Методи Зберігання Даних в Інформаційно-аналітичній Системі Об'єднаного Управління Військами (ІАЗ ОУВ): Архітектура, Технології та Практичне Застосування

Частина 1: Архітектурні Основи Зберігання Даних в ІАЗ ОУВ

1.1. Характеристика Даних в Сучасних Інформаційно-аналітичних Системах

Сучасні інформаційно-аналітичні системи, особливо у військовій сфері, функціонують в умовах інформаційного вибуху. Ефективність прийняття рішень безпосередньо залежить від здатності системи збирати, обробляти, зберігати та аналізувати величезні масиви різнорідної інформації. Для системного розуміння цих викликів використовується концепція "трьох V" великих даних, адаптована до специфіки ІАЗ ОУВ:

- **Volume (Обсяг):** Обсяг даних, що генеруються та обробляються, є колосальним. Це не лише традиційні звіти та таблиці, а й потоки даних із супутників, відео з безпілотних літальних апаратів (БПЛА), логи мережевого обладнання та систем озброєння, дані радіоелектронної розвідки. Ці обсяги вимірюються терабайтами та петабайтами, що висуває жорсткі вимоги до масштабованості та економічної ефективності систем зберігання.
- **Velocity (Швидкість):** Швидкість надходження та обробки даних є критичним фактором. Оперативна обстановка на полі бою змінюється щохвилини. Система

повинна обробляти потокові дані телеметрії з рухомих об'єктів, миттєво реагувати на оновлення розвідданих та забезпечувати аналітику в режимі, максимально наближеному до реального часу. Затримка в обробці може коштувати життів та призвести до втрати тактичної переваги.

- **Variety (Різноманітність):** Це, можливо, найскладніший виклик для архітектури ІАЗ ОУВ. Дані надходять у найрізноманітніших форматах:
  - **Структуровані дані:** Це інформація з чітко визначеною схемою, яку легко представити у вигляді таблиць з рядками та колонками. Приклади: кадрові записи особового складу, таблиці логістичного обліку, переліки озброєння та техніки, нормативи забезпечення.
  - **Напівструктуровані дані:** Дані, що не відповідають жорсткій табличній моделі, але містять теги або інші маркери для відокремлення семантичних елементів. Приклади: розвідувальні зведення у форматах JSON або XML, системні логи з метаданими, дані з датчиків.
  - **Неструктуровані дані:** Найбільший за обсягом клас даних, що не має попередньо визначеної структури. Приклади: текстові доповіді та накази, аудіозаписи перехоплень радіопереговорів, відео з розвідувальних дронів, супутникові знімки, фотографії з поля бою.

Різноманітність даних в ІАЗ ОУВ є не просто технічною проблемою, а фундаментальною операційною вимогою. Нездатність системи ефективно обробляти та, що важливіше, інтегрувати *всі* типи даних призводить до утворення "сліпих зон" в аналізі обстановки. Це безпосередньо уповільнює цикл прийняття рішень, відомий як OODA (Observe, Orient, Decide, Act – Спостерігай, Орієнтуйся, Вирішуй, Дій). Традиційні системи чудово справляються з аналізом структурованих даних, наприклад, можуть показати точний стан пального на складі N. Однак сучасна війна генерує величезну кількість неструктурованих даних, як-от відео з БПЛА, що фіксує переміщення ворожої колони. Якщо система не здатна автоматично поєднати ці два потоки – наприклад, розпізнати тип та кількість техніки на відео, геолокувати її та зіставити з розвідданими про маршрути постачання, пов'язаними зі складом N, – аналітик змушений виконувати цю кореляцію вручну. Це створює критичну затримку та підвищує ризик людської помилки. Отже, архітектура зберігання даних повинна бути спроектована з самого початку з урахуванням *інтеграції* різнорідних даних, а не їх ізольованого зберігання в окремих "силосах".

1.2. Фундаментальні Моделі Даних

Для роботи з різноманітними даними ІАЗ ОУВ необхідно розуміти фундаментальні моделі, на яких базуються сучасні системи управління базами даних (СУБД). Кожна модель має

свої сильні та слабкі сторони, що робить її оптимальною для певного класу завдань.

- **Реляційна модель (SQL):** Базується на математичній теорії множин та реляційній алгебрі. Дані організовані в таблиці (відношення), що складаються з рядків (кортежів) та колонок (атрибутів). Зв'язки між таблицями встановлюються за допомогою ключів. Ця модель є ідеальною для структурованих даних, де цілісність та узгодженість є найвищим пріоритетом.
- **Документна модель (NoSQL):** Дані зберігаються у вигляді гнучких, ієрархічних документів, зазвичай у форматі JSON або його бінарному представленні BSON. Кожен документ є самодостатньою одиницею, що може містити складні вкладені структури та масиви. Ця модель ідеально підходить для напівструктурованих даних, де схема може змінюватися з часом.
- **Модель Ключ-Значення (NoSQL):** Найпростіша модель, що являє собою асоціативний масив або словник. Кожному унікальному ключу відповідає певне значення, яке може бути простим (число, рядок) або складним (об'єкт, список). Головна перевага – надзвичайно висока швидкість операцій читання та запису.
- **Колонкова модель (NoSQL):** На відміну від реляційних БД, де дані зберігаються по рядках, у колонкових сховищах дані однієї колонки зберігаються разом на диску. Такий підхід значно прискорює аналітичні запити, які зазвичай оперують підмножиною колонок з великої кількості рядків.
- **Графова модель (NoSQL):** Дані представлені у вигляді графів, що складаються з вузлів (вершин) та зв'язків (ребер). Вузли представляють сутності (наприклад, особи, організації, об'єкти), а ребра – зв'язки між ними. Ця модель є незамінною для аналізу складних мереж, таких як структура командування противника, логістичні ланцюжки або соціальні мережі.

1.3. Теорема CAP: Тріада Вибору в Розподілених Системах

При проектуванні будь-якої сучасної розподіленої системи, якою є ІАЗ ОУВ, архітектори неминуче стикаються з фундаментальним компромісом, описаним теоремою CAP, сформульованою Еріком Брюером. Теорема стверджує, що в будь-якій розподіленій системі зберігання даних можна одночасно забезпечити не більше двох із трьох наступних властивостей:

- **Consistency (Узгодженість):** Гарантія того, що будь-яке читання повертає результат останнього успішного запису. Всі клієнти, що звертаються до системи, бачать однакові дані в один і той самий момент часу.
- **Availability (Доступність):** Гарантія того, що кожен запит до системи отримує відповідь (не обов'язково з найсвіжішими даними). Система завжди залишається працездатною для читання та запису.
- **Partition Tolerance (Стійкість до розділення):** Здатність системи продовжувати

функціонувати навіть у випадку втрати мережевого зв'язку між її вузлами (розділення мережі).

У контексті ІАЗ ОУВ, де вузли системи можуть бути географічно розподілені між стаціонарними центрами обробки даних (ЦОД), мобільними командними пунктами та тактичними підрозділами, втрата зв'язку є не винятком, а очікуваним сценарієм роботи. Тому стійкість до розділення (P) є **обов'язковою, не обговорюваною вимогою**. Таким чином, реальний вибір для архітектора зводиться до компромісу між узгодженістю (C) та доступністю (A).

- **Системи типу CP (Consistency / Partition Tolerance):** Ці системи обирають узгодженість на шкоду доступності. У разі розділення мережі, щоб уникнути ризику запису неузгоджених даних, частина системи (зазвичай, менша частина розділу) може стати недоступною для операцій запису. Вона чекатиме відновлення зв'язку для синхронізації, перш ніж дозволити нові зміни. **Приклад в ІАЗ ОУВ:** система управління кадровими записами або фінансового обліку. Неприпустимо, щоб внаслідок мережевого збою було створено два різних накази про призначення на одну й ту саму посаду.
- **Системи типу AP (Availability / Partition Tolerance):** Ці системи обирають доступність на шкоду миттєвій узгодженості. Навіть під час розділення мережі всі вузли системи залишаються доступними для читання та запису. Система завжди відповідатиме на запити, але є ризик, що дані, отримані з різних частин розділеної мережі, можуть бути тимчасово розсинхронізованими. **Приклад в ІАЗ ОУВ:** система моніторингу оперативної обстановки. Для оператора на передовій краще мати доступ до карти з даними про положення ворожих сил, які є застарілими на кілька хвилин, ніж не мати доступу до карти взагалі.

Цей вибір виходить далеко за межі суто технічного рішення; він визначає операційну доктрину використання системи. Технічний спеціаліст розглядає CAP як інженерний компроміс. Для командира ж "доступність" означає здатність отримати критично важливу інформацію тут і зараз, а "узгодженість" – гарантію її абсолютної точності. Розглянемо систему управління логістикою. Якщо вона спроектована як AP, два підрозділи в умовах нестабільного зв'язку можуть одночасно "списати" один і той самий дефіцитний ресурс (наприклад, комплект боєприпасів) зі складу, що призведе до операційного хаосу та зриву виконання завдання. Для таких даних потрібна сильна узгодженість, тобто система типу CP. Навпаки, для системи відображення цілей, якщо вона буде типу CP, оператор БПЛА при втраті зв'язку з центральним сервером може взагалі перестати бачити цілі на своїй карті, оскільки його вузол не зможе гарантувати узгодженість. Тут життєво необхідна доступність (AP). Таким чином, необхідно провести класифікацію всіх даних та підсистем в ІАЗ ОУВ за критерієм "ціна помилки через неузгодженість vs ціна затримки через недоступність". Проектування підсистем має відбуватися відповідно до цієї класифікації, а операційні наслідки кожного архітектурного вибору повинні бути чітко роз'яснені та узгоджені з військовим командуванням.

Частина 2: Реляційний Підхід (SQL): Надійність та Структурованість

2.1. Проектування Схеми для ІАЗ ОУВ: Принципи Нормалізації

Реляційні бази даних, що використовують мову SQL (Structured Query Language), є основою для зберігання структурованих даних завдяки своїй надійності, передбачуваності та строгим гарантіям цілісності. Фундаментом правильного проектування реляційної схеми є процес нормалізації. Його мета – усунення надлишковості даних та аномалій (помилок), що можуть виникнути при їх оновленні, вставці або видаленні. Розглянемо основні нормальні форми (NF) на практичному прикладі.

Уявімо погано спроектовану, ненормалізовану таблицю unit_equipment, що зберігає інформацію про техніку в підрозділах:

unit_id	unit_name	commander_name	equipment_id	equipment_type	equipment_model	status
101	1-й мех. батальйон	Майор Петренко	Т-64-001	Танк	Т-64БВ	Боєготовий
101	1-й мех. батальйон	Майор Петренко	Т-64-002	Танк	Т-64БВ	Боєготовий
101	1-й мех. батальйон	Майор Петренко	БМР-2-005	БМП	БМП-2	В ремонті

Така структура має низку серйозних недоліків:

- **Надлишковість:** Інформація про назву підрозділу та ім'я командира дублюється для кожної одиниці техніки.
- **Аномалія оновлення:** Якщо зміниться командир 1-го батальйону, доведеться оновити *всі* записи, що стосуються цього підрозділу. Якщо хоча б один запис буде пропущено, виникне неузгодженість даних.
- **Аномалія вставки:** Неможливо додати новий підрозділ, якщо за ним ще не закріплено жодної одиниці техніки.
- **Аномалія видалення:** Якщо з підрозділу буде видалено останню одиницю техніки, інформація про сам підрозділ (його назва, командир) також буде втрачена.

Процес нормалізації до третьої нормальної форми (3NF) дозволяє усунути ці проблеми шляхом розділення великої таблиці на менші, логічно пов'язані сутності:

1. **Перша нормальна форма (1NF):** Всі значення в комірках є атомарними (неподільними), і немає груп полів, що повторюються. Наша початкова таблиця вже задовольняє цю умову.
2. **Друга нормальна форма (2NF):** Таблиця знаходиться в 1NF, і всі неключові атрибути повністю залежать від первинного ключа.
3. **Третя нормальна форма (3NF):** Таблиця знаходиться в 2NF, і немає транзитивних залежностей (коли неключовий атрибут залежить від іншого неключового атрибута).

Застосувавши ці принципи, ми отримуємо наступну нормалізовану структуру:

- **personnel** (Особовий склад):
  - personnel_id (PK), name, rank
- **units** (Підрозділи):
  - unit_id (PK), unit_name, commander_id (FK до personnel)
- **equipment_types** (Типи техніки):
  - type_id (PK), type_name
- **equipment** (Одиниці техніки):
  - equipment_id (PK), model_name, type_id (FK до equipment_types)
- **unit_equipment_assignment** (Таблиця зв'язку: призначення техніки підрозділам):
  - assignment_id (PK), unit_id (FK до units), equipment_id (FK до equipment), status

Приклад коду на PostgreSQL для створення нормалізованої схеми:

SQL

```sql
-- Таблиця для особового складу
CREATE TABLE personnel (
    personnel_id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
```

```
    rank VARCHAR(100) NOT NULL
);

-- Таблиця для підрозділів
CREATE TABLE units (
    unit_id SERIAL PRIMARY KEY,
    unit_name VARCHAR(255) NOT NULL UNIQUE,
    commander_id INT,
    FOREIGN KEY (commander_id) REFERENCES personnel(personnel_id)
);

-- Таблиця для типів техніки
CREATE TABLE equipment_types (
    type_id SERIAL PRIMARY KEY,
    type_name VARCHAR(100) NOT NULL UNIQUE
);

-- Таблиця для конкретних одиниць техніки
CREATE TABLE equipment (
    equipment_id VARCHAR(50) PRIMARY KEY, -- Використовуємо VARCHAR для бортових номерів
    model_name VARCHAR(100) NOT NULL,
    type_id INT NOT NULL,
    FOREIGN KEY (type_id) REFERENCES equipment_types(type_id)
);

-- Таблиця зв'язку для призначення техніки підрозділам
CREATE TABLE unit_equipment_assignment (
    assignment_id SERIAL PRIMARY KEY,
    unit_id INT NOT NULL,
    equipment_id VARCHAR(50) NOT NULL,
    status VARCHAR(50) NOT NULL CHECK (status IN ('active', 'in_repair', 'transferring',
'decommissioned')),
    assignment_date DATE DEFAULT CURRENT_DATE,
    FOREIGN KEY (unit_id) REFERENCES units(unit_id),
    FOREIGN KEY (equipment_id) REFERENCES equipment(equipment_id),
    UNIQUE (equipment_id) -- Гарантує, що одна одиниця техніки не може бути призначена двом
підрозділам одночасно
);
```

Ця структура вирішує всі проблеми початкової схеми: дані не дублюються, зміна командира вимагає оновлення лише одного запису в таблиці units, а підрозділи та техніка існують незалежно одне від одного.

2.2. Транзакції та Гарантії ACID

Основою надійності реляційних баз даних є підтримка транзакцій з гарантіями ACID. Транзакція – це послідовність операцій, що виконується як єдине ціле. Акронім ACID розшифровується наступним чином:

- **Atomicity (Атомарність):** Гарантує, що транзакція буде виконана повністю або не виконана взагалі. Якщо будь-яка операція всередині транзакції зазнає невдачі, всі попередні операції цієї транзакції будуть скасовані (відкат). Не існує проміжних, частково виконаних станів.
- **Consistency (Узгодженість):** Гарантує, що кожна успішна транзакція переводить базу даних з одного валідного (узгодженого) стану в інший. Всі правила цілісності, визначені в схемі (первинні та зовнішні ключі, обмеження CHECK), будуть дотримані.
- **Isolation (Ізольованість):** Гарантує, що паралельно виконувані транзакції не впливають одна на одну. Кожна транзакція виконується так, ніби вона єдина в системі в даний момент.
- **Durability (Довговічність):** Гарантує, що результати успішно завершеної (зафіксованої) транзакції будуть збережені в базі даних назавжди, навіть у випадку збою живлення, відмови обладнання або перезавантаження системи.

Практичний приклад: Розглянемо операцію переведення одиниці техніки з одного підрозділу в інший. Ця операція складається з двох кроків: оновлення статусу в старому призначенні та створення нового запису про призначення. Без транзакції, якщо система вийде з ладу між цими двома кроками, техніка може "зависнути" в невизначеному стані. Використання транзакції гарантує атомарність цієї операції.

Код транзакції на PostgreSQL:

SQL

```sql
-- Початок транзакції
BEGIN;

-- Припустимо, ми переводимо техніку з ID 'T-64-002' з підрозділу 101 до підрозділу 105.
-- Спочатку, ми можемо видалити старе призначення або змінити його статус.
-- Обережний підхід - змінити статус, щоб зберегти історію.
UPDATE unit_equipment_assignment
SET status = 'decommissioned', assignment_date = '2024-05-20' -- Вказуємо дату зняття з обліку
```

```
WHERE equipment_id = 'T-64-002' AND unit_id = 101;

-- Потім додаємо нове призначення для нового підрозділу.
INSERT INTO unit_equipment_assignment (unit_id, equipment_id, status, assignment_date)
VALUES (105, 'T-64-002', 'active', '2024-05-21');

-- Якщо обидві операції пройшли успішно, фіксуємо зміни.
COMMIT;

-- У випадку помилки, можна виконати ROLLBACK, щоб скасувати всі зміни з моменту BEGIN.
-- ROLLBACK;
```

Завдяки обгортці BEGIN...COMMIT, система гарантує, що або обидві операції (UPDATE та INSERT) будуть успішно виконані та збережені, або, у випадку будь-якого збою, база даних повернеться до стану, який був до початку транзакції.

2.3. Практикум з SQL: Складні Запити для Аналітики

Сила SQL полягає не лише в надійному зберіганні даних, а й у потужних можливостях для їх вибірки та аналізу. Розглянемо кілька прикладів складних запитів до нашої нормалізованої схеми.

1. Звіт про боєготовність підрозділу (використання JOIN):
Отримати повний список особового складу та техніки для конкретного підрозділу з їх поточними статусами.

SQL

```
SELECT
    u.unit_name,
    p.name AS commander_name,
    p.rank AS commander_rank,
    e.equipment_id,
    et.type_name,
    e.model_name,
    uea.status AS equipment_status
FROM units u
JOIN personnel p ON u.commander_id = p.personnel_id
```

```
LEFT JOIN unit_equipment_assignment uea ON u.unit_id = uea.unit_id
LEFT JOIN equipment e ON uea.equipment_id = e.equipment_id
LEFT JOIN equipment_types et ON e.type_id = et.type_id
WHERE u.unit_name = '1-й мех. батальйон';
```

Цей запит об'єднує п'ять таблиць, щоб надати комплексний звіт. Використання LEFT JOIN гарантує, що підрозділ буде показаний у звіті, навіть якщо за ним ще не закріплено жодної техніки.

2. Агрегація даних (використання GROUP BY та HAVING):
Порахувати кількість боєготових одиниць техніки кожного типу по всіх підрозділах та вивести лише ті типи, яких більше 5.

SQL

```
SELECT
    et.type_name,
    COUNT(uea.assignment_id) AS ready_count
FROM equipment_types et
JOIN equipment e ON et.type_id = e.type_id
JOIN unit_equipment_assignment uea ON e.equipment_id = uea.equipment_id
WHERE uea.status = 'active'
GROUP BY et.type_name
HAVING COUNT(uea.assignment_id) > 5
ORDER BY ready_count DESC;
```

3. Ранжування за допомогою віконних функцій (Window Functions):
Проранжувати підрозділи за кількістю закріпленої за ними техніки.

SQL

```
SELECT
    u.unit_name,
    COUNT(uea.assignment_id) AS equipment_count,
    RANK() OVER (ORDER BY COUNT(uea.assignment_id) DESC) AS equipment_rank
FROM units u
JOIN unit_equipment_assignment uea ON u.unit_id = uea.unit_id
GROUP BY u.unit_name;
```

4. Складний аналіз за допомогою Common Table Expressions (CTE):
Знайти підрозділи, у яких частка техніки в ремонті перевищує 20% від загальної кількості.

SQL

```sql
WITH UnitEquipmentStats AS (
    SELECT
        u.unit_id,
        u.unit_name,
        COUNT(uea.assignment_id) AS total_equipment,
        SUM(CASE WHEN uea.status = 'in_repair' THEN 1 ELSE 0 END) AS repair_equipment
    FROM units u
    JOIN unit_equipment_assignment uea ON u.unit_id = uea.unit_id
    GROUP BY u.unit_id, u.unit_name
)
SELECT
    unit_name,
    total_equipment,
    repair_equipment,
    (repair_equipment::DECIMAL / total_equipment) * 100 AS repair_percentage
FROM UnitEquipmentStats
WHERE (repair_equipment::DECIMAL / total_equipment) > 0.20;
```

2.4. Оптимізація Продуктивності

Зі зростанням обсягів даних навіть добре спроектована реляційна база може почати працювати повільно. Ключовими інструментами для оптимізації продуктивності є індексація та аналіз планів запитів.

- **Індексація:** Індекс – це спеціальна структура даних, що дозволяє СУБД швидко знаходити рядки в таблиці, не скануючи її повністю. Це схоже на алфавітний покажчик у книзі. Індекси слід створювати на:
    - **Зовнішніх ключах (Foreign Keys):** unit_id, equipment_id в таблиці unit_equipment_assignment. Це значно прискорює операції JOIN.
    - **Полях, що часто використовуються в умовах WHERE:** Наприклад, status в unit_equipment_assignment або unit_name в units.

    SQL

    ```sql
    -- Створення індексів для прискорення JOIN-операцій та пошуку
    ```

```
CREATE INDEX idx_uea_unit_id ON unit_equipment_assignment(unit_id);
CREATE INDEX idx_uea_equipment_id ON unit_equipment_assignment(equipment_id);
CREATE INDEX idx_uea_status ON unit_equipment_assignment(status);
```

- **Аналіз плану запиту (EXPLAIN ANALYZE):** Це потужний інструмент PostgreSQL, що показує, як саме СУБД планує виконувати запит. Він відображає послідовність операцій (сканування таблиць, з'єднання, сортування), їхню вартість (оціночні витрати ресурсів) та реальний час виконання. Аналізуючи вивід EXPLAIN ANALYZE для складного запиту до та після створення індексу, можна наочно побачити ефект оптимізації. Наприклад, замість повільного Seq Scan (послідовного сканування всієї таблиці) план запиту почне використовувати швидкий Index Scan.

Частина 3: Гнучкі Підходи (NoSQL): Масштабованість та Швидкість

3.1. Огляд Екосистеми NoSQL та Практичні Сценарії

Якщо SQL-бази даних є ідеальним вибором для структурованих даних з жорсткими вимогами до узгодженості, то NoSQL ("Not Only SQL") бази даних пропонують альтернативні моделі для вирішення проблем, пов'язаних з великими даними, високою швидкістю та гнучкістю схеми. Розглянемо ключові типи NoSQL баз даних та їх застосування в ІАЗ ОУВ.

3.1.1. Документо-орієнтовані БД (MongoDB)

Концепція: Дані зберігаються у вигляді документів, найчастіше у форматі BSON (бінарний JSON), що дозволяє зберігати складні ієрархічні структури даних в одній сутності. На відміну від SQL, тут немає жорсткої, попередньо визначеної схеми. Поля можуть додаватися до документів на льоту, а структура документів в одній колекції може відрізнятися.

Сценарій для ІАЗ ОУВ: Зберігання розвідувальних зведень. Структура такого зведення є вкрай динамічною. Одне зведення може містити текстовий опис, координати цілі у

форматі GeoJSON, масив ідентифікованих об'єктів (кожен зі своїм набором атрибутів), посилання на фото- та відеоматеріали у файловому сховищі, рівень достовірності інформації та набір тегів для швидкого пошуку. Спроба змоделювати таку структуру в реляційній базі призвела б до створення десятків таблиць зі складною логікою JOIN, тоді як в MongoDB все це може зберігатися як один цілісний документ.

Приклад коду (MongoDB Query Language):

JavaScript

```
// Створення нового розвідувального зведення в колекції 'intelligence_reports'
db.intelligence_reports.insertOne({
  report_id: "REP-2024-05-21-001",
  timestamp_received: new Date("2024-05-21T14:30:00Z"),
  source: {
    type: "UAV",
    id: "Leleka-100-7B"
  },
  location: {
    type: "Point",
    coordinates: [35.1234, 48.5678] // [довгота, широта]
  },
  summary_text: "Виявлено скупчення бронетехніки та артилерії в лісосмузі. Орієнтовна кількість - до 15 одиниць.",
  identified_objects:,
  raw_data_links: [
    "/storage/video/20240521_1425_uav7b.mp4",
    "/storage/images/20240521_1428_uav7b_thermal.jpg"
  ],
  confidence_level: 0.85,
  tags: ["armor", "convoy", "artillery", "north_sector"]
});

// Пошук всіх зведень з тегом "artillery" в радіусі 10 км від заданої точки
// з рівнем достовірності вище 0.8
db.intelligence_reports.find({
  tags: "artillery",
  confidence_level: { $gt: 0.8 },
  location: {
    $geoWithin: {
```

```
    // Радіус в метрах
    $centerSphere: [35.1, 48.5], 10000 / 6378100 ]
   }
  }
}).sort({ timestamp_received: -1 }); // Сортування за часом, найновіші спочатку
```

3.1.2. Сховища Ключ-Значення (Redis)

Концепція: Це надзвичайно швидкі, переважно in-memory (що зберігають дані в оперативній пам'яті) бази даних. Вони надають просту модель доступу: є унікальний ключ і відповідне йому значення. Redis розширює цю модель, підтримуючи різні типи даних для значень: рядки, списки, хеші, множини, що робить його потужним та універсальним інструментом.

Сценарії для ІАЗ ОУВ:

1. **Кешування:** Зберігання результатів "важких" та часто виконуваних SQL-запитів (наприклад, звіт про боєготовність) або часто запитуваних даних (профілі користувачів, довідники). Це дозволяє значно знизити навантаження на основну реляційну базу даних.
2. **Управління сесіями:** Зберігання даних сесій операторів ІАЗ ОУВ, що забезпечує швидкий доступ та легке горизонтальне масштабування веб-серверів.
3. **Черги повідомлень та завдань:** Організація черг для асинхронної обробки завдань, наприклад, черга відеофайлів для розпізнавання об'єктів, або черга повідомлень для розсилки сповіщень.
4. **Дані реального часу:** Зберігання та швидке оновлення даних, що змінюються з високою частотою, наприклад, останні відомі координати дружніх підрозділів (Blue Force Tracking).

Приклад коду (команди redis-cli):

```
Bash
```

```
# Зберегти останню позицію підрозділу "Сокіл-1" (координати та час)
# Використовуємо геопросторові дані для координат
GEOADD friendly_units_live 35.1234 48.5678 "unit:sokil-1"

# Зберігаємо додаткову інформацію (швидкість, напрямок) у хеші
```

```
HSET unit:sokil-1:data timestamp 1684679400 speed_kmh 25 heading_deg 90

# Встановити час життя для запису (наприклад, 10 хвилин), після чого він автоматично видалиться,
# якщо не буде оновлений. Це допомагає позбутися застарілих даних.
EXPIRE unit:sokil-1:data 600

# Отримати всі дружні підрозділи в радіусі 5 км від точки [35.1, 48.5]
# з їхніми координатами та відстанню до центру
GEORADIUS friendly_units_live 35.1 48.5 5 km WITHCOORD WITHDIST

# Отримати дані про підрозділ "Сокіл-1"
HGETALL unit:sokil-1:data
```

3.1.3. Колонкові БД (Cassandra/ScyllaDB)

Концепція: Ці бази даних оптимізовані для запису величезних обсягів даних та їх швидкого читання. Дані фізично зберігаються по колонках, а не по рядках. Вони забезпечують високу доступність та лінійну масштабованість (додавання нових серверів до кластера пропорційно збільшує його продуктивність). Важливою особливістю є підхід до моделювання даних "під запит": схему таблиці проектують так, щоб вона максимально ефективно відповідала на конкретні, заздалегідь відомі запити.

Сценарій для ІАЗ ОУВ: Зберігання часових рядів (time-series data). Це будь-які дані, що прив'язані до часової мітки: телеметрія з БПЛА (висота, швидкість, залишок пального, координати кожну секунду), логи з серверів та мережевого обладнання, дані з акустичних та сейсмічних сенсорів на полі бою. Обсяги таких даних можуть бути астрономічними, а типовий запит – "показати всі показники для сенсора X за останню годину".

Приклад моделювання та запиту (CQL - Cassandra Query Language):

SQL

```
-- Створення таблиці для зберігання телеметрії з БПЛА.
-- Ключ розділення (PARTITION KEY) - uav_id. Це означає, що всі дані для одного БПЛА
-- будуть зберігатися на одному вузлі (або його репліках), що забезпечує швидкий доступ.
-- Ключ кластеризації (CLUSTERING KEY) - capture_time. Дані всередині розділу
```

```
-- будуть фізично відсортовані на диску за цим ключем у спадаючому порядку.
CREATE TABLE uav_telemetry (
    uav_id uuid,
    capture_time timestamp,
    location frozen<point>, -- Використовуємо кастомний тип для гео-координат
    altitude_m float,
    speed_kmh float,
    fuel_level_percent float,
    PRIMARY KEY ((uav_id), capture_time)
) WITH CLUSTERING ORDER BY (capture_time DESC);

-- Вставка даних
INSERT INTO uav_telemetry (uav_id, capture_time, location, altitude_m, speed_kmh,
fuel_level_percent)
VALUES (uuid(), '2024-05-21T15:00:01Z', {lat: 48.567, lon: 35.123}, 1200, 150, 75.5);

-- Надзвичайно ефективний запит для отримання телеметрії для конкретного БПЛА за останню
годину.
-- Cassandra швидко знайде потрібний розділ за uav_id і прочитає послідовний блок даних на
диску,
-- оскільки вони вже відсортовані за часом.
SELECT altitude_m, speed_kmh, fuel_level_percent
FROM uav_telemetry
WHERE uav_id = 123e4567-e89b-12d3-a456-426614174000
  AND capture_time >= '2024-05-21T14:00:00Z'
  AND capture_time <= '2024-05-21T15:00:00Z';
```

3.2. Модель Узгодженості BASE

На противагу строгим гарантіям ACID в SQL-світі, більшість розподілених NoSQL систем побудовані на принципах моделі узгодженості BASE. Це інженерний компроміс, що надає перевагу доступності та масштабованості над миттєвою узгодженістю. Акронім BASE розшифровується так:

- **Basically Available (Базова доступність):** Система гарантує доступність для читання та запису, як це вимагає сторона 'A' теореми CAP. Запити до системи завжди отримують відповідь.
- **Soft state (Нестійкий стан):** Стан системи може змінюватися з часом, навіть без надходження нових даних від користувача. Це відбувається через фонові процеси синхронізації, які поширюють оновлення між вузлами.
- **Eventually consistent (Узгодженість в кінцевому рахунку):** Це ключова концепція.

Вона гарантує, що якщо припинити записувати нові дані в систему, то через деякий проміжок часу всі копії даних на всіх вузлах кластера стануть однаковими (узгодженими). "Деякий проміжок часу" може тривати від мілісекунд до хвилин, залежно від налаштувань системи та стану мережі.

Важливо розуміти, що "узгодженість в кінцевому рахунку" – це не помилка або недолік, а свідома архітектурна особливість. Це компроміс, який дозволяє системам залишатися повністю функціональними (доступними для запису та читання) навіть під час серйозних мережевих збоїв, наприклад, при втраті зв'язку між двома дата-центрами. Ключове завдання архітектора – чітко визначити, для яких типів даних та операційних сценаріїв такий компроміс є прийнятним.

Уявімо ситуацію: дані про оперативну обстановку (позиції ворожих сил) реплікуються між головним ЦОД та мобільним командним пунктом. Раптово зв'язок між ними зникає (network partition). Оператор в ЦОД-1 ідентифікує та додає на карту нову ціль. Одночасно оператор у мобільному КП, працюючи автономно, додає іншу ціль, виявлену місцевою розвідкою. В цей момент дані в двох локаціях є неузгодженими. Проте обидва оператори можуть продовжувати працювати, оновлюючи свою локальну картину бою. Коли зв'язок відновлюється, система автоматично запускає процес синхронізації (реплікації). Вона обмінюється змінами, що відбулися за час розриву, і за допомогою спеціальних механізмів вирішення конфліктів (наприклад, "останній запис перемагає" – Last-Write-Wins, або більш складних векторних годинників) приводить дані на обох вузлах до єдиного, узгодженого стану. Для даних оперативної обстановки така короткочасна неузгодженість є абсолютно прийнятною платою за можливість продовжувати роботу в автономному режимі. Однак для фінансових чи кадрових даних такий підхід був би катастрофічним.

Частина 4: Синтез та Прийняття Рішень: Вибір Оптимальної Стратегії для ІАЗ ОУВ

4.1. SQL vs. NoSQL: Детальна Порівняльна Матриця

Вибір між SQL та NoSQL не є вибором між "добрим" та "поганим". Це вибір найбільш відповідного інструменту для конкретного завдання. Щоб систематизувати критерії цього вибору, наведемо детальну порівняльну матрицю з коментарями, специфічними для ІАЗ

ОУВ.

Таблиця 1: Порівняльна характеристика SQL та NoSQL баз даних

Критерій	SQL (на прикладі PostgreSQL)	NoSQL (на прикладах MongoDB, Cassandra, Redis)	Коментар для ІАЗ ОУВ
Модель даних	Реляційна (таблиці, рядки, колонки)	Документи, Ключ-Значення, Колонки, Графи	SQL ідеальний для чітко структурованих сутностей (особовий склад, техніка). NoSQL – для різнорідних розвідданих, телеметрії, даних з сенсорів.
Схема	Жорстка, визначена наперед (Schema-on-Write)	Гнучка, динамічна (Schema-on-Read)	Жорстка схема SQL гарантує якість та цілісність даних для облікових систем. Гнучкість NoSQL є критичною для швидкої інтеграції нових типів розвідданих без необхідності змінювати структуру БД.
Масштабування	Переважно вертикальне (збільшення потужності сервера: CPU, RAM, SSD)	Переважно горизонтальне (додавання нових, відносно недорогих серверів до кластера)	Горизонтальне масштабування NoSQL є єдиним життєздатним варіантом для обробки петабайтів даних

			телеметрії та логів. Вертикальне масштабування має свої фізичні та фінансові межі.
Модель узгодженості	ACID (Атомарність, Узгодженість, Ізольованість, Довговічність)	BASE (Базова доступність, Нестійкий стан, Узгодженість в кінцевому рахунку)	ACID є обов'язковим для систем логістики, кадрів, управління озброєнням, де помилка неприпустима. BASE є прийнятним і навіть необхідним для систем ситуаційної обізнаності, де доступність важливіша за миттєву узгодженість.
Мова запитів	Стандартизований SQL	Різні для кожної БД (MQL для MongoDB, CQL для Cassandra, команди для Redis)	SQL є потужним, декларативним та добре відомим. NoSQL-мови часто є більш процедурними та оптимізованими під конкретну модель даних, що може вимагати додаткового навчання розробників.
Цілісність даних	Забезпечується на рівні БД (зовнішні ключі, обмеження	Зазвичай забезпечується на рівні додатку	SQL перекладає відповідальність за цілісність на

	CHECK, тригери)		СУБД, що робить додатки простішими та надійнішими. В NoSQL розробник сам має реалізовувати логіку для підтримки зв'язків між даними.
Типові сценарії в ІАЗ ОУВ	**Ядро системи:** користувачі, ролі, права доступу. **Облікові підсистеми:** кадри, логістика, фінанси, облік озброєння.	**Оперативні дані:** розвідзведення (MongoDB), телеметрія (Cassandra), кеш та дані реального часу (Redis), аналіз мереж противника (Neo4j).	Чітке розмежування завдань є ключем до ефективної архітектури. Немає універсального рішення.

Ця таблиця слугує швидким довідником, який допомагає зрозуміти фундаментальні відмінності та уникнути поширеної помилки – намагання використати один тип бази даних для вирішення всіх завдань. Вона структурує знання за ключовими критеріями, що є важливими для архітектора, та безпосередньо пов'язує абстрактні властивості БД з конкретними військовими завданнями, роблячи матеріал максимально практичним.

4.2. Патерн Polyglot Persistence: Гібридна Архітектура

Аналіз порівняльної матриці підводить до логічного висновку: жодна окрема технологія зберігання даних не може ефективно задовольнити всі різноманітні вимоги ІАЗ ОУВ. Спроба "втиснути" всі дані – від кадрових записів до потоків відео з БПЛА – в єдину реляційну або документо-орієнтовану базу даних неминуче призведе до компромісів у продуктивності, масштабованості та гнучкості.

Рішенням є архітектурний патерн **Polyglot Persistence** (багатомовне зберігання). Його суть полягає у тому, що єдина комплексна система використовує кілька різних технологій

зберігання даних, обираючи для кожної підсистеми або навіть для кожного типу даних найкращий, найбільш відповідний інструмент.

Приклад гібридної архітектури ІАЗ ОУВ:

- **Ядро системи (PostgreSQL):** Тут зберігаються фундаментальні, повільно змінювані дані, що вимагають максимальної надійності та цілісності. Це таблиці користувачів, їхніх ролей та прав доступу, організаційно-штатна структура підрозділів, довідники. ACID-транзакції тут є абсолютно необхідними.
- **Підсистема розвідки (MongoDB):** Відповідає за збір, зберігання та аналіз розвідувальних зведень, досьє на об'єкти, звіти з різних джерел. Гнучка документна модель дозволяє легко інтегрувати дані різної структури без необхідності зупиняти систему для міграції схеми. Потужні можливості запитів, включаючи повнотекстовий та геопросторовий пошук, є критично важливими для аналітиків.
- **Підсистема моніторингу та телеметрії (Cassandra/ScyllaDB):** Ця підсистема обробляє величезні потоки часових рядів: телеметрію з техніки, дані з сенсорів, системні та мережеві логи. Висока пропускна здатність на запис та лінійна масштабованість дозволяють системі витримувати екстремальні навантаження.
- **Кеш та оперативні дані реального часу (Redis):** Використовується як високошвидкісний шар для даних, що потребують миттєвого доступу. Це кешування результатів складних запитів, зберігання сесій користувачів, поточні координати дружніх сил, черги завдань для обробки.
- **Підсистема аналізу зв'язків (Neo4j або інша графова БД):** Хоча детально не розглядалася, ця підсистема є важливою. Вона моделює та аналізує складні мережі: структуру командування противника, логістичні ланцюжки, зв'язки між особами та організаціями, виявлені в розвідданих. Графові запити дозволяють ефективно знаходити приховані зв'язки, слабкі місця та ключові вузли в мережі.

Впровадження патерну Polyglot Persistence, однак, має свою ціну. Воно переносить складність з рівня окремої бази даних (де ми намагаємося вирішити всі проблеми одним інструментом) на рівень архітектури додатку та операційної інфраструктури. Це створює нові виклики. Якщо раніше проблема полягала в тому, що SQL-база погано справляється з логами, то тепер, додавши Cassandra для логів, ми отримуємо низку нових проблем. По-перше, розробникам тепер потрібно володіти знаннями і SQL, і CQL, і MQL, і командами Redis. По-друге, команді експлуатації (DevOps/SRE) потрібно вміти розгортати, моніторити, налаштовувати, створювати резервні копії та відновлювати чотири абсолютно різні стеки технологій. По-третє, виникає складна проблема забезпечення узгодженості даних *між* різними сховищами. Наприклад, видалення користувача в основній PostgreSQL базі має каскадно ініціювати видалення його даних або їх анонімізацію в усіх інших підсистемах.

Переваги Polyglot Persistence можуть бути повністю реалізовані лише тоді, коли організація готова інвестувати в розвиток зрілої DevOps-культури. Це вимагає впровадження автоматизації на всіх рівнях (Infrastructure as Code, CI/CD), створення

єдиної системи моніторингу та логування (єдині дашборди для всіх типів БД), а також постійного навчання та підвищення кваліфікації персоналу. Таким чином, перехід до гібридної архітектури – це не просто технічне рішення, а стратегічна інвестиція в організаційні та процесні зміни.

4.3. Фреймворк для Вибору Технології

Щоб зробити процес вибору технології зберігання для нового модуля чи підсистеми ІАЗ ОУВ більш структурованим та обґрунтованим, пропонується наступний покроковий алгоритм у вигляді чек-листа:

1. **Аналіз даних:**
    - Яка структура даних? Вона жорстко фіксована (наприклад, структура наказу) чи динамічна та мінлива (наприклад, розвідзвіт)?
    - Який очікуваний обсяг даних (гігабайти, терабайти, петабайти)?
    - Які зв'язки між даними? Вони прості та чітко визначені (один-до-багатьох) чи складні, мережеві?
2. **Аналіз запитів (Workload Analysis):**
    - Які операції будуть переважати: читання (Read-heavy) чи запис (Write-heavy)? Наприклад, система телеметрії є екстремально Write-heavy, а довідкова система – Read-heavy.
    - Які основні патерни доступу до даних? Пошук за унікальним ключем? Запити по діапазону (наприклад, за часом)? Складні аналітичні запити з JOIN? Повнотекстовий пошук? Геопросторові запити?
3. **Вимоги до узгодженості:**
    - Наскільки критичною є миттєва, строга узгодженість даних для бізнес-логіки? Чи є операції, що вимагають ACID-транзакцій?
    - Чи припустима узгодженість в кінцевому рахунку (eventual consistency)? Який максимальний час затримки синхронізації є прийнятним?
4. **Вимоги до масштабованості та доступності:**
    - Яка очікувана кількість запитів на секунду (RPS)?
    - Які вимоги до відмовостійкості та часу простою (SLA)? Чи повинна система пережити відмову окремого сервера? Цілого дата-центру?
5. **Оцінка екосистеми та операційних витрат:**
    - Чи існують стабільні клієнтські бібліотеки для мов програмування, що використовуються в проекті?
    - Наскільки зрілими є інструменти для моніторингу, резервного копіювання та адміністрування?
    - Чи є в команді або на ринку достатня кількість фахівців з даної технології? Яка вартість володіння (TCO)?

Для швидкого попереднього вибору типу NoSQL бази даних можна використовувати наступну матрицю.

Таблиця 2: Матриця вибору типу NoSQL бази даних для завдань ІАЗ ОУВ

Тип завдання	Приклад в ІАЗ ОУВ	Рекомендований тип NoSQL	Ключова перевага
Зберігання складних, напівструктурованих документів	Розвідзведення, досьє на об'єкти, звіти, накази	**Документна (MongoDB)**	Гнучкість схеми, потужна мова запитів, вбудована підтримка гео-даних.
Кешування, сесії, лічильники, дані реального часу	Кеш звітів, поточні координати дружніх сил, сесії операторів	**Ключ-Значення (Redis)**	Надзвичайно низька затримка (in-memory), різноманітні структури даних.
Запис та аналіз великих обсягів часових рядів	Телеметрія з БПЛА, логи систем, дані з сенсорів	**Колонкова (Cassandra)**	Висока пропускна здатність на запис, лінійна масштабованість, оптимізація для діапазонних запитів за часом.
Аналіз складних зв'язків та мереж	Структура командування противника, логістичні ланцюжки, соціальні мережі	**Графова (Neo4j)**	Висока ефективність виконання запитів на обхід зв'язків (траверс графа).

Частина 5: Забезпечення Надійності та Безпеки Даних

5.1. Стратегії Резервного Копіювання та Відновлення

Незалежно від обраної технології, дані в ІАЗ ОУВ є надзвичайно цінним активом, втрата якого може мати катастрофічні наслідки. Тому надійна стратегія резервного копіювання (backup) та відновлення (recovery) є обов'язковою.

- **Для SQL (PostgreSQL):**
  - **Логічні бекапи (pg_dump):** Створюють файл з SQL-командами, що дозволяють відтворити схему та дані. Вони є гнучкими, дозволяють відновлювати окремі таблиці та не залежать від версії ОС чи архітектури процесора.
  - **Фізичні бекапи:** Копіювання файлів бази даних на рівні файлової системи. Вони значно швидші для великих баз даних.
  - **Point-in-Time Recovery (PITR):** Найпотужніший механізм, що дозволяє відновити базу даних до будь-якого моменту часу. Він базується на комбінації періодичного фізичного бекапу та безперервного архівування логів транзакцій (Write-Ahead Log - WAL).
  - **Потокова реплікація (Streaming Replication):** Налаштування одного або кількох серверів-реплік (standby), які в реальному часі отримують зміни з основного сервера (master). Це забезпечує наявність гарячого резерву (hot standby), на який можна миттєво переключити навантаження у випадку відмови основного сервера.
- **Для NoSQL (MongoDB/Cassandra):**
  - **Вбудована реплікація:** Більшість NoSQL систем спроектовані з урахуванням відмовостійкості. MongoDB використовує набори реплік (Replica Sets), де дані автоматично копіюються на кілька серверів. Cassandra використовує реплікацію між вузлами та навіть між географічно розподіленими дата-центрами. Це забезпечує високу доступність, але не є заміною бекапам (наприклад, випадкове видалення даних буде також відрепліковано).
  - **Снепшоти (Snapshots):** Створення "знімків" стану даних на певний момент часу. Це може бути реалізовано як на рівні самої БД, так і на рівні файлової системи або віртуалізованого середовища.
  - **Регулярне тестування процедур відновлення:** Створення резервних копій – це лише половина справи. Критично важливо регулярно проводити навчання з відновлення даних з бекапу на тестовому середовищі, щоб переконатися, що процес працює, та щоб команда була готова до реального інциденту.

5.2. Моделі Безпеки Баз Даних

Безпека даних в ІАЗ ОУВ є найвищим пріоритетом. Несанкціонований доступ, модифікація або витік інформації можуть призвести до провалу операцій та втрати особового складу. Необхідний комплексний, багатошаровий підхід до безпеки.

- **Аутентифікація та Авторизація:**
  - **Аутентифікація:** Процес перевірки, ким є користувач або сервіс, що намагається отримати доступ до БД. Необхідно використовувати надійні методи, такі як паролі з високою складністю, сертифікати або інтеграцію з централізованими системами управління ідентифікацією (наприклад, LDAP, Active Directory).
  - **Авторизація:** Процес надання прав доступу аутентифікованому користувачу. Тут ключовим є **принцип найменших привілеїв (Principle of Least Privilege)**: кожен користувач або додаток повинен мати доступ лише до тих даних і операцій, які є абсолютно необхідними для виконання його функцій. Це реалізується через модель доступу на основі ролей (Role-Based Access Control - RBAC), де привілеї групуються в ролі (наприклад, "аналітик", "оператор", "адміністратор"), а користувачам призначаються відповідні ролі.
- **Шифрування даних:**
  - **Encryption in Transit (Шифрування при передачі):** Весь мережевий трафік між додатками та сервером бази даних, а також між вузлами кластера БД, повинен бути зашифрований за допомогою протоколів TLS/SSL. Це запобігає перехопленню та прослуховуванню даних у мережі.
  - **Encryption at Rest (Шифрування при зберіганні):** Дані, що зберігаються на дисках (файли БД, бекапи, логи), повинні бути зашифровані. Сучасні СУБД пропонують механізми прозорого шифрування даних (Transparent Data Encryption - TDE), які шифрують дані автоматично, не вимагаючи змін у коді додатків.
- **Аудит:**
  - Необхідно вмикати та регулярно аналізувати логи аудиту бази даних. Ці логи повинні фіксувати всі важливі події: спроби входу (успішні та невдалі), доступ до чутливих таблиць, зміну даних, модифікацію прав доступу. Аналіз логів аудиту дозволяє виявляти підозрілу активність, розслідувати інциденти безпеки та відповідати на питання "хто, що, коли і звідки робив".
- **Маскування та анонімізація даних:**
  - Для середовищ розробки, тестування та навчання часто потрібні реалістичні дані. Однак використання реальних бойових даних у цих середовищах є неприпустимим ризиком. Необхідно використовувати техніки маскування (заміна чутливих даних, таких як імена чи координати, на фіктивні, але правдоподібні значення) або анонімізації для створення безпечних наборів даних.

Висновки

Архітектура зберігання даних для сучасної Інформаційно-аналітичної системи Об'єднаного управління військами не може базуватися на єдиному універсальному рішенні. Ефективна система повинна бути гібридною, використовуючи архітектурний патерн **Polyglot Persistence**.

1. **Реляційні бази даних (SQL)**, такі як PostgreSQL, залишаються незамінними для ядра системи та облікових підсистем, де потрібні строгі гарантії цілісності даних (ACID) та потужні аналітичні можливості для роботи зі структурованою інформацією.
2. **NoSQL бази даних** є критично важливими для роботи з величезними обсягами різноманітних та швидкозмінних даних. Документні БД (MongoDB) ідеально підходять для гнучкого зберігання розвідданих, колонкові (Cassandra) – для обробки потоків телеметрії, а сховища ключ-значення (Redis) – для забезпечення миттєвого доступу до оперативних даних та кешування.
3. Вибір між узгодженістю та доступністю, формалізований у **теоремі CAP**, є не просто технічним, а фундаментальним операційно-доктринальним рішенням. Для кожної підсистеми необхідно свідомо обирати модель (CP або AP) на основі аналізу "ціни помилки" проти "ціни затримки".
4. Впровадження гібридної архітектури вимагає високого рівня технічної зрілості організації, інвестицій в автоматизацію (DevOps), моніторинг та, що найважливіше, в постійне навчання та розвиток компетенцій технічних фахівців.
5. Безпека та надійність даних повинні бути закладені в архітектуру з самого початку. Комплексний підхід, що включає надійне резервне копіювання, багатошарову модель безпеки (RBAC, шифрування in-transit та at-rest) та постійний аудит, є обов'язковою умовою функціонування системи в умовах сучасних кіберзагроз.

Таким чином, успіх ІАЗ ОУВ залежить від здатності архітекторів та розробників мислити гнучко, обираючи правильний інструмент для кожного конкретного завдання та поєднуючи їх у єдину, надійну та масштабовану систему, здатну забезпечити інформаційну перевагу на полі бою.