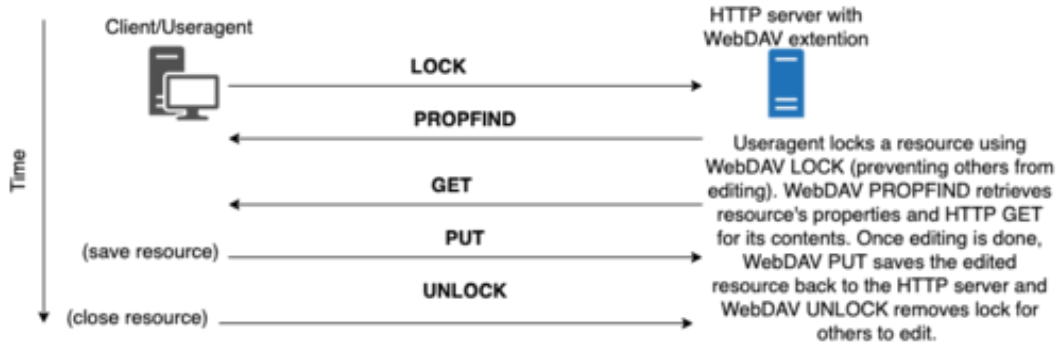

Рисунок 1.2. – Опис застосування протоколу WebDAV [48]

Веб-додатки часто запитують файли із сервера за допомогою RESTful API та отримують Blob (шматок) пам'яті. Аналогічно відбувається і відправка файлів в іншу сторону.

1.2 Централізовані системи збереження та обміну файлами

Централізовані системи збереження та обміну файлами використовуються для ефективного управління даними, забезпечують централізований доступ до файлів, полегшують спільну роботу та забезпечують надійне збереження і обмін інформацією серед користувачів. Це сприяє підвищенню продуктивності та забезпечує безпеку даних.

За допомогою цих доволі простих та розповсюджених технологій працюють і деякі комерційні продукти для зберігання та обміну файлами, як Dropbox, Microsoft OneDrive, Google Drive та інші (рис. 1.3). Вони призначені в

першу чергу для взаємодії із кінцевими користувачами напряму через різні застосунки, а також для легкої колаборації між кількома користувачами, які взаємодіють із тим самим документом чи текою файлів.

Рисунок 1.3. – Комерційні продукти для зберігання та обміну файлами

Ці продукти зазвичай мають програмний інтерфейс (API) для автономного доступу. Таким чином можна створювати сторонні веб-сервіси для взаємодії з ними, як наприклад боти Telegram, чи власні застосунки з використанням офіційних бібліотек. Під час дослідження, не було знайдено практичного порівняння цих технологій в наукових публікаціях, що дало б можливість обгрунтованого вибору, як максимум лише їх дотичне використання у деяких роботах.

Комерційні технології для власне програмного доступу (через програмний інтерфейс API) для зберігання та обміну файлами веб-сервісів пропонуються від багатьох хмарних провайдерів, як рішення для інтеграції у власний продукт. Доволі популярними є Microsoft Azure Blob Storage та Amazon Web Services S3, вони мають багато можливостей та найкраще застосовуються у тандемі з іншими компонентами цих хмарних середовищ. Тут вдалось знайти доволі цікаву публікацію 2018-го року, де Zouheir Daher та Hassan Hajjdiab [7] розглядають та порівнюють ціни на ці два рішення - Azure Blob Storage та AWS S3, а також є певна інформація про різні опції зберігання даних. Але, якщо скористатись Azure Pricing Calculator, то стає очевидно, що на жаль наведені дані про ціни вже не актуальні, а також наведена робота не містить детального порівняння для вибору

тої чи іншої технології в конкретних випадках.

Окрім використання файлової системи напряму для зберігання файлів, веб-сервіси також можуть використовувати спеціально призначені для цього бази даних, як наприклад документні бази зокрема MongoDB, Azure CosmosDB, Amazon DynamoDB. Тут можна зберігати невеликі JSON чи BSON (бінарний JSON) документи, а також виконувати індексований пошук по них чи редагувати дані. Є доволі стара стаття про MongoDB авторства Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon та Lavanya Ramakrishnan [13], але з того часу цей продукт доволі змінився та розвинувся, тож потрібно проаналізувати його актуальну версію.

Часто буває, що невеликих файлів може бути величезна кількість, і їх потрібно передавати між веб-сервісами, як наприклад у випадку централізованого логування, чи якщо повідомлення-сповіщення з одного сервісу до іншого вважати файликами. Для цього часто використовують брокери повідомлень з різноманітними патернами проектування для коректного налаштування відповідних сервісів. Розглянемо популярний брокер Apache Kafka а також хмарні сервіси такі як Azure Service Bus, Azure Event Hub, Amazon SNS, Amazon SQS. Такі рішення доволі добре вивчені, оскільки гарантії консистентності та дизайн самої системи брокера доволі сильно прив'язаний на дослідження, тому наприклад Apache Kafka дуже добре описаний зокрема у статті [11]. Але не вистачає порівняння з комерційними продуктами, та аналізу що саме та в якій ситуації використовувати, адже знайдені дослідження більше спрямовані на досягнення певних характеристик та огляд патернів, які часто використовують у розподілених системах.

Для пришвидшення завантаження різних файлів для кінцевого користувача з віддаленого веб-сервіса використовують Content Delivery Network (CDN), що фактично являє собою проміжний кеш, що знаходиться ближче до користувача ніж оригінальний контент (рис.1.4).

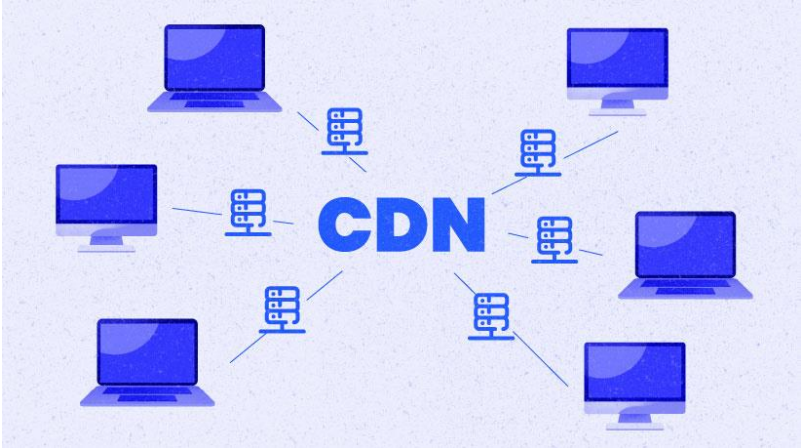

Рисунок 1.4. – Схема мережі доправлення контенту CDN

Популярні хмарні рішення від Azure та Amazon оптимізовані для роботи зі своїми сервісами. Спираючись на доволі хорошу, хоч і відносно застарілу статтю від Sipat Triukose, Zhihua Wen та Michael Rabinovich [12] про цю технологію, її особливості та деякі порівняння, тож у цій роботі буде доповнено ці дані новою інформацією.

1.3 Децентралізовані системи збереження та обміну файлами

Наразі згадані технології та рішення є централізованими, тобто їх контролює наприклад вендор як Microsoft, Google чи Amazon, або конкретний сервер провайдер, де власне і зберігаються оригінальні файли. Додатково розглянемо децентралізовані технології для зберігання та обміну файлами як веб-сервісів так і кінцевих користувачів. Відомим прикладом є Peer-to-Peer (P2P) мережі та протокол BitTorrent (також відомий як просто torrent) (рис.1.5). Широкої популярності він набрав завдяки піратському контенту, про що ще буде згадано у цій роботі в наступних розділах, але при цьому привніс новий спосіб як для зберігання так і покращення швидкодії завантаження файлів.

Технологія BitTorrent базується на багатьох наукових дослідженнях, як і багато публікацій було зроблено саме на цю тему, зокрема за авторства Ashwin R. Bharambe, Cormac Herley, Venkata N. Padmanabhan [16] та Pouwelse, J., Garbacki, P., Epema, D., Sips, H. [17], тож опиратимемось на ці публікації.

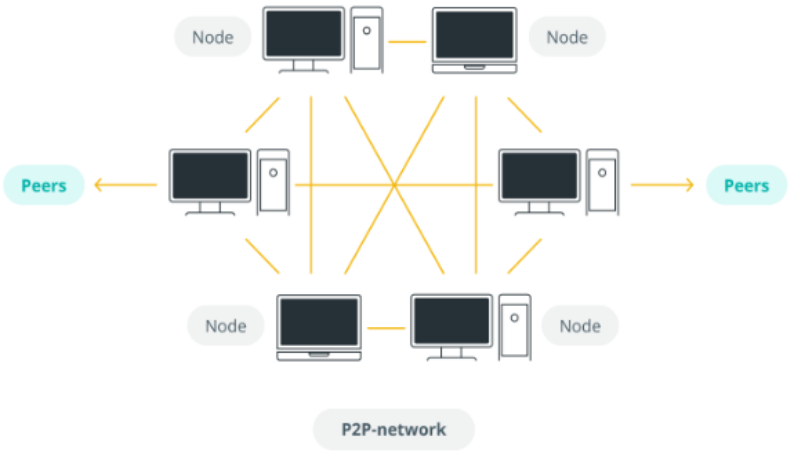

Рисунок 1.5. – Схема Peer-to-Peer (P2P) мережі

Також набирають популярності рішення на основі блокчейну (Blockchain) та з їх допомогою створюються нові системи та технології. Однією з таких є децентралізована файлова система, що використовує блокчейн для верифікації, InterPlanetary File System (IPFS). Вона подібна за механізмами роботи до BitTorrent, але з додатковими можливостями та перевагами верифікації цілісності контенту інструментами блокчейну. Фактично, IPFS використовує P2P мережу для передачі файлів подібно до BitTorrent, але потрібно знати ідентифікаційний номер файлу наперед, оскільки пошуку немає. Також, копії одного і того самого файлу або його частин можуть зберігатись на декількох нодах для досягнення розподіленого зберігання даних за допомогою реплікації.

Технології блокчейну дозволяють забезпечити цілісність файлів, що завантажуються, а також зробити певні оптимізації для зменшення обсягу використаної пам'яті. Ця технологія насамперед використовується за допомогою програмного інтерфейсу API. Зокрема, розглянемо наукові статті за авторства Huang, Huawei & Lin, Jianru & Zheng, Baichuan & Zheng, Zibin & Bian, Jing [8] та Y. Chen, H. Li, K. Li and J. Zhang [9], та у цій роботі доповнимо порівняннями з іншими рішеннями для використання у різних випадках, насамперед комерційних. Візуально різницю архітектур у централізованих та децентралізованих системах зображено на рисунку 1.6.

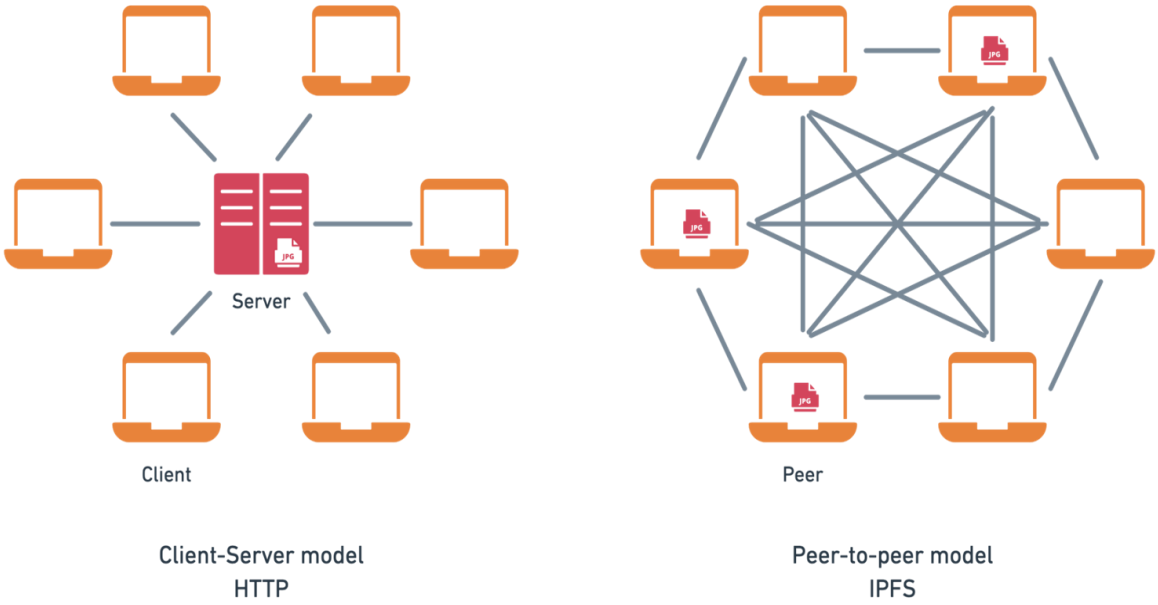

Рисунок 1.6 Порівняльне представлення архітектур централізованих та децентралізованих системах

Тож з наведених матеріалів публікацій можна поставити наступне завдання для цієї магістерської роботи. А саме, порівняти наведені технології та рішення для зберігання та обміну файлів, навести оновлені ціни на платні продукти, та визначити за яких умов використання краще підійде та чи інша технологія, зокрема й у використанні при побудові у комерційних проектах.

РОЗДІЛ 2.
ОБҐРУНТУВАННЯ, ВИБІР ТА РЕАЛІЗАЦІЯ ІНСТРУМЕНТАРІЮ ВИРІШЕННЯ ЗАДАЧІ

2.1 Обґрунтування інструментарію вирішення задачі

Для порівняння необхідно зібрати певні метрики, як от ціна, лімітації, швидкість та потенційні особливості. Потрібно обрати приклади файлів, різного розміру та кількості, наприклад один великий файл - це дозволить побачити швидкодію мережі та ефективність алгоритмів певної технології чи рішення, а також певну кількість файлів невеликого розміру - тут буде видно як добре справляється те чи інше рішення з короткочасним навантаженням, затримкою мережі при географічно віддалених регіонах. Також, за можливості, оберемо принаймні два різних географічно розташованих сервера для тестування. Деталі будуть у наступних розділах.

Для коректних результатів, будемо порівнювати подібні технології, наприклад метрики зі зберігання чи обміну багатьох невеликих файлів одного рішення з тою самою метрикою іншого сервісу. Також, необхідно дослідити як хмарні рішення, для чого будуть використано безкоштовні пробні акаунти, так і певні продукти, які можна встановити у власному кластері - тут використаємо локальний комп'ютер як середовище.

Щоб виміряти якісь метрики, використаємо офіційні програмні інтерфейси (API) та бібліотеки для кожного продукту відповідно - це необхідно для використання всіх переваг тої чи іншої технології по максимуму, адже конкретна бібліотека розроблялась під конкретний продукт з урахуванням його особливостей. Для збору інформації про актуальні ціни на платні продукти та особливості чи переваги конкретного рішення буде використано офіційна документація.

Кожна технологія може потребувати окремої мови програмування для написання скрипта з вимірювання метрик швидкодії, тому важливо

використовувати сучасні версії цих мов, пакунків та додатків до них, а також актуальних на момент написання роботи фреймворків.

Зокрема, буде використано Python 3 із стандартними бібліотеками у середовищі Jupyter Notebook для тестування Google Drive - саме ця мова часто використовується у прикладних задачах, як от тренування моделей штучного інтелекту, які й використовують віртуальні машини Google Collab. Це середовище також добре підходить для виконання невеликих інтерактивних скриптів, чого достатньо для поставленої задачі.

Рисунок 2.1. – Вибір скриптів для вимірювання метрик швидкодії

Для багатьох технологій та рішень буде використано мову програмування C# - це доволі популярна об'єктно-орієнтована мова для написання серверних застосунків, з інтеграціями та хорошою документацією до продуктів Microsoft, а також безліччю офіційних бібліотек під різноманітні інтерфейси API. Також буде задіяно фреймворк .NET Core 6, що є актуальною версією на момент написання роботи. Наведені далі програмні коди виконуються у консольному середовищі, де на екран виводиться результат заміру наприклад швидкодії.

Для деяких продуктів буде використано консольні shell скрипти, як буде видно в наступному підрозділі, оскільки вони використовують інтерфейс командного рядка (CLI).

При використанні комерційних рішень буде використано безкоштовний пробний період для апробації продукту, що дасть достатньо часу та ресурсів для виміру результатів.

Для розгортання певних продуктів на локальному комп'ютері буде використано технологію контейнеризації Docker з можливим використанням docker-compose для складніших сценаріїв (рис.2.2). Це дозволить описати

скрипти розгортання, що можуть бути повторені на іншому комп'ютері для створення того самого середовища, а потім використати програмні коди наведені в роботі для повторних вимірів. Зокрема, це може бути корисним при оновленні продуктів, для перевірки роботи, а також для порівняння якогось певного комп'ютера чи сервера, адже результати сильно залежать від середовища, де вони виконуються, можливостей мережі та інших факторів.

Рисунок 2.2.- Скрипти опису інфраструктури (IaaC) для повторюваності вимірів

Для розгортання ж сервісів в хмарних провайдерах використаємо скрипти Terraform з офіційною реалізацією під кожного провайдера. Це дозволить розгортати аналогічні середовища іншим людям, подібно до Docker на локальному комп'ютері. Terraform також можна використати для локального розгортання в Docker, але це лише додаткова обгортка, що не впливає на результат роботи, тож для локального середовища використано Docker напряму для спрощення реалізації.

Усі ці додаткові технології дозволяють зробити підхід описаний у цій роботі повторюваним на інших середовищах, адже наприклад віртуалізація Docker, фреймворк .NET Core 6 та мова Python 3 можуть виконуватись на різноманітних середовищах, зокрема Windows, Mac, Linux.

Практичний комерційний досвід з цими технологіями дозволить ефективно використати їх у цій роботі, а результати практичного застосування різних сервісів додатково підкріпить теоретичні напрацювання.

2.2 Вибір та реалізація інструментарію вирішення задачі

Для порівняння наведених технології та рішень для зберігання та обміну файлів використаємо розробимо скрипти з використанням офіційних Software Development Kit (SDK) для доступу через програмний інтерфейс API та проведемо тести для виміру швидкодії цих технологій при роботі з різного розміру файлами. У випадку платних продуктів, використаємо також публічні дані про ціну таких рішень при їх впровадженні у розробці комерційних сервісів і також порівняємо між собою. Оцінимо переваги, недоліки та особливості кожної технології при її використанні у тій чи іншій ситуації, щоб мати можливість швидко приймати рішення про її впровадження.

Також буде використано практичний досвід, набутий під час проходження переддипломної практики в компанії з розробки програмного забезпечення, де було розроблено комерційний продукту, частина функціоналу якого власне і опиралась на збереження та передачу файлів різного розміру та кількості між веб сервісами.

Отже, для Google Drive використаємо середовище Jupyter Notebook та скрипт на мові Python 3, підключимо туди Google Drive як додатковий диск на файловій системі та виконаємо читання файлу звідти декілька разів та виміряємо швидкодію на рисунку 2.3.

```
from google.colab import drive
drive.mount('/mnt/drive')
!ls '/mnt/drive/MyDrive/data/models'

import time

start_time = time.time()

with open('/mnt/drive/MyDrive/data/models/model103', 'rb') as f:
    data = f.read()
    print(len(data))

end_time = time.time()
elapsed_time = end_time - start_time
print(f"Elapsed time: {elapsed_time} seconds")
```

Рисунок 2.3. - Python скрипт

Також порівняємо це із завантаженням того ж файлу на локальну машину, і використаємо дані з публічної документації Google Drive щодо можливостей та цін на цей продукт. Аналогічну процедуру застосуємо для Microsoft OneDrive та Dropbox.

Додатково використаємо наступний Python скрипт у тому самому середовищі щоб згенерувати 100 JSON файлів невеликого розміру (до 1 КБ) та завантажимо їх локально для подальших тестів на рисунку 2.4.

```
import json
import os
import random
import string

def random_string(length):
    letters = string.ascii_letters + string.digits + string.punctuation
    return ''.join(random.choice(letters) for _ in range(length))

def generate_random_fields():
    num_fields = random.randint(3, 10)
    fields = {}
    for i in range(num_fields):
        field_name = f"field_{i+1}"
        field_value = random_string(random.randint(5, 20))
        fields[field_name] = field_value
    return fields

if not os.path.exists('/mnt/test1'):
    os.makedirs('/mnt/test1')

for i in range(1, 101):
    filename = f"/mnt/test1/file_{i}.json"
    data = {
        "name": f"file_{i}",
        "description": random_string(random.randint(10, 100)),
        **generate_random_fields()
    }

    with open(filename, 'w') as file:
        json.dump(data, file, indent=4)

print("JSON files created successfully.")

import zipfile
import os

def zip_folder(folder_path, output_path):
    with zipfile.ZipFile(output_path, 'w', zipfile.ZIP_DEFLATED) as zipf:
        for root, dirs, files in os.walk(folder_path):
            for file in files:
                file_path = os.path.join(root, file)
                zipf.write(file_path, os.path.relpath(file_path, folder_path))

folder_to_zip = '/mnt/test1'  # Change this to the folder you want to zip
output_zip = '/mnt/test1.zip'  # Change this to the desired output zip file name
zip_folder(folder_to_zip, output_zip)
print(f"Folder '{folder_to_zip}' zipped successfully to '{output_zip}'")
```

Рисунок 2.4. Python скрипт для генерації JSON файлів невеликого розміру

Наступним протестуємо хмарне рішення від Microsoft - Azure Blob Storage

завантажуючи на нього різні файли з локального комп'ютера за допомогою програми написаної на C#, фреймворку .NET Core 6 та офіційної Nuget бібліотеки з SDK від Azure на рисунку 2.5

```
var blobServiceClient = new BlobServiceClient(connectionString);
var containerClient = blobServiceClient.GetBlobContainerClient("container1");
var blobClient = containerClient.GetBlobClient("model103");

var stopwatch = Stopwatch.StartNew();
await blobClient.UploadAsync("C:\\Users\\ivask\\Downloads\\model103", true);
stopwatch.Stop();
Console.WriteLine($"File uploaded to Azure Blob Storage in {stopwatch.ElapsedMilliseconds}ms.");

var containerClient2 = blobServiceClient.GetBlobContainerClient("container2");
stopwatch = Stopwatch.StartNew();
foreach (var fpath in Directory.EnumerateFiles("C:\\Users\\ivask\\Downloads\\test1"))
{
    using var f = File.OpenRead(fpath);
    await containerClient2.UploadBlobAsync(Path.GetFileName(fpath), f);
}
stopwatch.Stop();

Console.WriteLine($"Files uploaded to Azure Blob Storage in {stopwatch.ElapsedMilliseconds}ms.");
```

Рисунок 2.5. – Приклад тестування хмарного рішення Azure Blob

Для розгортання та конфігурацію власне середовища у Azure хмарі використаємо скрипт на Terraform з модулем для Azure Cloud та мовою розмітки YAML на рисунку 2.6.

```
provider "azurerm" {
  features {}
}

resource "azurerm_storage_account" "example" {
  name                     = "teststorage123av"
  resource_group_name      = "my_rg1"
  location                 = "West US 2"
  account_tier             = "Standard"
  account_replication_type = "LRS"
}

resource "azurerm_storage_container" "example" {
  name                  = "container1"
  storage_account_name  = azurerm_storage_account.example.name
  container_access_type = "blob"
}

resource "azurerm_storage_container_anonymous_access_policy" "example" {
  storage_account_name = azurerm_storage_account.example.name
  container_name       = azurerm_storage_container.example.name
  enabled              = true
}

resource "azurerm_storage_container" "example2" {
  name                  = "container2"
  storage_account_name  = azurerm_storage_account.example.name
  container_access_type = "blob"
}

resource "azurerm_storage_container_anonymous_access_policy" "example2" {
  storage_account_name = azurerm_storage_account.example.name
  container_name       = azurerm_storage_container.example2.name
  enabled              = true
}
```

Рисунок 2.6. - Azure Blob Terraform

Додатково візьмемо відомості з Azure Pricing Calculator та їх офіційної документації про переваги та ціни на послуги.

Тепер протестуємо хмарне рішення від Amazon - AWS (Amazon Web Services) S3 (Simple Storage Service) завантажуючи на нього ті самі файли з локального комп'ютера, що й у хмару Azure, також допомогою програми написаної на C#, фреймворку .NET Core 6 та офіційної Nuget бібліотеки з SDK від AWS на рисунку 2.7.

```
var credentials = new
Amazon.Runtime.BasicAWSCredentials(accessKey, secretKey);
using var client = new AmazonS3Client(credentials,
RegionEndpoint.EUCentral1);
var fileTransferUtility = new TransferUtility(client);

var stopwatch = Stopwatch.StartNew();
await
fileTransferUtility.UploadAsync("C:\\Users\\ivask\\Downloads\\mode
l103", "testbucket123av2", $"folder1/model103");
stopwatch.Stop();
Console.WriteLine($"File uploaded to S3 bucket in 'folder1' folder
in {stopwatch.ElapsedMilliseconds}ms.");

stopwatch = Stopwatch.StartNew();
foreach (var fpath in
Directory.EnumerateFiles("C:\\Users\\ivask\\Downloads\\test1"))
{
    await fileTransferUtility.UploadAsync(fpath,
"testbucket123av2", $"folder2/{Path.GetFileName(fpath)}");
}
stopwatch.Stop();

Console.WriteLine($"Files uploaded to S3 bucket in 'folder2'
folder in {stopwatch.ElapsedMilliseconds}ms.");
```

Рисунок 2.7. - AWS S3 скрипт

Для розгортання та конфігурацію подібного середовища у AWS хмарі аналогічно використаємо скрипт на Terraform з модулем для AWS Cloud та мовою розмітки YAML на рисунку 2.8.

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_s3_bucket" "example" {
  bucket = "testbucket123av"
  acl    = "public-read"

  policy = <<POLICY
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "${aws_s3_bucket.example.arn}/*"
      ]
    }
  ]
}
POLICY
}

resource "aws_s3_bucket_object" "folder1" {
  bucket = aws_s3_bucket.example.bucket
  key    = "folder1/"
  source = "/dev/null"
}

resource "aws_s3_bucket_object" "folder2" {
  bucket = aws_s3_bucket.example.bucket
  key    = "folder2/"
  source = "/dev/null"
}
```

Рисунок 2.8. - AWS S3 Terraform скрипт

Додатково візьмемо відомості з AWS Pricing Calculator та їх офіційної документації про переваги та ціни на послуги.

Для аналізу MongoDB, використаємо Docker для розгортання віртуального середовища з цією базою даних та налаштуємо його на рисунку 2.9.

```
docker run -d -p 27017:27017 --name my_mongo mongo
```

Рисунок 2.9. - Docker скрипт

Після цього завантажимо деякі об’єкти як JSON-файли через програму написану на C#, фреймворку .NET Core 6 та офіційного MongoDB C# драйвера в

якості бібліотеки на рисунку 2.10.

```
var connectionString = "mongodb://localhost:27017";
var client = new MongoClient(connectionString);
var database = client.GetDatabase("test_db");
var collection = database.GetCollection<BsonDocument>("files");

var messagesBatch =
Directory.EnumerateFiles("C:\\Users\\ivask\\Downloads\\test1")
    .Select(File.ReadAllText)
    .Select(BsonDocument.Parse);

var stopwatch = Stopwatch.StartNew();
await collection.InsertManyAsync(messagesBatch);
stopwatch.Stop();

Console.WriteLine($"Files saved in
{stopwatch.ElapsedMilliseconds}ms.");
```

Рисунок 2.10. - MongoDB скрипт

Для аналізу Apache Kafka аналогічно використаємо Docker Compose скрипт для розгортання та налаштування на рисунку 2.11.

```
version: '2'
services:
  zookeeper:
    image: confluentinc/cp-zookeeper:latest
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000
    ports:
      - 22181:2181

  kafka:
    image: confluentinc/cp-kafka:latest
    depends_on:
      - zookeeper
    ports:
      - 29092:29092
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS:
PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP:
PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1


docker compose up -d
```

Рисунок 2.11 - Apache Kafka скрипт

А тоді за допомогою програмного стеку на C# + .NET Core 6 зробимо тести вставки багатьох маленьких файлів-повідомлень в цей брокер на рисунку 2.12.

```
var config = new ProducerConfig
{
    BootstrapServers = "localhost:29092",
    BatchNumMessages = 100,
};

using var adminClient = new AdminClientBuilder(config).Build();
await adminClient.CreateTopicsAsync(new TopicSpecification[]
{
    new TopicSpecification { Name = "files", NumPartitions = 1,
ReplicationFactor = 1 }
});

using var producer = new ProducerBuilder<Null,
string>(config).Build();

var messagesBatch =
Directory.EnumerateFiles("C:\\Users\\ivask\\Downloads\\test1")
    .Select(File.ReadAllText)
    .Select(x => new Message<Null, string>
    {
        Value = x
    });

var stopwatch = Stopwatch.StartNew();
foreach (var message in messagesBatch)
{
    await producer.ProduceAsync("files", message);
}
producer.Dispose();
stopwatch.Stop();

Console.WriteLine($"Files sent in
{stopwatch.ElapsedMilliseconds}ms.");
```

Рисунок 2.12. - C# Apache Kafka скрипт для тестування вставки файлів-повідомлень

В даному випадку Apache Kafka використовується як брокер повідомлень з доволі простою топологією - одна нода, з однією темою (topic), куди програма буде записувати повідомлення. Аналогічно буде зроблено і для наступних

брокерів чи сервісів для обміну повідомленнями чи сповіщеннями, з єдиною різницею, що десь це називається темою (topic), а десь це просто черга (queue).

Тепер на основі попередніх скриптів на Terraform аналогічно налаштуємо для Azure компонент Service Bus на рисунку 2.13.

```
provider "azurerm" {
  features {}
}

resource "azurerm_servicebus_namespace" "example" {
  name                = "servbus123va"
  location            = "West US 2"
  resource_group_name = "my_rg1"

  sku {
    tier = "Basic"
  }

  default_action {
    action = "Allow"
  }

  network_rule_set {
    default_action = "Allow"
  }

  authorization_rule {
    name         = "listen-rule"
    rights       = ["Listen"]
    listen       = true
    send         = false
    manage       = false
    has_primary = true
  }

  local_authentication {
    enabled = true
  }
}

resource "azurerm_servicebus_queue" "example" {
  name                = "queue1"
  namespace_name      = azurerm_servicebus_namespace.example.name
  resource_group_name = azurerm_resource_group.example.name
}
```

Рисунок 2.13. – Налаштування компоненту Service Bus для Azure

Після цього використаємо наступний код програми для вставки файлів-повідомлень в ці черги на рисунку 2.14.

```
await using var client = new ServiceBusClient(connectionString);
var sender = client.CreateSender("queue1");

var messagesBatch =
Directory.EnumerateFiles("C:\\Users\\ivask\\Downloads\\test1")
    .Select(File.ReadAllText)
    .Select(x => new
ServiceBusMessage(Encoding.UTF8.GetBytes(x)));

var stopwatch = Stopwatch.StartNew();
await sender.SendMessagesAsync(messagesBatch);
stopwatch.Stop();

Console.WriteLine($"Files sent in
{stopwatch.ElapsedMilliseconds}ms.");
```

Рисунок 2.14. - Service Bus C# скрипт

Аналогічно повторимо для Amazon компоненту SQS (Simple Queue Service) на рисунку 2.15.

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_sqs_queue" "queue1" {
  name                      = "queue1"
  delay_seconds             = 0
  max_message_size          = 262144
  message_retention_seconds = 345600  # 4 days
  visibility_timeout_seconds = 30
  receive_wait_time_seconds = 0
  fifo_queue                = false  # Standard queue type

  # Encryption configuration - disabled in this case
  kms_master_key_id         = ""
  kms_data_key_reuse_period_seconds = 0

  # Access policy - Basic type
  policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": "*",
      "Action": "SQS:SendMessage",
      "Resource": "${aws_sqs_queue.queue1.arn}"
    }
  ]
}
EOF
}
```

Рисунок 2.15 - AWS SQS Terraform

Та програмний код на рисунку 2.16.

```
var credentials = new BasicAWSCredentials(accessKey, secretKey);
var sqsConfig = new AmazonSQSConfig { RegionEndpoint =
RegionEndpoint.EUCentral1 };
using var sqsClient = new AmazonSQSClient(credentials, sqsConfig);

var messagesBatch =
Directory.EnumerateFiles("C:\\Users\\ivask\\Downloads\\test1")
    .Select(File.ReadAllText)
    .Select(x => new SendMessageBatchRequestEntry
    {
        Id = Guid.NewGuid().ToString(),
        MessageBody = x
    });

var stopwatch = Stopwatch.StartNew();
foreach (var c in messagesBatch.Chunk(10))
{
    var request = new SendMessageBatchRequest
    {
        QueueUrl = queueUrl,
        Entries = c.ToList()
    };

    await sqsClient.SendMessageBatchAsync(request);
}
stopwatch.Stop();

Console.WriteLine($"Files sent in
{stopwatch.ElapsedMilliseconds}ms.");
```

Рисунок 2.16. - AWS SQS C# скрипт

Також використаємо інформацію про актуальні можливості та ціни на ці компоненти таким самим методом як і для Azure Blob Storage та AWS S3.

Для розгляду CDN (Content Delivery Network) використаємо популярні сервіси такі, як Youtube, Discord, Nebula. Зокрема, Youtube та Nebula - це стрімінгові сервіси відео, які використовують декілька рівнів CDN-нод для реплікації та кешування контенту по всьому світу та поближче до кінцевих користувачів. Тож порівняємо швидкість завантаження та наявність буферизації відео різної якості та популярності, адже останнє напряму впливає на

ймовірність потрапляння його у кеш. Додатково ознайомимось з цінами на послуги цього сервісу від популярних хмарних провайдерів таких, як Azure та Cloudflare. У випадку месенджера Discord порівняємо швидкість завантаження файлу, який було використано для тестування Google Drive.

Далі роглянемо технологію Peer-to-peer (P2P) передачі даних. На жаль, BitTorrent набрав свою популярність саме через завантаження піратського контенту, адже можливість скачувати якийсь файл по частинах від декількох користувачів, а не лише оригінального сервера, суттєво пришвидшував завантаження, особливо раніше, коли сервери ще не могли легко задовольнити всіх користувачів. Ще однією очевидною перевагою є децентралізація та репліки файлу у декількох користувачів, що його завантажили, тобто якщо оригінальний сервер видалить цей файл, то він всерівно буде доступний для відвантаження іншим користувачам. Але у цієї технології є і недоліки, як наприклад те, що вона добре працює лише з популярними файлами, оскільки швидкодія напряму залежить від кількості на якості сідів (seed) котрі власне і роздають шматки файлу для скачування.

Тому як приклад використання цієї технології використаємо ігрову платформу Steam - вона дозволяє завантажувати ігри та оновлення до них не лише зі своїх серверів та допоміжних CDN-нод, але й від інших користувачів поблизу, що значно покращує швидкодію. Використаємо метрики, що надаються самим Steam клієнтом при завантаженні якоїсь гри.

Для аналізу IPFS (InterPlanetary File System) - піднімемо одну ноду за допомогою наступного Docker скрипта на рисунку 2.17.

```
docker run -d --name ipfs_host -v
C:\Users\ivask\Documents\asdasdasd:/mnt/test1 -p 4001:4001 -p
4001:4001/udp -p 127.0.0.1:8080:8080 -p 127.0.0.1:5001:5001
ipfs/kubo:latest
```

Рисунок 2.17. - Docker IPFS скрипт

Далі, завантажимо туди різні файли та проведемо заміри часу - це

включатиме в себе внутрішнє подрібнення файлу на частини, та обрахування необхідних хешів на рисунку 2.18.

```
read up rest </proc/uptime; \
t1="${up%.*}${up#*.}"; \
ipfs add /mnt/test1/model103; \
read up rest </proc/uptime; \
t2="${up%.*}${up#*.}"; \
echo "Execution took: $(( 10*(t2-t1) )) ms";
```

Рисунок 2.18 - IPFS Shell скрипт

Огляд та застосування зазначених вище технологій дозволяє перейти до наступного етапу виконання завдань кваліфікаційної роботи.

2.3 Практичне використання технологій та результати

Результати дослідження хмарної платформи буде також додатково доповнено власними практичними знаннями та метриками здобутими під проходження практики в компанії з розробки програмного забезпечення в розрізі теми цієї роботи. Тут передача даних під час реплікації фактично і є зберіганням, обробкою та подальшим обміном файлів різного розміру та кількості веб-сервісів.

Ця компанія отримала замовлення на створення програмного забезпечення для моніторингу віддалених пристроїв – розподіленої платформи заснованої на принципах Інтернету речей (IoT), яка дозволяє збирати та аналізувати дані з кінцевих підстанцій для подальшого використання операторами загальної системи. Основною метою було розробити ефективну та масштабовану систему моніторингу для забезпечення операторів центральної станції надійною та швидкою інформацією про стан віддалених пристроїв.

Проаналізувавши вимоги, було прийнято рішення розробляти систему на розподіленій архітектурі, де кожна кінцева підстанція є незалежним вузлом системи. Кожна підстанція відповідає за збір та передачу даних в центральну

станцію. Забезпечується висока доступність та надійність шляхом резервування підключень та механізмів повтору в разі втрати зв'язку.

Центральна станція відповідає за прийом, аналіз та зберігання даних від усіх кінцевих підстанцій. Реалізована високопродуктивна система для оптимальної обробки великого обсягу даних. Здійснює реплікацію даних на кінцеві підстанції для підтримки аналізу в обох напрямках.

Всі взаємодії між кінцевими підстанціями та центральною станцією базуються на безпечних та ефективних протоколах зв'язку. Використовуються протоколи HTTPS та SFTP для передачі даних через захищене з'єднання, що гарантує конфіденційність та цілісність даних. Для забезпечення високої продуктивності використовується асинхронна комунікація між компонентами системи. Кожен вузол може відправляти та отримувати повідомлення в асинхронному режимі, що дозволяє збільшити швидкість обміну даними та зменшити затримки.

Усі взаємодії та події в системі моніторяться та логуються для наступного аналізу та відлагодження. Використовуються механізми логування для виявлення помилок, а також для аналізу використання ресурсів системи. Усі передані дані шифруються з використанням сучасних криптографічних алгоритмів для захисту конфіденційності та цілісності інформації. Враховуються сучасні стандарти безпеки для захисту від можливих загроз.

Система забезпечує ефективний та масштабований механізм збору даних з кінцевих підстанцій. Кожна підстанція обладнана агентом збору даних, який відповідає за зчитування параметрів віддалених пристроїв та їх передачу до центральної станції. Спочатку планувалось використання протоколу MQTT (Message Queuing Telemetry Transport) для ефективного та надійного збору даних. Цей легкий та масштабований протокол дозволяє передавати дані в реальному часі та забезпечує оптимальну взаємодію між кінцевими підстанціями та центральною станцією. Але на жаль, через певні обмеження зі сторони

бізнесу, довелось використати повільну класичну модель Запит-Відповідь (Request-Response) на базі HTTPS з використанням long- та short-polling.

За допомогою криптографічних сертифікатів забезпечено механізми контролю доступу до даних для забезпечення конфіденційності. Кожен агент має унікальні ідентифікатори та ключі доступу, що гарантує лише авторизованим пристроям можливість передачі даних.

Система реалізує двонаправлену реплікацію даних між кінцевими підстанціями та центральною станцією. Використовується гібрид технологій Change Data Capture (CDC) та рішення на базі тригерів, що дозволяє визначити та реплікувати тільки змінені дані, зменшуючи обсяг передачі та покращуючи ефективність. Це також додає гнучкості в обробці цих даних та додаткових можливостей, як наприклад забезпечення логічної цілісності даних - підстанція навіть не отримає даних яких у ній не має бути згідно визначених бізнес залежностей. Забезпечено систему резервного копіювання для збереження інтегрітету даних та можливості відновлення в разі випадкового втрати або пошкодження інформації. Регулярно проводяться аудити та тести відновлення для перевірки працездатності системи.

Центральна станція також має бути обладнана веб-застосунком для обробки та аналізу даних, яка дозволяє виконувати складні операції обробки і агрегації інформації та забезпечити інструменти для статистичного аналізу, трендів та прогнозування змін в даних. Система включає автоматизовані механізми попередження, які виявляють аномалії чи проблеми. Оператори отримують сповіщення у реальному часі про потенційні проблеми, що дозволяє швидко реагувати та уникати негативних наслідків. В системі існують інструменти для виконання запитів та аналізу для специфічних сценаріїв. Оператори можуть визначати та виконувати складні запити для виявлення конкретних подій чи станів пристроїв.

Веб-інтерфейс також дозволяє операторам використовувати фільтри та групування для аналізу даних та швидкого виділяти ключові аспекти та

здійснювати глибший розбір інформації за визначеними параметрами. Забезпечено збереження історії та архіву даних для аналізу та вивчення тенденцій протягом часу, що дозволяє виконувати порівняльний аналіз та виявляти довгострокові зміни в стані системи.

Розподілена платформа підтримує динамічне приєднання нових підстанцій в режимі реального часу. Під час приєднання нової підстанції система автоматично реєструє та конфігурує її, додаючи до загальної мережі моніторингу. Оператори системи мають можливість гнучкої настройки та керування підстанціями. Це включає в себе встановлення параметрів, конфігурацій та прав доступу для кожної підстанції, забезпечуючи індивідуальний підхід до керування.

Кожна підстанція обладнана системою моніторингу, яка постійно спостерігає за її станом. За допомогою агентів система виявляє можливі проблеми, такі як втрата зв'язку, мало дискового простору чи інші аномалії. У разі виявлення проблеми, система автоматично генерує сповіщення та повідомлення для операторів, що дозволяє майже в режимі реального часу реагувати на ситуації та усувати неполадки як напряму так і направлення контракторів фізично на підстанції. Система підтримує автоматичне видалення підстанцій у разі виявлення несправностей чи інших критичних ситуацій. Також існує можливість автоматичного повторного приєднання підстанцій, які раніше були видалені, після усунення проблеми чи установки нового обладнання.

Основну інфраструктурну частину проекту складає саме реплікація, а саме оркестрація пересилання змін баз даних від однієї до іншої. Зокрема, на рисунку 2.19 зображена спрощена система комунікації між двома терміналами:

Система була побудована з використанням технології .NET та екосистеми продуктів Microsoft, включаючи хостинг центрального сервера в хмарному середовищі Azure з використанням його сервісів. Тож розробка передачі даних для реплікації також була виконана з залученням компонентів Azure, завдяки

чому вдалось зібрати певну кількість метрик щодо ефективності впровадженого рішення в подальшому.

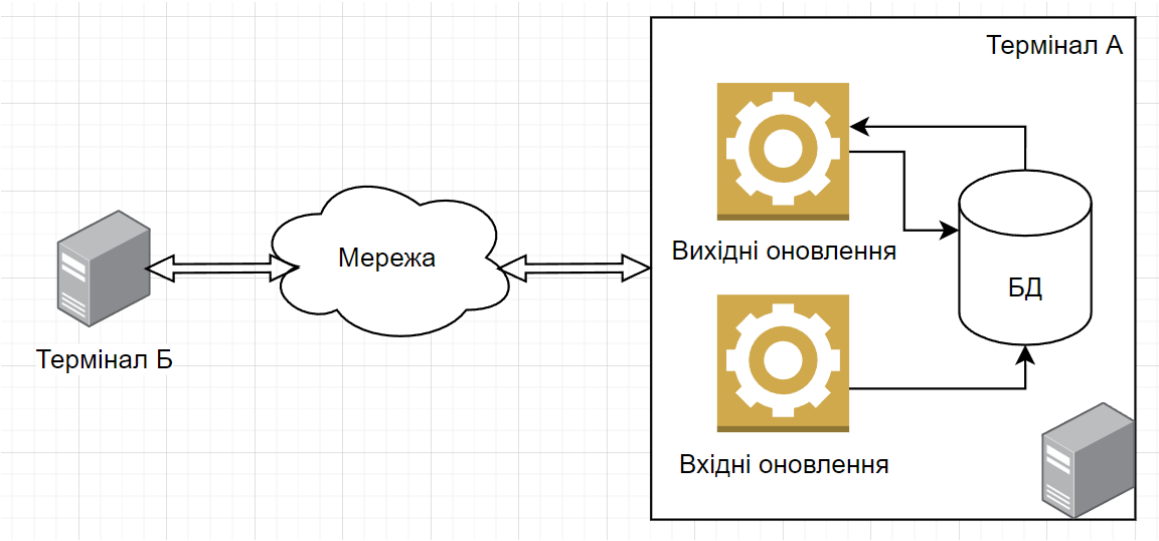

Рисунок 2.19 - Схема реплікації (комунікації між двома терміналами)

Загалом було визначено два основні сценарії роботи реплікації, а відповідно й два основні види даних для обміну між центральної та кінцевих підстанцій.

Перший - це близька до реального часу інформація про нові зміни бази даних, які генерувались неперервно під час роботи підстанції та мали незначні флуктуації у кількості згенерованих даних, тобто навантаження було доволі прогнозоване з деякими сплесками час від часу. Ці дані зберігались як JSON-документи, тоді буферизувались перед відправкою і нарешті відправлялись у хмару для подальшої обробки.

Для забезпечення цілісності даних використовувалась техніка щонайменше однієї доставки (at least once delivery), а також на приймаючій базі даних одні дані додатково дедублікувались перед вставкою, а іншi були за своєю природою ідемпотентні, що гарантувало консистентність реплікованих даних.

Такі JSON-документи надсилались періодично залежно від налаштувань сервера. Вони були доволі малого розміру (часто не більше 1-2 КБ) та

відправлялись пачками з буферу, використовуючи додатково Brotli стиснення. Оскільки в одну партію відправлення часто потрапляли дані про зміни в одній і тій же табличці, або суміжних, а відправка зазвичай була по декілька десятків файликів в секунду з однієї підстанції - було досягнуто доволі хороших результатів стиснення. А саме, згідно наших метрик, співвідношення стиснення становив близько 3-4:1 (стиснення без втрат), що значно пришвидшило процеси відвантаження та завантаження даних, оскільки більшість кінцевих терміналів були суттєво обмежені в трафіку та часто були доволі далеко географічно.

На стороні хмари Azure завдяки зменшенню загального розміру файлів ми могли комфортно обробляти декілька сотень таких терміналів одночасно ефектично використовуючи кошти на таку інфраструктуру. Отримані стиснені JSON документи пачками далі зберігались разом з певною метадатою у Azure Cosmos DB. Менший розмір значно знизив витрати, оскільки це доволі дорогий компонент, в якому крім процесорного часу також потрібно платити близько 25 центів за зберігання 1 ГБ даних.

Зберігання стиснених пачок файлів також дозволило обробляти їх пачками транспортуючи їх через Azure Service Bus до Azure Functions які власне й оркестрували весь процес реплікації і оновлювали базу станцій. Схему обробки цих файлів можна побачити на рисунку 2.20.

Другий вид даних для реплікації був складнішим. Він застосовувався, коли від'єднану станцію приєднували назад, при цьому маючи якісь дані на ній. Таке найчастіше ставалось під час міграцій із старої системи на нову розроблену платформу. Важливо також було врахувати час затримок бізнес процесів на самій підстанції під час такої міграції - потрібно було дочекатись завершення цього процесу.

В даному випадку із повністю всієї бази підстанції генерувались JSON-документи у форматі змін, але фактично описуючи всі дані. Цей процес було зроблено асинхронним, тому станцією могли продовжувати користуватись і оновлення збирались в чергу до кінця міграції. Цих даних було багато, по

декілька ГБ (мільйони рядків бази даних) на підстанцію. Вони також буферувались у пачки, але вже більші - тисячі JSON-файликів в одній групі залежно від налаштувань. При цьому рядки описувались з табличок послідовно, тож ступінь стиснення була більша за 10:1 в цьому випадку.

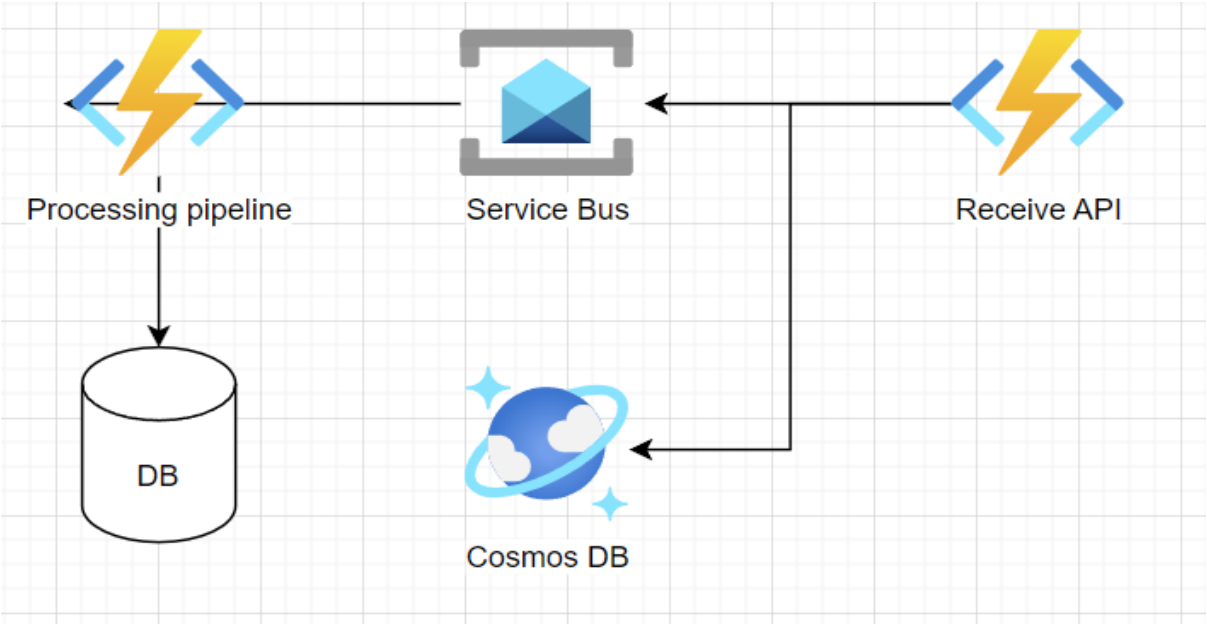

Рисунок 2.20 - Перший тип реплікації

Додатково було розділено дані на декілька груп залежно від вимог бізнесу - критичні, та інші. Критичні дані мусили обов'язково мігруватись першими, та підстанція мусила очікувати завершення реплікації. Інші ж дані вже імпортувались після критичних але перед всіма новими змінами, та не перешкоджали роботі.

Загалом файли стиснені до декількох МБ вже доволі стабільно відвантажувались по мережі, рідко коли виникали повторні надсилання. Але використовувати Azure Cosmos DB було нереалістично дорого з такою кількістю даних та дуже точковими сплесками навантаження, а також доводилось би розпаковувати дані перед відправкою через Azure Service Bus. Тож після певного аналізу було вирішено використати Azure Blob Storage як повільний але дешевий варіант для зберігання великої кількості файлів більшого розміру. Зокрема це коштувало близько 2 центів за 1 ГБ даних.

В цьому випадку додаткові метадані для обробки були відокремлені від файлів та зберігались в Azure Cosmos DB для швидкої обробки. Їх малий розмір (до 100 Б на 1 метадату) дозволив також їх надсилати через Azure Service Bus до Azure Functions, які вже під час обробки даних безпосередньо розпаковували стриснені файли з Azure Blob Storage та вносили зміни в базу даних.

Зокрема, ось схема цього типу реплікації зображена на рисунку 2.21.

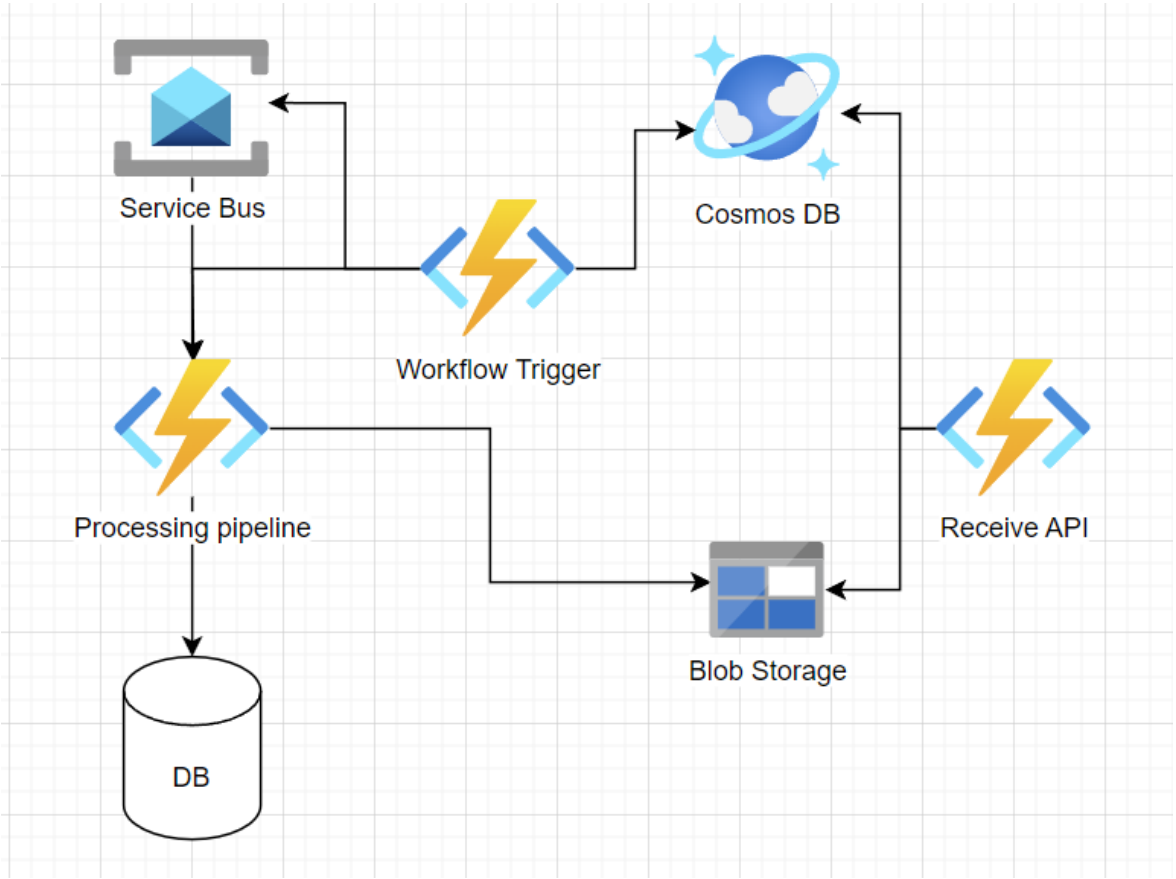

Рисунок 2.21 - Другий тип реплікації

В результаті можна спостерігати скорочення навантаження на копіювання даних по мережі, оскільки тепер вони вичитувались лише 1 раз відразу перед опрацюванням.

РОЗДІЛ 3.
РЕЗУЛЬТАТИ ВИРІШЕННЯ ЗАДАЧІ

3.1 Рішення орієнтовані на кінцевого користувача

Отже, під час тестування було використано 1 файл більшого розміру 93.5 МБ - контент А, а також 100 JSON файлів меншого розміру, загальним розміром 30 КБ - контент Б. Тести з завантаження, відвантаження та пересилання файлів виконувались 10 разів, а тоді бралось середнє значення за результат вимірювання. Також, наведені коди програм на C# запускались в оптимізованому режимі.

Google Drive - завантаження контенту А з хмари на локальний комп'ютер ~2.1 с. При цьому гугл диск можна приєднати як додаткове сховище до публічних онлайн машин від гугл Jupyter Notebook на платформі Collab - це доволі популярне рішення як для виконання простих Python скриптів, так і для тренування моделей для штучного інтелекту.

Швидкість завантаження контенту А з підв'язаного диску перший раз ~2.5 с., наступні близько 0.15 с. завдяки кешуванню. У Google Drive за замовчуванням доступно 15 ГБ простору безкоштовно, далі доступні опції на 100 ГБ за 69.99 грн\міс, 200 ГБ за 80 грн\міс та максимум в 2 ТБ за 229 грн\міс. При цьому доступні деякі знижки на випробувальний термін та при умові оплати одразу на рік вперед. Dropbox пропонує 2 ТБ за $11.99 / міс, 3 ТБ за $22 / міс, 9 ТБ за $24 / міс / користувача та 3 ТБ за кожного додаткового користувача до 1 ПБ, 15 ТБ за $32 / міс / користувача та 5 ТБ за кожного додаткового користувача до 1 ПБ.

Microsoft OneDrive має безкоштовний план на 5 ГБ, 100 ГБ за $1.99 / міс, 1 ТБ за $6.99 / міс, 6 ТБ на 6 користувачів за $9.99 / міс; а також комерційні пропозиції 1 ТБ за $5, $6 або $12.5 / міс / користувача залежно від опцій. Microsoft SharePoint пропонується в розмірі 1 ТБ за $5 або $12.5 / міс / користувача залежно від опцій (фактично ті самі плани, що і для OneDrive, оскільки вони включають в себе обидва продукти та ще деякі).

Отримані результати наведені у таблиці 3.1.

Таблиця 3.1. – Порівняльна характеристика сервісів

	Google Drive	Dropbox	Microsoft Sharepoint
Максимальний простір	2 ТБ	1000 ТБ (спільне)	1 ТБ (на користувача)
Мінімальна ціна за 1 ТБ простору	2 ТБ за 229грн/міс	2 ТБ за $11.99/міс	1 ТБ за $5/міс за користувача
Комерційні ліцензії	ні (для окремих користувачів)	так	так + інтеграція з продуктами Microsoft
Максимальний безкоштовний простір	15 ГБ	-	-

Стрімінгові сервіси такі як Youtube, Nebula використовують CDN для пришвидшення завантаження контенту для кінцевих користувачів, часто це може бути 2 рівні кешу на різному рівні. Це демонструється тим, що доволі популярні відео з багатьма переглядами мають великий шанс опинитись у кеші CDN поряд з користувачем, оскільки доволі ймовірно, що цей контент вже переглянули користувачі географічно поряд, які під'єднані до того самого серверу. Тож натискаючи на таке відео - для нового користувача відтворення відбувається майже миттєво, та початкова буферизація практично непомітна.

З іншого ж боку, якщо пошукати якесь відео, що має лише декілька переглядів та є не популярне - його з великою ймовірністю не буде у найближчому кеші до користувача, тож буде доволі помітне як завантаження початку відео, так і буферизація при ручному перемотуванні програвача на випадкові моменти у цьому відео.

Для прикладу наведемо спрощену схему дворівневого CDN кешу, що використовує Nebula на рисунку 3.1.

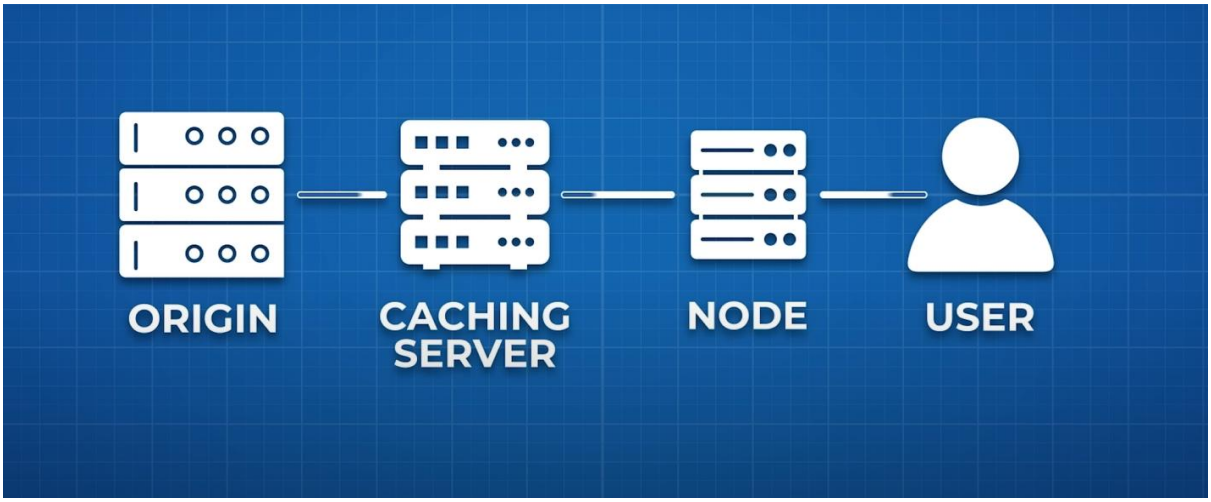

Рисунок 3.1. - CDN використання Nebula

Популярний ігровий месенджер Discord також використовує CDN для пришвидшення завантаження надісланих користувачами файлів та їх реплікацію в інші географічні регіони на наявні там CDN-ноди, а в базі даних зберігає лише метадані про ці файли. Оскільки в цьому месенджері є обмеження у 25 МБ на один файл, то було протестувано завантаження файлу на 24.2 МБ, що зайняло близько 0.8 с, тобто контент А міг би теоретично завантажитись за ~3.1 с.

Сервіс CDN від Azure з провайдером Verizon на стандартному тірі коштуватиме $0.08 / 1 ГБ / міс та покриватиме регіон Північної Америки та Європи (найпопулярніший та найдешевший регіон).

Ігрова бібліотека Steam також використовує CDN разом з Peer-to-peer (P2P) мережею для пришвидшення завантажень як самих ігор так і для їх оновлень (причому в стиснутому вигляді), зокрема одне з оновлень завантажувалось зі швидкістю 18.9 МБ / с, тобто контент А завантажився би теоретично до 5 с.

3.2 Рішення орієнтовані на програмне використання

Azure Blob Storage та Azure Service Bus було протестовано у двох різних регіонах: West US 2 - регіон А та Central Poland - регіон Б, при цьому підключення були з локального комп’ютера. Завантаження у Blob Storage в регіон А контенту А в середньому було ~15.9 с., а контенту Б ~18.5 с., в регіон Б контенту А ~3.1с., контенту Б ~4.4с. Відправка в Service Bus контенту Б в регіоні А зайняла ~4.2с., а у регіоні Б ~1с. Тут очевидний вплив віддаленого регіону на швидкість завантаження, а також те, що багато маленьких файлів сумарним об'ємом менше одного великого завантажились довше через затримку (latency) в мережі. Ціна Azure Blob Storage для Hot tier зі стандартною швидкодією становить близько $21.3 / міс / 1 ТБ при цьому можна зберігати файли практично довільного розміру. Користування Service Bus для обміну файлами на базовому тірі коштуватиме $0.05 / 1 мільйон операцій, але матиме обмеження у 256 КБ на розмір (можна збільшити до 100 МБ у дорожчому тірі), а Azure Cosmos DB для зберігання у серверлес (serverless) варіації обійдеться у $0.25 / 1 Request Unit (юніт запиту) та ще $0.25 / 1 ГБ / міс при максимальному розмірі файлу у 2 МБ.

Amazon AWS S3 та AWS SQS також протестувано у максимально подібних локаціях: West US 2 - регіон А та EU Central 1 - регіон Б, та аналогічні підключення з власного комп’ютера. Завантаження у S3 в регіон А контенту А в середньому було ~7.7 с., а контенту Б ~14.7 с., в регіон Б контенту А ~4.8 с., контенту Б ~8.7 с. Відправка в SQS контенту Б в регіоні А зайняла ~3.4с., а у регіоні Б ~1.1с.

Тут можемо спостерігати такий самий вплив на підключення в географічно далекий сервер. Сервери Azure та Amazon цілком можливо розміщені у фізично різних локаціях, тому швидкості відрізняються, зокрема регіон А швидший у порівнянні у сервісів від Amazon, а регіон Б - у Azure. Використання AWS S3 у стандартному тірі обійдеться в $23.55 / міс / 1 ТБ для зберігання файлів різноманітного розміру. AWS SQS може використовуватись для передачі файлів розміром до 256 КБ (аналогічно до Azure Service Bus, але без можливості

збільшення) за ціною в $0.4 / міс / 1 мільйон повідомлень, а AWS DynamoDB для зберігання файлів розміром до 400 КБ за $0.25 / 1 ГБ / міс. Наведемо результати у Таблиці 3.2 (швидкість в перших двох рядках зображена у вигляді регіон А / регіон Б).

Таблиця 3.2. – Порівняльна характеристика представлених результатів

	Azure Blob	AWS S3	Azure Service Bus	AWS SQS
Контент А	15.9с / 3.1с	7.7с / 4.7с	-	-
Контент Б	18.5с / 4.4с	14.7с / 8.7с	4.1с / 1.0с	3.4с / 1.1с
Орієнтовна ціна в міс	$21.3 / 1 ТБ	$23.55 / 1 ТБ	$0.05 / 10^6 оп	$0.4 / 10^6 оп
Максимальний розмір	-	-	256 КБ (100 МБ дорожчий)	256 КБ

База даних MongoDB може бути використана для збереження JSON файлів, тут є і комерційне рішення, але у цій роботі використано безкоштовний варіант із встановленням локально на комп'ютері цієї БД, зокрема в середовищі Docker. Завантаження контенту Б займає близько 0.5с. Важливим нюансом є те, що обмеження на розмір одного файлу становить 16 МБ. Apache Kafka - це безкоштовний брокер повідомлень з відкритим кодом (Open Source), який я протестував для обміну JSON файлів також встановивши локально, де відправка контенту Б становить ~2.7 с.

Для тестування децентралізованої файлової системи на основі блокчейну IPFS було розгорнуто одну ноду локально з використанням Docker. Ця технологія дозволяє працювати з файлами будь-якого розміру, але їй потрібен деякий час для обрахунку хешів, розбиття файлу на частини, дедублікацію цих частин та обрахування фінального ідентифікатора контенту (CID), за яким власне і можна відвантажувати цей файл іншим нодам у мережі.

Наведені системи представлено нижче у порівняльній таблиці 3.3.

Таблиця 3.3. – Порівняльна характеристика аналізованих систем

	Apache Kafka	IPFS	MongoDB
Контент Б	2.7с	0.9с	0.5с
Складність управління	середня	висока	низька
Тип	розподілена	децентралізована	централізована або розподілена
Ліцензія	Apache v2.0	MIT	SSPL v1.0

Загалом додавання контенту А у ході реалізації поставленого завдання зайняло ~1.2 с., а контенту Б ~0.9с. Хоч контент Б загалом в майже 3000 разів менший за контент А, доволі великі затримки викликані опрацюванням кожного файлу окремо. Це рішення також є безкоштовним, якщо його хостити самостійно.

РОЗДІЛ 4.
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА У НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Обґрунтування можливих чинників травмонебезпечних ситуацій

Офіс IT фірми представляє собою простір площею 200 квадратних метрів, розподілений на робочі зони та конференц-зали. Приміщення оснащене великою кількістю комп'ютерів, серверів та технічних пристроїв. Воно обладнане системами вентиляції та кондиціонування, а також резервними джерелами енергії для забезпечення неперервності роботи. В офісі створені умови для праці великої кількості людей, зокрема цілих команд разом із усім необхідним обладнанням у відкритих приміщеннях (open space).

Для зручності у окремій кімнаті облаштована міні кухня, з різноманітним приладдям для зберігання та розігрівання їжі чи приготування гарячих напоїв.

Електробезпека включає в себе перевірку стану електромережі. Може виникати перевантаження електричних мереж, зокрема доволі часто при теперішній нестабільному стані в енергосистемі країни, а також під час пікових годин, та наприклад у дуже холодні чи дуже гарячі дні, коли звичне навантаження на мережу збільшується. Небезпеку можуть представляти оголені проводи, не заземлене обладнання тощо.

Також виникає проблема довготривалого перебування за комп'ютером, що може призвести до проблем із зором та спиною, або при неправильному розположені рук - синдром карпального каналу.

Пожежна небезпека виникає при неправильному використанні кухонного приладдя та техніки, або часто як загоряння електроприладів, їх акумуляторів чи іншої проводки. Додатково, цю небезпеку представляють собою дизельні генератори, розташованні зовні будівлі для забезпечення резервного живлення обладнання.

Хімічна небезпека може виникати при задимленні приміщення, наприклад внаслідок пожежі чи загоряння. Деякі неприємні запахи можуть поширюватись по приміщенню із кухні, чи у випадку плавлення пластику електроприладів при їх невірному використанні.

4.2 Умови та обставини виникнення травмонебезпечних ситуацій та їх наслідки

Для покращення електробезпеки заплановані регулярні технічні перевірки електричної мережі та навчання персоналу правилам безпеки використання електротехніки. Також встановлені різноманітні реле напруги та пристрої захисного відключення.

Також передбачено організацію періодичних перерв у роботі за комп'ютером для зменшення впливу тривалого сидіння на здоров'я працівників. Зокрема, використовуються програмні продукти, що сповіщають про необхідність перерви за таймером.

Регулярно проводяться тренінги з пожежної безпеки, навчання персоналу алгоритмам дій у випадку надзвичайних ситуацій. Ці тренінги включать в себе інформацію про поводження з електропристроями, навчають застосуванню різних видів вогнегасників, конкретні дії при виникненні тої чи іншої надзвичайної ситуації, розташування додаткових виходів, сходових кліток та інше, а також контакти відповідальних осіб.

Для усунення неприємних запахів та дотримання чистоти та циркуляції повітря працює вентиляція. Додатково встановлені газоаналізатори та датчики, такі як виявлення чадного газу, щоб якомога скоріше сповістити про небезпеку, враховуючи, що чадний газ непомітний для людини.

Приділення уваги цим та іншим обставинам виникнення травмонебезпечних ситуацій дозволяє запобігти їх виникненню в майбутньому та зменшити рішень потенційних загроз і викликів для компанії.

4.3 Безпека в надзвичайних ситуаціях

Враховуючи можливість пожежі або відключення електропостачання, розроблений план евакуації та встановлені системи попередження аварій. Крім того, проводяться тренування з використання моделювання пожежних ситуацій. Додатково розташовані вогнегасники різних типів для використання, в тому числі, і працівниками, для цього вони проходять спеціальні тренінги.

Для критичного серверного обладнання передбачені онлайн (on-line) блоки безперебійного живлення, а зовні офіс обладнано промисловими генераторами із функціями автозапуску та супроводжуючої автоматики для безперебійного відновлення роботи. Також встановлені резервні системи зв'язку та інтернету для забезпечення з'єднання.

Ефективність заходів з безпеки та охорони праці забезпечується через постійну моніторингову систему безпеки на робочих місцях, аналіз звітів про інциденти та їхній вплив на стан безпеки працівників, а також оцінку рівня безпеки під час робочого процесу та їх адаптацію відповідно до змін в робочому середовищі.

В надзвичайних ситуаціях, таких як війна або конфлікт, забезпечення безпеки для багатьох компаній стає критично важливим завданням. IT-компанії повинні приділити особливу увагу заходам безпеки для захисту своїх систем та даних. Відповідальні особи мають здійснювати розробку і впровадження планів надзвичайних ситуацій для оперативного реагування та відновлення роботи під час кризових ситуацій, співпрацювати з відповідними органами державної влади.

Забезпечення безпеки працівників IT-компаній в умовах війни або конфлікту включає ряд важливих аспектів, які охоплюють фізичну безпеку, цифрову безпеку та організаційні заходи.

Не менш важливим питанням окрім захисту працівників є забезпечення збереження цілісності інфраструктури та даних, захисту фізичних інфраструктур від несанкціонованого доступу, включаючи забезпечення безпеки серверних приміщень та дата-центрів.

Важливо приділити увагу і надати рекомендацій та засоби для фізичної безпеки працівників, які можуть опинитися в зоні конфлікту, опрацювати можливості для евакуації та надання консультацій.

Компанія може проводити тренінги та навчання з метою опрацювання алгоритму дій захисту працівників в умовах настання надзвичайної ситуації, військових дій та інших потенційних чи непередбачуваних загроз.

Доцільно передбачити застосування безпечних комунікаційних каналів для обміну конфіденційною інформацією при роботі над важливими IT-проектами в умовах військових загроз та викликів.

В умовах повітряних загроз працівники, які працюють безпосередньо в офісі, мають мати доступ до сховищ, які обладнані та сертифіковані із врахуванням вимог та рекомендацій. У випадку потреби має бути забезпечена можливість віддаленої роботи для працівників, що дасть змогу уникнути потенційних небезпек при виконанні роботи.

РОЗДІЛ 5.
ВИЗНАЧЕННЯ ЕФЕКТИВНОСТІ

Визначення ефективності процесу дослідження та вибору технологій зберігання та обміну файлами веб-сервісів є важливим етапом кваліфікаційної роботи. З представлених попередніх результатів можна побачити, що для зберігання власних файлів, чи колаборації з декількома розробниками можна використати прості та умовно безкоштовні рішення, як Google Drive. У якості комерційного варіанту слід розглянути Dropbox та Microsoft Sharepoint, зокрема останній зазвичай використовується у великих організаціях, оскільки велика ймовірність купівлі ліцензії одразу на декілька продуктів від Microsoft та інтеграції між ними. Ці рішення, як і Google Drive підтримують програмний інтерфейс (API) для автоматизації певних задач. Зокрема, гугл диск ефективно показав себе при роботі у Google Collab Jupyter Notebook - швидкість завантажень зросла в рази після першого кешування, що часто застосовується для тренування різних моделей штучного інтелекту, де такі оптимізації на завантаження даних надзвичайно важливі.

Для створення власного продукту варто використовувати або власну інфраструктуру або хмарні рішення. Так наприклад, для збереження великих файлів, відео, тощо можна застосовувати як і файлову систему напряму на сервері чи у кластері як Kubernetes, так і певні хмарні рішення як Azure Blob Storage та Amazon AWS S3. Веб-сервіси можуть обмінюватись між собою за допомогою цих рішень, як наприклад один сервіс записує дані, інший його читає та обробляє і т.д. Тут варто пам'ятати про ціну таких продуктів та подбати за мінімізацію використаної пам'яті, наприклад за допомогою кешування, стискання та стрімінгу, що додатково може скоротити операційні витрати на ресурси самого серверу. Ці сервіси також підтримують реплікацію в інші регіони, ближче до користувачів. Для відвантаження різноманітних файлів для кінцевих користувачів можна скористатись протоколами HTTP(S), gRPC-Web,

web-sockets, в той час як між сервісами можна застосовувати і FTP(S). А за допомогою CDN (Content Delivery Network) можна ще пришвидшити завантаження поширеного або популярного контенту як відео, фото тощо до віддалених користувачів.

Зберігання менших за розміром файлів, особливо для швидкого пошуку, редагування та роботи з ними можна здійснити за допомогою баз даних призначених для цього. Це може бути MongoDB (як піднята у власному кластері так і платна версія від Atlas), Azure Cosmos DB, Amazon AWS DynamoDB. Ці рішення є відповідно дорожчими, але взаємодія сервісів з ними є швидшою. Також, можна використовувати брокери повідомлень для обміну файлами, що доволі популярно у використанні підходу побудови мікросервісної архітектури. Це може бути Azure ServiceBus, Amazon AWS SQS у випадку відносно більших повідомлень, або Azure Event Grid, Amazon AWS SNS, Apache Kafka (якщо у власному кластері) якщо повідомлення відносно невеликі.

Хмарні рішення добре працюють у тандемі та мають зручну інтеграцію, а інші технології можна самостійно розгортати та підтримувати у власному кластері чи сервері. Це може бути дешевше при більших навантаженнях. Також використовують гібридні підходи із інтеграцією хмарних рішень у мікросервіси із кластеру.

Технології Peer-to-peer (P2P) не такі розповсюдженні у комерційному використанні, зокрема через неконтрольованість мережі, якщо ноди є у самих користувачів, бо великий ризик пірацтва контенту та більший вектор для кібер атак. Але ці рішення можуть бути вигідними у розрізі IoT (Internet of Things) речей, де якісь файли можна доставляти по частинах до локально згрупованих девайсів для пришвидшення швидкодії.

Технології з використанням Blockchain доволі нові, як наприклад IPFS (InterPlanetary File System), тож часно практичне використання можна знайти наразі у стартапах або окремих користувачів, що пробують нові технології. Хоч таке рішення забезпечує перевірку цілісності контенту, недоліком є відсутність

шифрування файлів, які фактично стають доступні всім під’єднаним нодам, де часто це всі публічні ноди, тому комерційне використання такої технології вимагає ретельного аналізу ризиків та часто юридичних аспектів використання.

Також, як видно з практичних результатів переддипломної практики в компанії з розробки програмного забезпечення, де було впровадженно зберігання та передачі файлів різного розміру та кількості між веб-сервісами, використання хмарних рішень є доволі не дешеве, але ефективним та приносить хороші результати. Часто оптимізація самої структури даних, а також підлаштування певних бізнес процесів можуть в рази покращити пропускну здатність системи та значно знизити операційні витрати. При цьому застосування стратегій кешування, стиснення та інтеграції правильно підібраних хмарних технологій грає важливу роль в збереженні та завантаженні контенту, коли його об’єми ростуть.

ВИСНОВКИ

Підсумовуючи наведені у роботі результати, можна побачити, що існує велика кількість різних технологій для збереження та обміну файлами веб-сервісів. Кожна з них має свої переваги та недоліки, а також відрізняється легкістю налаштування та цінами, адже деякі рішення є платними. Під час реалізації власного проекту слід користуватись зваженим вибором підходу для досягнення необхідного функціоналу при цьому маючи певні обмеження зі сторони бізнесу. Для цього можна використовувати порівняння наведених технологій у цій роботі, щоб підібрати та застосувати кращий спосіб для вирішення певної задачі.

З кожним днем об'єми даних та розміри файлів, якими оперують користувачі дедалі збільшується. А отже, збільшується і навантаження на веб-сервіси, що їх обробляють, що веде до збільшення операційних витрат на підтримку таких рішень, зменшення продуктивності продукту та ускладнення можливості розширення у майбутньому. Як приклад, можна навести нещодавну історію з Amazon Prime Video, де вони використовували свої ж хмарні AWS сервіси неефективно, що при зростанні навантаження змусило їх перебудувати частину системи з обробки відео [45], що дозволило не лише вирішити проблему з навантаженням, значно зменшило вартість використання сервісів, але й спростило архітектуру системи для зменшення витрат на її підтримку. У цій роботі можна скористатись наведеними результатами для вирішення різних задач як і з розгортанням безкоштовних продуктів у власному кластері, так і використовувати різноманітні хмарні технології чи інтегрувати гібридні технології для ефективного досягнення поставленої цілі.

Наведені ціни на комерційні рішення додані також для порівняння, але можуть динамічно змінюватись залежно від ситуації на ринку та бажання вендора. Тому, при проектуванні систем варто скористатись актуальними цінами, що можна обрахувати в онлайн калькуляторах, наведених в джерелах роботи. При цьому варто враховувати можливі інтеграції з існуючими

продуктами чи сервісами, та використовувати їх за можливості.

Результати дослідження зроблені у цій роботі, також було використано на практиці під час розробки комерційного продукту для клієнта, задача та результати якого також наведені у роботі. Тож можна підсумувати, що отримані результати свідчать про ефективність вибору тої чи іншої технології для зберігання та обміну файлами веб-сервісів згідно описаних в роботі характеристик.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Technical Guide to IPFS – the Decentralized Storage of Web3 - URL: https://www.freecodecamp.org/news/technical-guide-to-ipfs-decentralized-storage-of-web3/
2. InterPlanetary File System - URL: https://en.wikipedia.org/wiki/InterPlanetary_File_System
3. BitTorrent - URL:https://en.wikipedia.org/wiki/BitTorrent
4. Comparison of distributed file systems - URL: https://en.wikipedia.org/wiki/Comparison_of_distributed_file_systems
5. Peer-to-peer file sharing - URL: https://en.wikipedia.org/wiki/Peer-to-peer_file_sharing
6. Децентралізована система для зберігання та поширення даних на основі технології S3 - URL: https://ela.kpi.ua/bitstream/123456789/35625/1/Kyrychenko_bakalavr.pdf
7. Cloud Storage Comparative Analysis Amazon Simple Storage vs. Microsoft Azure Blob Storage - URL: http://www.ijmlc.org/vol8/668-LC0040.pdf
8. Huang, Huawei & Lin, Jianru & Zheng, Baichuan & Zheng, Zibin & Bian, Jing. (2020). When Blockchain Meets Distributed File Systems: An Overview, Challenges, and Open Issues. IEEE Access. PP. 10.1109/ACCESS.2020.2979881.
9. Y. Chen, H. Li, K. Li and J. Zhang, "An improved P2P file system scheme based on IPFS and Blockchain," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 2017, pp. 2652-2657, doi: 10.1109/BigData.2017.8258226.
10. Bo Dong, Qinghua Zheng, Feng Tian, Kuo-Ming Chao, Rui Ma, Rachid Anane, An optimized approach for storing and accessing small files on cloud storage, Journal of Network and Computer Applications, Volume 35, Issue 6, 2012, Pages 1847-1862, ISSN 1084-8045, https://doi.org/10.1016/j.jnca.2012.07.009.
11. Kafka-Apache Kafka - URL: http://archive.keyllo.com/L-

%E7%BC%96%E7%A8%8B/Kafka-Apache%20Kafka.pdf

12. Sipat Triukose, Zhihua Wen, and Michael Rabinovich. 2011. Measuring a commercial content delivery network. In Proceedings of the 20th international conference on World wide web (WWW '11). Association for Computing Machinery, New York, NY, USA, 467–476. https://doi.org/10.1145/1963405.1963472
13. Elif Dede, Madhusudhan Govindaraju, Daniel Gunter, Richard Shane Canon, and Lavanya Ramakrishnan. 2013. Performance evaluation of a MongoDB and hadoop platform for scientific data analysis. In Proceedings of the 4th ACM workshop on Scientific cloud computing (Science Cloud '13). Association for Computing Machinery, New York, NY, USA, 13–20. https://doi.org/10.1145/2465848.2465849
14. Azure Cosmos DB – Unified AI Database - URL:https://learn.microsoft.com/en-us/azure/cosmos-db/introduction
15. Document-oriented database - URL:https://en.wikipedia.org/wiki/Document-oriented_database
16. Analyzing and Improving BitTorrent Performance - URL: https://www.researchgate.net/profile/Ashwin-Bharambe/publication/245396846_Analyzing_and_Improving_BitTorrent_Performance/links/54ef331d0cf2495330e1b8ac/Analyzing-and-Improving-BitTorrent-Performance.pdf
17. Pouwelse, J., Garbacki, P., Epema, D., Sips, H. (2005). The Bittorrent P2P File-Sharing System: Measurements and Analysis. In: Castro, M., van Renesse, R. (eds) Peer-to-Peer Systems IV. IPTPS 2005. Lecture Notes in Computer Science, vol 3640. Springer, Berlin, Heidelberg. https://doi.org/10.1007/11558989_19
18. SQL Server Replication - URL: https://learn.microsoft.com/en-us/sql/relational-databases/replication/sql-server-replication?view=sql-server-ver16

19. What is change data capture - URL: https://learn.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server?view=sql-server-ver16
20. What Is Data Replication? (Replication Types and Schemes Explained) - URL: https://phoenixnap.com/kb/data-replication
21. Azure Cosmos DB - URL: https://learn.microsoft.com/en-us/azure/cosmos-db/
22. Azure Blob Storage - URL: https://learn.microsoft.com/en-us/azure/storage/blobs/
23. Uploading files to Azure Blob Storage - URL: https://bluexp.netapp.com/blog/azure-cvo-blg-how-to-upload-files-to-azure-blob-storage
24. Optimizing throughput cost in Azure Cosmos DB - URL: https://learn.microsoft.com/en-us/azure/cosmos-db/optimize-cost-throughput
25. Advanced Merge Replication - Conflict Detection and Resolution - URL: https://learn.microsoft.com/en-us/sql/relational-databases/replication/merge/advanced-merge-replication-conflict-detection-and-resolution?view=sql-server-ver16
26. Database Replication - URL: https://atlan.com/what-is/database-replication/
27. Resolving Replication Conflicts - URL: https://docs.oracle.com/cd/E11882_01/timesten.112/e21635/conflict.htm
28. Reducing JSON Data Size - URL: https://www.baeldung.com/json-reduce-data-size
29. JSON compression: alternative binary formats and compression methods - URL: https://www.lucidchart.com/techblog/2019/12/06/json-compression-alternative-binary-formats-and-compression-methods/
30. Google One plans - URL:https://one.google.com/about/plans
31. Microsoft OneDrive plans - URL:https://www.microsoft.com/en-us/microsoft-365/onedrive/compare-onedrive-plans
32. Dropbox plans - URL:https://www.dropbox.com/plans?billing=monthly
33. Microsoft Sharepoint plans - URL:https://www.microsoft.com/en-us/microsoft-365/sharepoint/compare-sharepoint-plans

34. Azure Pricing calculator - URL: https://azure.microsoft.com/en-us/pricing/calculator/
35. Azure Blob Storage documentation - URL:https://learn.microsoft.com/en-us/azure/storage/blobs/
36. Azure Service Bus Messaging documentation - URL:https://learn.microsoft.com/en-us/azure/service-bus-messaging
37. Azure Cosmos DB documentation - URL:https://learn.microsoft.com/en-us/azure/cosmos-db
38. AWS Pricing Calculator - URL:https://calculator.aws/
39. What is Amazon S3 - URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
40. Amazon SQS Quotas - URL: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/quotas-messages.html
41. Amazon DynamoDB Quotas - URL: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ServiceQuotas.html
42. IPFS - URL: https://ipfs.tech/
43. MongoDB C# Driver - URL: https://www.mongodb.com/docs/drivers/csharp/current/
44. Terraform - URL:https://developer.hashicorp.com/terraform/docs
45. Exploring Amazon Prime Video's Architecture: Migrating from Microservices to Monolith for Audio/Video Monitoring Service - URL: https://medium.com/@anshita.bhasin/exploring-amazon-prime-videos-architecture-migrating-from-microservices-to-monolith-for-aacbf9fabc73
46. https://www.youtube.com/watch?v=0K1pITq4mSk
47. Overview of WebSocket support in Application Gateway - URL: https://learn.microsoft.com/uk-ua/azure/application-gateway/application-gateway-websocket
48. What is WebDAV?- URL: https://www.quora.com/What-is-WebDAV