НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Навчально-науковий інститут телекомунікаційних систем

Кафедра електронних комунікацій та інтернету речей

«На правах рукопису»
УДК 004.764:004.77

До захисту допущено:

Завідувач кафедри

_____ Анатолій МАКАРЕНКО

«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою «Системи електронних комунікацій та Інтернету речей»

зі спеціальності 172 «Електронні комунікації та радіотехніка»

на тему: «Дослідження хмарних сервісів і платформ для побудови рішень IoT»

Виконав (-ла): в
студент (-ка) ІІ курсу, групи ЦС-41мп
Родічев Ігор Дмитрович _____

Науковий керівник:
к.т.н., доцент, доцент кафедри ЕКІР
Осипчук Сергій Олександрович _____

Рецензент:
к.т.н., доцент кафедри ТК, с.н.с.
Міночкін Дмитро Анатолійович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2025 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Навчально-науковий інститут телекомунікаційних систем
Кафедра електронних комунікацій та інтернету речей

Рівень вищої освіти – другий (магістерський)

Спеціальність – 172 «Електронні комунікації та радіотехніка»

Освітньо-професійна програма «Системи електронних комунікацій та Інтернету речей»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Анатолій МАКАРЕНКО
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Родічеву Ігорю Дмитровичу

1. Тема дисертації «Дослідження хмарних сервісів і платформ для побудови рішень IoT», науковий керівник дисертації Осипчук Сергій Олександрович, кандидат наук, доцент, затверджені наказом по університету від «3» листопада 2025 р. №4772-с.

2. Термін подання студентом дисертації: 17 грудня 2025.

3. Об’єкт дослідження: процес вибору та розгортання хмарної інфраструктури для рішень Інтернету речей.

4. Предмет дослідження: методологія порівняльного оцінювання та обґрунтованого вибору хмарних сервісів і платформ для хостингу IoT-рішень, зокрема із застосуванням математичної моделі.

5. Перелік завдань, які потрібно розробити:

1. Проаналізувати специфіку рішень Інтернету речей, їх еволюцію, архітектуру та вимоги до хмарних технологій.

2. Здійснити огляд та порівняльний аналіз ринку хмарних сервісів та спеціалізованих платформ для IoT.

3. Розробити формалізовану методику оцінювання та вибору платформ на основі функції корисності та зважених критеріїв (час, бюджет, функціональність).

4. Провести практичну перевірку методики шляхом порівняння реалізацій на базі AWS, Blynk та кастомного хостингу.

5. Розробити рекомендації щодо створення стійких та економічно ефективних IoT-рішень.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: робота містить ілюстрації та таблиці.

7. Орієнтовний перелік публікацій:

Yadav, P., Osypchuk, S., Bidikar, M., & Rodichev, I. (2025). Cloud services and platforms research for internet of things applications deployment. The 19th International Scientific and Technical Conference "Modern Challenges in Telecommunications" (MCT-2025), Kyiv, Ukraine.

8. Дата видачі завдання: 14.10.2024

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Вибір та формування теми дисертації	14.10.2024	Виконано
2	Аналіз специфіки IoT: архітектури, протоколи, хмарні моделі	01.09.2025 – 14.09.2025	Виконано
3	Дослідження ринку публічних хмарних провайдерів	15.09.2025 – 28.09.2025	Виконано
4	Дослідження ринку та спеціалізованих платформ	29.09.2025 – 12.10.2025	Виконано
5	Розробка методики оцінювання: визначення критеріїв математична модель	13.10.2025 – 26.10.2025	Виконано
6	Практичне застосування методики: оцінювання AWS, Blynk, Custom Hosting	27.10.2025 – 09.11.2025	Виконано
7	Розробка рекомендацій щодо забезпечення стійкості та економічної ефективності IoT-рішень	10.11.2025 – 23.11.2025	Виконано
8	Оформлення магістерської дисертації	10.12.2025	Виконано

Студент Ігор РОДІЧЕВ

Науковий керівник Сергій ОСИПЧУК

РЕФЕРАТ

Темою магістерської дисертації є дослідження хмарних сервісів і платформ для побудови рішень Інтернету речей (IoT). Робота містить 89 сторінок, зокрема 9 ілюстрацій, 11 таблиць та 18 джерел інформації.

Актуальність теми зумовлена стрімким розвитком ринку IoT та різноманіттям доступних хмарних рішень, що ускладнює вибір оптимальної інфраструктури. Необхідність обґрунтованого вибору платформи є критичною для мінімізації витрат та забезпечення надійності систем.

Мета дисертації полягає у розробці формалізованої методики для порівняльного оцінювання та вибору хмарних сервісів і платформ, а також наданні рекомендацій щодо створення стійких та економічно ефективних IoT-рішень.

Об’єктом дослідження є процес вибору та розгортання хмарної інфраструктури для рішень IoT. Предметом дослідження є методологія оцінювання хмарних платформ на основі критеріїв часу, бюджету та функціональності.

При виконанні роботи застосовувалося математичне моделювання (функція оцінки корисності) та економічне моделювання (розрахунок TCO) для порівняння платформ AWS, Blynk та кастомного хостингу.

У дисертації запропоновано методику багатокритеріального вибору платформи з урахуванням вагових коефіцієнтів пріоритетів проєкту. Надано практичні рекомендації щодо забезпечення резильєнтності (стійкості) систем в умовах нестабільного зв'язку та оптимізації енергоспоживання.

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ РЕЧЕЙ, IOT, ХМАРНІ ПЛАТФОРМИ, AWS, BLYNK, TCO, ФУНКЦІЯ КОРИСНОСТІ, FINOPS, MQTT, ЕНЕРГОЕФЕКТИВНІСТЬ.

ABSTRACT

The topic of the Master’s thesis is the research of cloud services and platforms for building Internet of Things (IoT) solutions. The work contains 89 pages, including 9 illustrations, 11 tables, and 18 sources.

The relevance of the topic is due to the rapid growth of the IoT market and the variety of available cloud solutions, which complicates the choice of optimal infrastructure. A justified choice of platform is critical for minimizing costs and ensuring system reliability.

The purpose of the thesis is to develop a formalized methodology for comparative evaluation and selection of cloud services and platforms, as well as to provide recommendations for creating sustainable and cost-effective IoT solutions.

The object of research is the process of selecting and deploying cloud infrastructure for IoT solutions. The subject of research is the methodology for evaluating cloud platforms based on time, budget, and functionality criteria.

The work employed mathematical modeling (utility scoring function) and economic modeling (TCO calculation) to compare AWS, Blynk, and custom hosting platforms.

The thesis proposes a multi-criteria platform selection methodology considering the weight coefficients of project priorities. Practical recommendations are provided for ensuring system resilience under unstable connectivity conditions and optimizing power consumption.

KEYWORDS: INTERNET OF THINGS, IOT, CLOUD PLATFORMS, AWS, BLYNK, TCO, UTILITY FUNCTION, FINOPS, MQTT, ENERGY EFFICIENCY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

AI – Artificial Intelligence – Штучний інтелект.

API – Application Programming Interface – Прикладний програмний інтерфейс.

AWS – Amazon Web Services – Веб-сервіси Amazon.

CapEx – Capital Expenditure – Капітальні витрати.

CRM – Customer Relationship Management – Система управління взаємовідносинами з клієнтами.

DevOps – Development and Operations – Методологія взаємодії розробників та фахівців з інформаційно-технологічного обслуговування.

ERP – Enterprise Resource Planning – Система планування ресурсів підприємства.

FinOps – Financial Operations – Практика управління витратами на хмарні обчислення.

FRs – Functional Requirements – Функціональні вимоги.

GCP – Google Cloud Platform – Хмарна платформа Google.

GDPR – General Data Protection Regulation – Загальний регламент захисту даних.

HTTP – HyperText Transfer Protocol – Протокол передачі гіпертексту.

HTTPS – HyperText Transfer Protocol Secure – Захищений протокол передачі гіпертексту.

IaaS – Infrastructure as a Service – Інфраструктура як сервіс.

PaaS – Platform as a Service – Платформа як сервіс.

SaaS – Software as a Service – Програмне забезпечення як сервіс.

IAM – Identity and Access Management – Управління ідентифікацією та доступом.

IIoT – Industrial Internet of Things – Промисловий Інтернет речей.

IoT – Internet of Things – Інтернет речей.

JSON – JavaScript Object Notation – Текстовий формат обміну даними.

LPWAN – Low-Power Wide-Area Network – Енергоефективна мережа дальнього радіуса дії.

MCU – Microcontroller Unit – Мікроконтролер.

ML – Machine Learning – Машинне навчання.

MQTT – Message Queuing Telemetry Transport – Протокол передачі телеметрії.

MVP – Minimum Viable Product – Мінімально життєздатний продукт.

NB-IoT – Narrowband Internet of Things – Вузькосмуговий Інтернет речей.

NFRs – Non-Functional Requirements – Нефункціональні вимоги.

NoSQL – Not Only SQL – Не лише SQL (тип баз даних).

OEM – Original Equipment Manufacturer – Оригінальний виробник обладнання.

OpEx – Operational Expenditure – Операційні витрати.

OTA – Over-The-Air – Оновлення програмного забезпечення «по повітрю».

PLC – Programmable Logic Controller – Програмований логічний контролер.

QoS – Quality of Service – Якість обслуговування.

REST – Representational State Transfer – Архітектурний стиль взаємодії компонентів.

SCADA – Supervisory Control and Data Acquisition – Диспетчерське управління та збір даних.

SLA – Service Level Agreement – Угода про рівень обслуговування.

SNS – Simple Notification Service – Сервіс сповіщень.

TCO – Total Cost of Ownership – Сукупна вартість володіння.

TLS – Transport Layer Security – Протокол захисту транспортного рівня.

VPS – Virtual Private Server – Віртуальний приватний сервер.

ІКТ – Інформаційно-комунікаційні технології.

ВСТУП

Магістерську дисертацію присвячено темі «Дослідження хмарних сервісів і платформ для побудови рішень IoT». Актуальність цього дослідження зумовлена стрімким поширенням технологій Інтернету речей (IoT) у різноманітних галузях – від промисловості та сільського господарства до розумних міст та охорони здоров'я. Цей процес генерує величезні обсяги даних та вимагає надійних, масштабованих і гнучких інфраструктурних рішень. Хмарні сервіси та платформи стали де-факто стандартом для розгортання, управління та аналізу даних IoT-систем, пропонуючи потужні інструменти для обробки інформації та швидкого виведення продуктів на ринок.

Однак, сучасний ринок хмарних рішень характеризується значною різноманітністю пропозицій: від комплексних екосистем глобальних провайдерів (AWS, Azure, GCP) до спеціалізованих платформ (Blynk, ThingsBoard) та можливостей використання кастомних хостингових рішень. Кожен із цих підходів має унікальні технічні характеристики, моделі ціноутворення, рівні безпеки та складності впровадження. Це створює суттєву проблему для розробників та архітекторів IoT-систем, оскільки вибір оптимальної платформи стає нетривіальним завданням, яке часто базується на неповній інформації або суб'єктивних перевагах. Відсутність формалізованих, об'єктивних методик для порівняльного аналізу призводить до ризиків неефективного використання ресурсів, перевитрат бюджету та невідповідності кінцевого рішення функціональним вимогам. Тому розробка науково обґрунтованого підходу, зокрема математичної моделі для оцінки корисності хмарних сервісів, є актуальною науково-практичною задачею.

У роботі комплексно досліджується сучасний ландшафт хмарних сервісів та платформ для IoT. Розглядається специфіка IoT-рішень, їх еволюція, архітектура, а також ключові функціональні та нефункціональні вимоги. Детально аналізуються різні типи хмарних сервісів (IaaS, PaaS, SaaS)

та моделі розгортання, проводиться порівняльний аналіз провідних провайдерів (AWS, Azure, GCP) та спеціалізованих платформ . Важливою частиною роботи є аналіз практичних прикладів реалізації IoT-рішень на базі платформи Blynk та кастомного хостингу, а також формулювання рекомендацій щодо створення стійких систем.

Метою магістерської дисертації є вирішення науково-практичної задачі підвищення ефективності та обґрунтованості процесу вибору хмарних сервісів і платформ для розгортання та хостингу рішень Інтернету речей. Для досягнення цієї мети у роботі передбачено розробку, теоретичне обґрунтування та практичну апробацію формалізованої методики – математичної моделі на основі функції оцінки корисності. Ця модель інтегрує ключові критерії вибору: очікуваний час впровадження, бюджетні обмеження, ступінь відповідності функціональним та нефункціональним вимогам, а застосування вагових коефіцієнтів дозволяє адаптувати її до пріоритетів різних бізнес-завдань .

Об'єктом дослідження виступає процес вибору та розгортання хмарної інфраструктури для рішень Інтернету речей, що включає аналіз доступних платформ, їх технічних можливостей та моделей ціноутворення. Предметом дослідження є методологія порівняльного оцінювання та обґрунтованого вибору хмарних сервісів і платформ, зокрема розроблена математична модель як інструмент для прийняття оптимальних архітектурних рішень при проєктуванні IoT-систем.

1 СПЕЦИФІКА РІШЕНЬ ІНТЕРНЕТУ РЕЧЕЙ ТА ХМАРНИХ ТЕХНОЛОГІЙ

1.1 Еволюція технологій Інтернету речей

Концепція Інтернету речей, що передбачає підключення фізичних об'єктів до мережі для збору та обміну даними, не є абсолютно новою. Її коріння сягає кількох десятиліть розвитку інформаційно-комунікаційних технологій. Проте саме в останні 5–10 років, завдяки значному прогресу в мікроелектроніці, бездротових мережах та хмарних обчисленнях, IoT перетворився з футуристичної ідеї на потужну технологічну парадигму, що змінює численні галузі економіки та повсякденне життя [1, 3].

Еволюцію технологій, що призвела до появи сучасного IoT, можна умовно розділити на кілька ключових фаз, які відображають зміни в способах комунікації та типах взаємодіючих об'єктів (рис. 1.1).

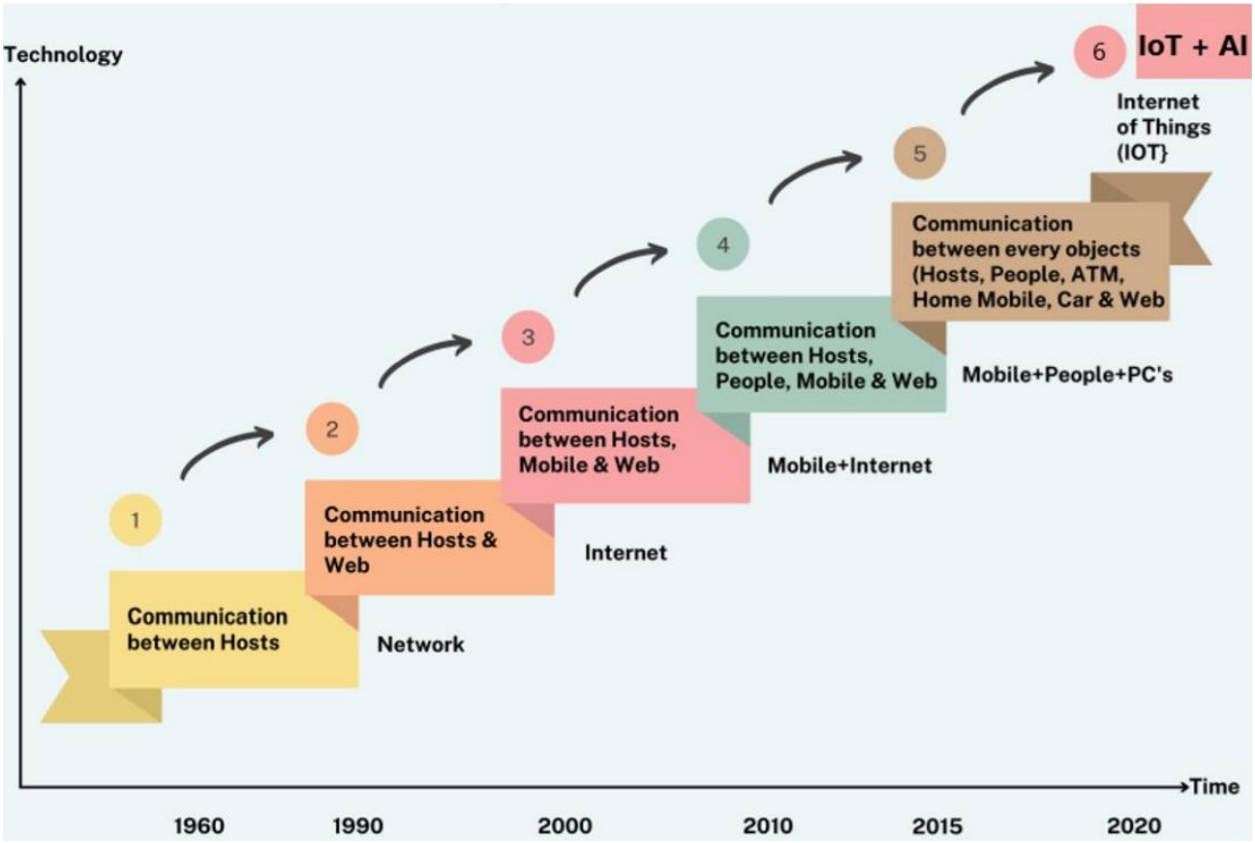

Рисунок 1.1 – Основні етапи еволюції технологій IoT [1]

1. *Комунікація між хостами (до ~1990 р.):* Початковий етап характеризувався створенням перших комп'ютерних мереж (наприклад, ARPANET), де основною метою була передача даних між окремими великими обчислювальними машинами (хостами).
2. *Комунікація між хостами та Web (1990-ті рр.):* Поява Всесвітньої павутини та протоколу HTTP революціонізувала обмін інформацією, зробивши її доступною для широкого кола користувачів через веб-браузери. Комунікація все ще відбувалася переважно між серверами та клієнтськими комп'ютерами.
3. *Комунікація між хостами, мобільними пристроями та Web (2000-ні рр.):* Розповсюдження мобільних телефонів та поява перших смартфонів розширили можливості доступу до Інтернету. Комунікація стала включати взаємодію між серверами, настільними ПК та мобільними пристроями.
4. *Комунікація між хостами, людьми, мобільними пристроями та Web (2010-ті рр.):* Поява соціальних мереж, розвиток мобільних додатків та хмарних сервісів призвели до епохи, де люди активно взаємодіють між собою та з цифровим контентом через різноманітні пристрої, підключені до Інтернету.
5. *Комунікація між усіма об'єктами (з ~2015 р.):* Цей етап характеризується підключенням до мережі не лише комп'ютерів та мобільних пристроїв, але й різноманітних фізичних об'єктів ("речей") – сенсорів, побутової техніки, промислового обладнання, автомобілів тощо. Саме це явище отримало назву Інтернет речей.
6. *IoT + Штучний інтелект (сьогодення та майбутнє):* Сучасний етап розвитку IoT нерозривно пов'язаний з інтеграцією технологій штучного інтелекту та машинного навчання для аналізу величезних обсягів даних, що генеруються підключеними пристроями, та прийняття інтелектуальних рішень в автоматичному режимі [4, 15, 16].

Таким чином, IoT є логічним продовженням багаторічного розвитку комунікаційних технологій, що перейшли від з'єднання комп'ютерів до глобальної мережі взаємопов'язаних фізичних та цифрових об'єктів.

Розуміння цієї еволюції є важливим для аналізу сучасних IoT-рішень та інфраструктури, необхідної для їх підтримки.

1.2 Визначення та ключові поняття IoT

Термін *Інтернет речей* (Internet of Things, IoT) використовується для опису мережі фізичних об'єктів – "речей" – які оснащені сенсорами, програмним забезпеченням та іншими технологіями, що дозволяють їм підключатися до Інтернету та обмінюватися даними з іншими пристроями та системами [1, 2]. Ці "речі" можуть варіюватися від простих побутових приладів, таких як термостати чи лампочки, до складних промислових машин, медичного обладнання та транспортних засобів.

Ключова ідея IoT полягає в тому, щоб надати можливість цим об'єктам *збирати дані* про навколишнє середовище або власний стан, *передавати ці дані* через мережу та, в багатьох випадках, *реагувати* на отриману інформацію або команди без прямого втручання людини.

Для розуміння IoT важливо визначити декілька основних понять:

- *Речі (Things):* Фізичні об'єкти, що є частиною IoT-системи. Вони зазвичай містять вбудовані системи (мікроконтролери), сенсори для збору даних (температура, рух, освітленість тощо) та/або виконавчі механізми (актуатори) для виконання дій (ввімкнення/вимкнення, регулювання).
- *Підключення (Connectivity):* Механізми та протоколи, що дозволяють "речам" обмінюватися даними з іншими пристроями, шлюзами або хмарними платформами. Це можуть бути як дротові (Ethernet), так і бездротові технології (Wi-Fi, Bluetooth, LoRaWAN, NB-IoT, 5G тощо).
- *Дані (Data):* Інформація, що збирається сенсорами та передається пристроями. Обробка та аналіз цих даних є основною цінністю IoT-рішень.
- *Платформа IoT:* Програмно-апаратний комплекс (часто хмарний), що забезпечує підключення пристроїв, збір, зберігання, обробку та аналіз даних, а також управління пристроями та взаємодію з користувацькими додатками.

- *Аналітика (Analytics):* Процес перетворення "сирих" даних, отриманих від IoT-пристроїв, на корисну інформацію та знання шляхом застосування статистичних методів, алгоритмів машинного навчання та інших інструментів аналізу.
- *Застосунок (Application):* Програмне забезпечення (веб-інтерфейс, мобільний додаток), що надає користувачам доступ до даних та функцій управління IoT-системою.

Розуміння цих ключових компонентів та їх взаємодії є основою для проєктування та аналізу будь-якого IoT-рішення.

1.3 Сфери застосування IoT та бізнес-домени

Технології Інтернету речей знайшли широке застосування у численних галузях економіки та аспектах суспільного життя, створюючи нові бізнес-моделі та підвищуючи ефективність існуючих процесів [3, 5]. Універсальність IoT дозволяє адаптувати його принципи до специфічних потреб різних доменів. Розглянемо основні сфери застосування (рис. 1.2).

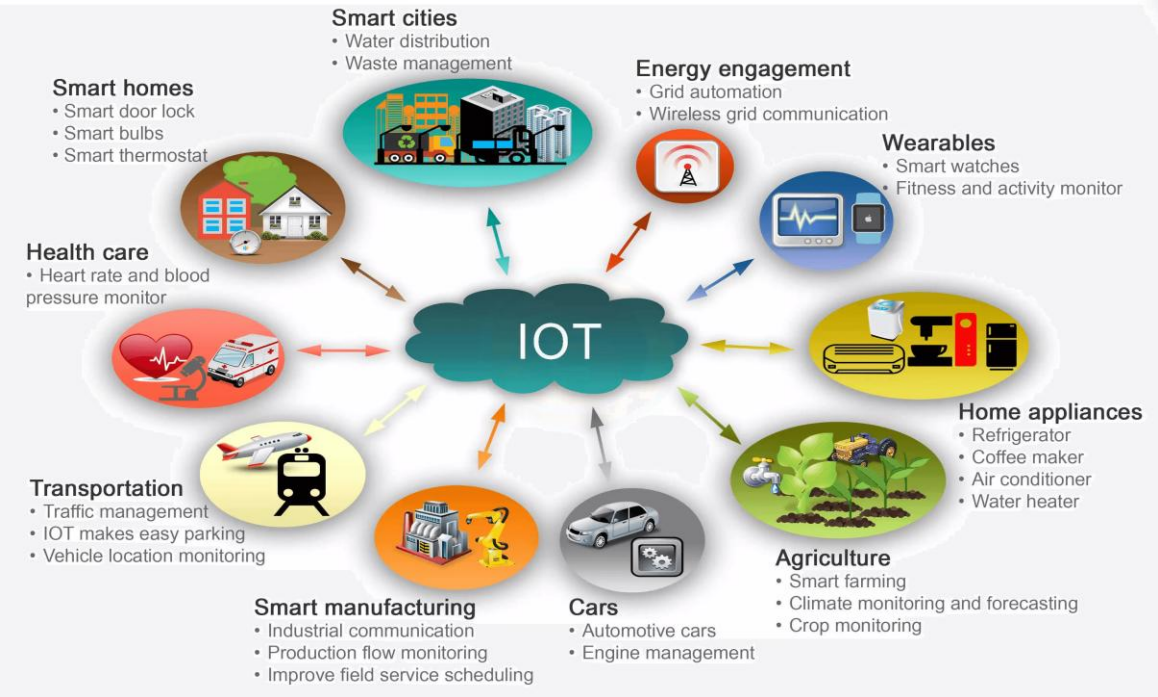

Рисунок 1.2 – Основні сфери застосування IoT [1]

- *Охорона здоров'я (Healthcare):* Носимі пристрої (фітнес-трекери, смарт-годинники) та медичні сенсори дозволяють здійснювати віддалений моніторинг показників життєдіяльності пацієнтів (пульс, тиск, рівень глюкози), що сприяє ранній діагностиці та персоналізованому лікуванню. Системи "розумної лікарні" оптимізують управління обладнанням та ресурсами.
- *Розумні будинки (Smart Homes):* IoT дозволяє автоматизувати управління освітленням, кліматом (розумні термостати), безпекою (датчики руху, розумні замки), побутовою технікою (холодильники, кавоварки), підвищуючи комфорт, безпеку та енергоефективність житла.
- *Розумні міста (Smart Cities):* Впровадження IoT допомагає оптимізувати міську інфраструктуру: управління трафіком (розумні світлофори, системи паркування), моніторинг стану навколишнього середовища (якість повітря, рівень шуму), управління ресурсами (водопостачання, енергоспоживання, вивіз сміття).
- *Промисловість (Industry, IIoT - Industrial Internet of Things):* У промисловості IoT використовується для моніторингу стану обладнання (predictive maintenance), оптимізації виробничих процесів, управління логістикою та ланцюгами постачання, підвищення безпеки праці.
- *Енергетика (Smart Grid):* IoT дозволяє створювати "розумні мережі" електропостачання, які забезпечують більш ефективний розподіл енергії, моніторинг споживання в реальному часі, інтеграцію відновлюваних джерел енергії та швидке реагування на аварійні ситуації.
- *Сільське господарство (Agriculture):* "Розумне фермерство" (smart farming) використовує IoT-сенсори для моніторингу стану ґрунту, вологості, температури, здоров'я тварин, що дозволяє оптимізувати полив, внесення добрив, прогнозувати врожайність та ефективніше управляти ресурсами.

- *Транспорт та логістика (Transport):* Підключені автомобілі, системи моніторингу вантажів та управління транспортними потоками підвищують безпеку, ефективність перевезень та оптимізують логістичні маршрути.
- *Роздрібна торгівля (Retail):* IoT використовується для управління запасами (розумні полиці), аналізу поведінки покупців, персоналізації пропозицій та оптимізації роботи магазинів.

Цей перелік не є вичерпним, оскільки IoT продовжує проникати в нові сфери, такі як екологічний моніторинг, безпека, освіта тощо. Широта застосування підкреслює універсальність технології та її значний потенціал для трансформації різних аспектів людської діяльності.

1.4 Життєвий цикл продукту IoT

Для повного розуміння процесу розробки, розгортання та експлуатації IoT-рішень важливо розглянути їх життєвий цикл. Подібно до будь-якого технологічного продукту, IoT-рішення проходить кілька послідовних етапів від ідеї до виведення з експлуатації. Розуміння цих етапів дозволяє ефективніше планувати ресурси, керувати проєктом та забезпечувати довгострокову підтримку системи [Стаття/Презентація].

Типовий життєвий цикл IoT-продукту можна представити у вигляді циклічної моделі, що включає чотири основні фази (рис. 1.3).

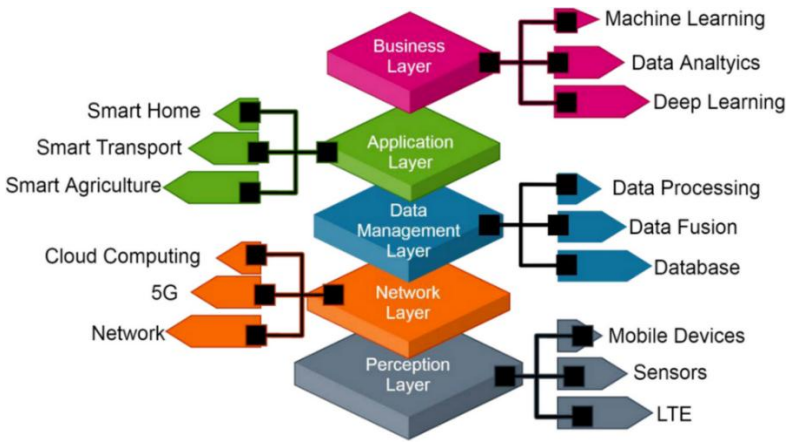

Рисунок 1.3 – Основні фази життєвого циклу IoT-продукту [1]

1. *Фаза проєктування (Design):* Це початковий етап, на якому формулюється ідея продукту, визначаються його цілі та бізнес-вимоги. На цій фазі відбувається збір та аналіз функціональних та нефункціональних вимог до рішення, вибір апаратної платформи (сенсори, мікроконтролери), протоколів зв'язку, хмарної платформи або сервісів для хостингу, а також розробка архітектури системи та дизайну користувацького інтерфейсу. Важливим аспектом є також планування безпеки та масштабованості рішення.
2. *Фаза розгортання (Deploy):* На цьому етапі розроблене рішення впроваджується в реальному середовищі. Це включає виробництво або закупівлю апаратних компонентів, встановлення та налаштування пристроїв, конфігурацію мережевої інфраструктури, розгортання програмного забезпечення на пристроях та в хмарі, а також тестування всієї системи на працездатність та відповідність вимогам.
3. *Фаза управління та експлуатації (Manage):* Це найтриваліша фаза життєвого циклу, протягом якої IoT-рішення активно використовується. Вона включає моніторинг стану пристроїв та системи в цілому, збір та аналіз даних, регулярне оновлення програмного забезпечення (включаючи оновлення "по повітрю" – OTA, Over-The-Air), технічне обслуговування апаратних компонентів, управління користувачами та забезпечення безпеки системи від нових загроз.
4. *Фаза виведення з експлуатації (Decommission):* На цьому етапі продукт або його окремі компоненти завершують свій життєвий цикл. Це може бути пов'язано із застаріванням технологій, зміною бізнес-потреб або досягненням кінцевого терміну служби обладнання. Фаза включає безпечне відключення пристроїв від мережі, видалення даних, утилізацію або переробку апаратних компонентів відповідно до екологічних норм.

Розуміння цих фаз та завдань, що вирішуються на кожній з них, є критично важливим для успішного створення та підтримки IoT-рішень, а

також для вибору відповідних хмарних сервісів та платформ, які мають підтримувати всі етапи життєвого циклу.

1.5 Вимоги та обмеження при проєктуванні IoT-рішень

Успіх будь-якого IoT-рішення значною мірою залежить від чіткого визначення вимог та врахування наявних обмежень ще на етапі проєктування (фаза *Design* у життєвому циклі продукту). Ці фактори визначають архітектуру системи, вибір технологій, апаратного забезпечення, хмарних сервісів та загальну стратегію реалізації [1]. Недооцінка або неправильне формулювання вимог може призвести до створення системи, яка не відповідає очікуванням користувачів, є ненадійною, незахищеною або економічно невиправданою.

Традиційно вимоги до програмних та апаратних систем поділяють на функціональні та нефункціональні. Окрім них, при розробці IoT-рішень необхідно враховувати специфічні обмеження проєкту.

1.5.1 Функціональні вимоги

Функціональні вимоги (Functional Requirements, FRs) описують, *що саме* повинна робити система, які функції вона має виконувати з точки зору користувача чи бізнес-логіки. Вони визначають основну поведінку та можливості IoT-рішення.

Приклади функціональних вимог для IoT-системи можуть включати:

- Здатність системи вимірювати температуру та вологість повітря з певною точністю та періодичністю.
- Можливість користувача дистанційно вмикати/вимикати освітлення через мобільний додаток.
- Надсилання системою сповіщень (наприклад, SMS або push-повідомлення) при виявленні руху в приміщенні.

- Відображення на веб-панелі графіків зібраних даних за обраний період часу.
- Вимоги до фізичних характеристик пристрою, таких як розмір, форма, колір, тип живлення (батарейне, від мережі).

Чітке визначення функціональних вимог є першочерговим завданням, оскільки вони лягають в основу проєктування всіх інших аспектів системи.

1.5.2 Нефункціональні вимоги

Нефункціональні вимоги (Non-Functional Requirements, NFRs) визначають, *як* система повинна виконувати свої функції. Вони описують атрибути якості, експлуатаційні характеристики та обмеження, що накладаються на систему. NFRs часто є критично важливими для користувацького досвіду та загальної життєздатності IoT-рішення, хоча кінцеві користувачі можуть не взаємодіяти з ними безпосередньо.

Ключові нефункціональні вимоги для IoT-систем включають:

- *Продуктивність (Performance):* Вимоги до швидкості реакції системи, часу передачі даних, кількості одночасно підтримуваних пристроїв чи користувачів. Наприклад, система моніторингу промислового обладнання може вимагати передачі даних з мінімальною затримкою.
- *Надійність (Reliability):* Здатність системи працювати без збоїв протягом певного часу або в певних умовах. Це включає стійкість до відмов окремих компонентів, мережевих проблем.
- *Доступність (Availability):* Відсоток часу, протягом якого система є доступною для використання. Висока доступність є критичною для систем моніторингу безпеки або здоров'я.
- *Масштабованість (Scalability):* Здатність системи адаптуватися до зростання навантаження (збільшення кількості пристроїв, обсягу даних, кількості користувачів) без суттєвого погіршення продуктивності.

- *Безпека (Security):* Вимоги до захисту даних (конфіденційність, цілісність) при передачі та зберіганні, автентифікації та авторизації пристроїв і користувачів, захисту від несанкціонованого доступу та атак [7].
- *Сумісність (Interoperability):* Здатність системи взаємодіяти з іншими системами або компонентами (наприклад, інтеграція з існуючими бізнес-системами).
- *Простота використання (Usability):* Легкість, з якою користувачі можуть взаємодіяти з системою (інтуїтивність інтерфейсу, простота налаштування).
- *Підтримуваність (Maintainability):* Легкість внесення змін, виправлення помилок та оновлення системи протягом її життєвого циклу.

Невиконання нефункціональних вимог може призвести до негативного досвіду користувачів (наприклад, повільна робота додатку, часті збої), фінансових втрат або навіть критичних наслідків у сферах безпеки чи охорони здоров'я.

1.5.3 Обмеження

Обмеження (Constraints) – це жорсткі рамки або умови, які впливають на процес розробки та кінцевий вигляд IoT-рішення, і які зазвичай неможливо змінити або обійти. Вони можуть суттєво звузити вибір можливих архітектурних рішень та технологій.

Типові обмеження для IoT-проєктів включають:

- *Бюджет (Budget):* Фінансові обмеження на розробку, закупівлю обладнання, розгортання та експлуатацію системи. Вартість хмарних сервісів є важливим фактором.
- *Час (Time):* Терміни розробки та виведення продукту на ринок (time-to-market).
- *Технологічні обмеження (Technology):* Доступність певних технологій зв'язку в місці розгортання, сумісність з існуючим обладнанням, обмеження обраної

апаратної платформи (обчислювальна потужність, пам'ять, енергоспоживання).

- *Регуляторні вимоги та стандарти (Regulatory):* Необхідність дотримання галузевих стандартів, вимог сертифікації, законодавства щодо захисту даних (наприклад, GDPR).
- *Кваліфікація команди (Engineering Skills):* Наявність у команди розробників необхідних знань та досвіду роботи з обраними технологіями.
- *Фізичні обмеження (Hardware):* Вимоги до розміру, ваги, стійкості пристроїв до умов навколишнього середовища (температура, вологість, вібрації).

Ретельний аналіз та документування всіх вимог (FRs, NFRs) та обмежень на ранніх етапах проєктування є запорукою створення успішного та життєздатного IoT-рішення. Саме ці вимоги та обмеження стануть основою для вибору оптимальних хмарних сервісів та платформ, що буде детально розглянуто в наступних розділах.

1.6 Багаторівнева архітектура IoT-рішен

Для ефективного проєктування, розгортання та управління IoT-рішеннями критично важливо розуміти їхню архітектуру. Незважаючи на різноманітність конкретних реалізацій, більшість IoT-систем можна описати за допомогою багаторівневої (або багатошарової) моделі. Ця модель допомагає декомпозувати складну систему на логічні компоненти, кожен з яких виконує специфічні функції.

У той час як ранні моделі пропонували просту трирівневу структуру (пристрої, мережа, хмара), сучасні підходи, враховуючи зростаючу складність систем та появу граничних (edge) обчислень, частіше використовують п'ятирівневу модель. Вона забезпечує більш детальне уявлення про потоки даних та етапи їх обробки.

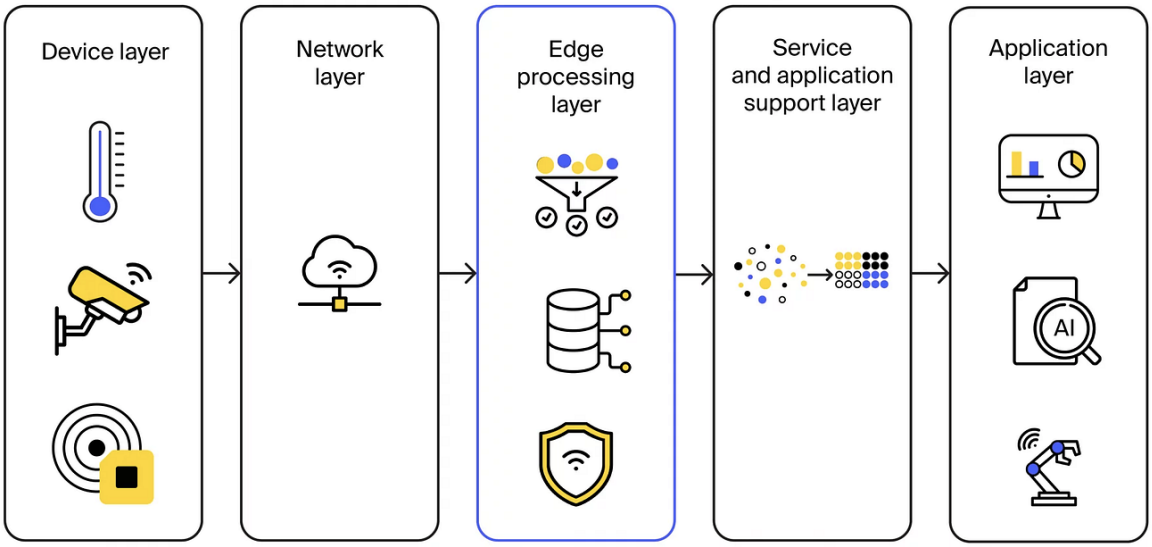

Рисунок 1.4 – Основні компоненти архітектури IoT-рішення

1.6.1 Рівень пристроїв (Device Layer)

Цей рівень є фізичною основою будь-якої IoT-системи. Він складається безпосередньо з "речей" – кінцевих пристроїв, які безпосередньо взаємодіють з фізичним світом. Основними компонентами цього рівня є *сенсори (Sensors)*, що збирають дані про навколишнє середовище або стан об'єкта (наприклад, датчики температури, вологості, руху, GPS-модулі, камери), та *виконавчі механізми (Actuators)*, що виконують дії у фізичному світі на основі отриманих команд (реле, сервоприводи, клапани). "Мозком" пристрою зазвичай виступає мікроконтролер (MCU), який зчитує дані з сенсорів, виконує базову логіку та керує модулями зв'язку. Саме на цьому рівні найгостріше постають обмеження, визначені у пункті 1.5.3, зокрема низьке енергоспоживання (для пристроїв з батарейним живленням) та обмежена обчислювальна потужність .

1.6.2 Мережевий рівень (Network Layer)

Цей рівень відповідає за передачу даних, зібраних на рівні пристроїв, до наступних рівнів обробки, а також за доставку команд у зворотному

напрямку – від хмари до пристроїв. Вибір технології зв'язку (підключення), про яке йшлося у пункті 1.2 , є критичним і залежить від вимог проєкту: дальності, швидкості передачі, енергоспоживання та вартості. Сюди відносяться як технології локальних мереж (Wi-Fi, Bluetooth, Zigbee), так і глобальних (LPWAN, LoRaWAN, NB-IoT, 4G/5G). На цьому ж рівні часто присутні *шлюзи (Gateways)*, які виступають мостом між різними протоколами (наприклад, збирають дані з BLE-сенсорів і передають їх у хмару через Wi-Fi).

1.6.3 Рівень граничних обчислень (Edge Processing Layer)

Це відносно новий, але дедалі важливіший рівень архітектури, що знаходиться "на межі" (edge) між локальною мережею пристроїв та глобальною хмарою. Його ключова функція – попередня обробка, фільтрація та агрегація даних до їх відправки у хмару. Замість того, щоб надсилати абсолютно всі "сирі" дані, цей рівень дозволяє приймати рішення локально. Це критично важливо для задоволення нефункціональних вимог (NFRs) , таких як *низька затримка (Performance)* для систем реального часу, *економія трафіку* (що впливає на бюджет) та *автономність* (система може виконувати базові функції навіть за тимчасової втрати з'єднання з Інтернетом). Обчислення на цьому рівні можуть виконуватися або на потужних шлюзах, або на спеціалізованих локальних серверах.

1.6.4 Рівень сервісної підтримки та застосунків (Service and Application Support Layer)

Це, по суті, і є *хмарна платформа (Cloud Layer),* яка є центральним об'єктом нашого дослідження. Цей рівень відповідає за основну обробку, зберігання та аналіз даних, а також надає інструменти для управління всією системою. Саме тут реалізується більшість складних завдань: прийом та

маршрутизація мільйонів повідомлень від пристроїв (зазвичай через MQTT), зберігання даних у спеціалізованих базах даних (наприклад, часових рядів), обробка потоків даних у реальному часі, застосування моделей машинного навчання (ML) та управління життєвим циклом пристроїв (включаючи оновлення "по повітрю" – OTA).

1.6.5 Прикладний рівень (Application Layer)

Верхній рівень архітектури, з яким безпосередньо взаємодіє кінцевий користувач і який надає бізнес-цінність. На цьому рівні відбувається реагування на отриману інформацію. Саме тут втілюються функціональні вимоги (FRs) , визначені на етапі проєктування. Це можуть бути *веб-додатки (Dashboard)* для моніторингу та аналітики (графіки, звіти), *мобільні додатки* для дистанційного керування (наприклад, у "розумному будинку") або *API (Application Programming Interfaces)* для інтеграції IoT-системи з іншими бізнес-системами (ERP, CRM тощо).

1.7 Типи хмарних сервісів для хостингу IoT-рішень

Вибір хмарної інфраструктури, як зазначалося у вступі, є одним з ключових завдань при проєктуванні IoT-рішення. Хмарні технології надають необхідну масштабованість, надійність та потужні інструменти, яких важко досягти при локальному (*on-premise*) розгортанні. Вибір конкретного рішення залежить від двох основних класифікацій: моделі обслуговування та моделі розгортання [1].

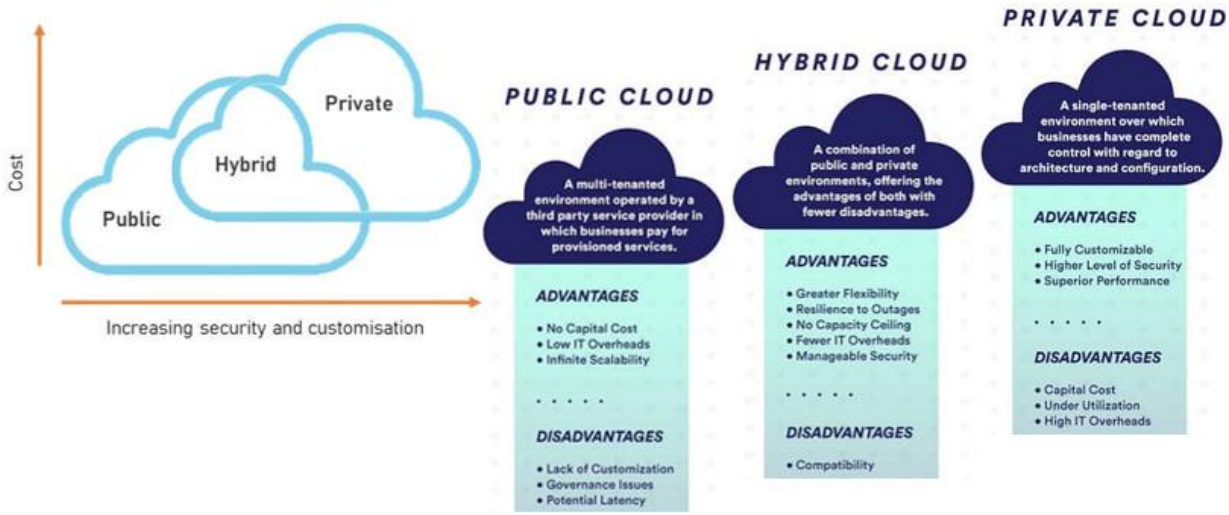

Рисунок 1.5 – Типи хмар та їх характеристики [1]

1.7.1 Моделі обслуговування (IaaS, PaaS, SaaS)

Моделі обслуговування визначають, який рівень управління інфраструктурою провайдер бере на себе, а який залишається у зоні відповідальності розробника.

Інфраструктура як сервіс (IaaS – Infrastructure as a Service) Суть цієї моделі полягає в наданні провайдером базових обчислювальних ресурсів: віртуальних машин (наприклад, AWS EC2, Azure VMs), сховищ (S3, Blob Storage) та мережевих компонентів. У цьому випадку розробник самостійно встановлює та керує операційною системою, проміжним програмним забезпеченням (бази даних, брокери повідомлень) та безпосередньо самим додатком. В контексті IoT це надає максимальну гнучкість. Такий підхід дозволяє розгорнути кастомний стек технологій, наприклад, встановити власні сервери MQTT (як-от EMQX або Mosquitto) на віртуальних машинах. Водночас цей підхід вимагає найбільших витрат часу та найвищої кваліфікації команди (*DevOps*).

Платформа як сервіс (PaaS – Platform as a Service) Тут провайдер надає не лише інфраструктуру, але й кероване проміжне програмне забезпечення. Це включає керовані бази даних (AWS RDS, Azure SQL), сервіси для

виконання коду (AWS Lambda, Azure Functions), та, що найважливіше для IoT, – керовані IoT-платформи (AWS IoT Core, Azure IoT Hub). У зоні відповідальності розробника залишається фокус виключно на написанні бізнес-логіки (коду) та конфігурації сервісів. Розробнику не потрібно перейматися оновленням ОС, масштабуванням баз даних чи адмініструванням брокерів. Це "золота середина" для більшості IoT-проєктів. Сервіси *PaaS* надають готову, масштабовану та безпечну "вхідну точку" для мільйонів пристроїв, що значно прискорює час виведення продукту на ринок (*time-to-market*).

Програмне забезпечення як сервіс (SaaS – Software as a Service) У цій моделі провайдер надає повністю готовий до використання додаток "під ключ". Користувач (або розробник) лише налаштовує додаток через графічний інтерфейс, не маючи доступу до коду чи інфраструктури. До цього типу належать спеціалізовані IoT-платформи, які будуть детально розглянуті в Розділі 2, такі як Blynk, ThingsBoard (у хмарній версії) та ThingSpeak. Вони ідеально підходять для швидкого прототипування, освітніх проєктів або невеликих комерційних рішень, де головна вимога – швидко отримати готовий мобільний додаток та веб-панель без програмування.

1.7.2 Моделі розгортання (Public, Private, Hybrid Cloud)

Моделі розгортання визначають, кому належить інфраструктура та де вона фізично розміщена.

Публічна хмара (Public Cloud) Інфраструктура належить глобальному провайдеру (AWS, Azure, GCP) і надається у користування багатьом клієнтам (*multi-tenanted*) через Інтернет. Це найбільш поширена модель для IoT завдяки низькому порогу входу, моделі оплати за фактом використання (*pay-as-you-go*) та практично нескінченній масштабованості.

Приватна хмара (Private Cloud) Інфраструктура розгорнута виключно для однієї організації (*single-tenanted*). Вона може фізично знаходитись у

власному дата-центрі (*on-premise*) або орендуватися у провайдера. Цю модель обирають організації з надвисокими вимогами до безпеки, конфіденційності даних (наприклад, військові або медичні установи) чи специфічними регуляторними обмеженнями. Вона вимагає значних капітальних інвестицій та кваліфікованого персоналу для підтримки.

Гібридна хмара (Hybrid Cloud) Ця модель є комбінацією публічної та приватної хмар. Це поширена стратегія для IoT-рішень. Наприклад, "чутливі" дані або критичні обчислення реального часу (які вимагають низької затримки, як обговорювалося в 1.6.3) можуть оброблятися у приватній хмарі або на рівні *Edge*. У той же час дані для довгострокового зберігання, аналітики та машинного навчання надсилаються у публічну хмару для використання її потужних та масштабованих сервісів.

1.7.3 Переваги та недоліки різних типів хмар

Вибір моделі обслуговування та розгортання є компромісом, що базується на вимогах та обмеженнях проєкту, які були визначені у пункті 1.5 [1]. Кожен підхід має унікальні переваги та недоліки, які необхідно враховувати при архітектурному проєктуванні.

Таблиця 1.1 – Порівняння моделей хмарних сервісів

Постачальник	Плюси	Мінуси
Веб-сервіси Amazon (AWS)	Багаті сервіси Інтернету речей (IoT Core, Greengrass). Потужна інтеграція аналітики/штучного інтелекту.	Складно для початківців. Вища вартість, якщо не оптимізовано
Microsoft Azure	Потужні інструменти промислового Інтернету речей. Інтеграція з підприємством.	Крута крива навчання. Деякі регіональні обмеження.

Продовження таблиці 1.1

Хмарна платформа Google (GCP)	Чудові інструменти для роботи зі штучним інтелектом/машинним навчанням та обробки даних. Конкурентні ціни.	Відійшов від підтримки нативний IoT Core. Менша екосистема IoT.
Хмара IBM	Потужність промислового та гібридного Інтернету речей. Watson IoT для аналітики	Менша спільнота розробників. Дорого для невеликих компаній.
Oracle Cloud	Відстеження активів та галузеві інструменти. Вбудована аналітика	Не сучасний, не зручний для розробників. Обмежені інструменти для периферійних/IoT пристроїв.

Вибір оптимального поєднання цих моделей (наприклад, використання *PaaS*-сервісів у *публічній* хмарі, або *гібридний* підхід з *PaaS* та *IaaS*) є ключовим архітектурним рішенням, яке напряму впливає на успішність IoT-проєкту.

1.8 Висновки до розділу 1

У першому розділі було проведено комплексний аналіз специфіки рішень Інтернету речей та хмарних технологій, що є фундаментом для подальшого дослідження.

Було проаналізовано еволюцію IoT, яка демонструє логічний перехід від з'єднання комп'ютерів до глобальної мережі взаємопов'язаних фізичних об'єктів, що сьогодні інтегруються зі штучним інтелектом. Сформульовано ключові поняття IoT (*речі, підключення, дані, платформа, аналітика, застосунок*), що дозволило визначити основні компоненти досліджуваних систем [2, 3].

Розглянуто широке коло сфер застосування IoT – від розумних будинків та охорони здоров'я до промисловості (*IIoT*) та розумних міст, що підкреслює універсальність технології та різноманітність вимог до хмарних платформ [1]. Детально описано життєвий цикл IoT-продукту (*проєктування,*

розгортання, управління, виведення з експлуатації), розуміння якого необхідне для вибору платформи, здатної підтримувати рішення на всіх етапах [1].

Систематизовано ключові вимоги до IoT-рішень. Визначено, що вибір інфраструктури має базуватися не лише на *функціональних вимогах* (*FRs*), що система робить, але й на критичних *нефункціональних вимогах* (*NFRs*), як-от масштабованість, надійність, безпека, а також на *обмеженнях* (бюджет, час виходу на ринок) [1, 8].

Запропоновано детальну п'ятирівневу архітектуру IoT-рішень (*пристрої, мережа, граничні обчислення, хмарна платформа, застосунки*). Ця модель чітко ідентифікує місце та роль хмарних сервісів (рівні 4 та 5) у загальній структурі системи [1].

На завершення розділу проведено класифікацію хмарних сервісів (*IaaS, PaaS, SaaS*) та моделей розгортання (*Public, Private, Hybrid*) [1]. Аналіз їхніх переваг та недоліків показав, що для більшості IoT-завдань оптимальним є використання *PaaS*-рішень у *публічних* або *гібридних* хмарах, оскільки це забезпечує найкращий баланс між швидкістю розробки, масштабованістю та вартістю.

Таким чином, перший розділ закладає теоретичну та термінологічну базу. Він обґрунтовує, що вибір хмарної платформи є складним, багатокритеріальним завданням, яке вимагає врахування архітектури, життєвого циклу та специфічних вимог проєкту. Наступний розділ буде присвячений детальному аналізу конкретних пропозицій на ринку хмарних IoT-платформ.

2 АНАЛІЗ РИНКУ ХМАРНИХ СЕРВІСІВ ТА ПЛАТФОРМ ДЛЯ IOT

Після формування теоретичного підґрунтя та визначення ключових архітектурних принципів у першому розділі, наступним логічним кроком є дослідження конкретних хмарних сервісів та платформ, доступних на ринку для розгортання та хостингу IoT-рішень. Цей аналіз є необхідним для збору даних, що ляжуть в основу розробки методики порівняльного оцінювання. Ринок хмарних рішень для IoT характеризується значною різноманітністю пропозицій: від комплексних екосистем глобальних провайдерів до вузькоспеціалізованих платформ.

2.1 Огляд провідних публічних хмарних провайдерів для IoT

Провідні публічні хмарні провайдери, яких часто називають "великою трійкою" (*Amazon Web Services, Microsoft Azure, Google Cloud Platform*), пропонують найпотужніші та найбільш комплексні набори сервісів для побудови IoT-рішень. Окрім них, на ринку присутні й інші значні гравці, такі як IBM Cloud та Oracle Cloud, які також мають власні пропозиції, орієнтовані переважно на корпоративний та промисловий сектори.

Ці провайдери інвестують значні ресурси у розвиток своїх IoT-екосистем, пропонуючи масштабовані сервіси для підключення пристроїв, збору, обробки, зберігання та аналізу даних, а також інтеграцію з інструментами машинного навчання та штучного інтелекту [1].

Amazon Web Services (AWS) є одним із лідерів ринку, пропонуючи надзвичайно багату екосистему IoT-сервісів, центральним з яких є *AWS IoT Core*. Перевагою AWS є сильна інтеграція з іншими сервісами, зокрема з *AWS Greengrass* для граничних обчислень та потужними інструментами для аналітики та ШІ. Водночас, ця широка функціональність може бути складною для початківців, а вартість може бути високою, якщо не приділяти належної уваги оптимізації витрат.

Microsoft Azure робить сильний акцент на промислових IoT-рішеннях (*Industrial IoT*) та інтеграції з корпоративними продуктами Microsoft. Їхні ключові сервіси, *Azure IoT Hub* та *IoT Edge*, є прямими конкурентами AWS. Платформа вважається потужною, але може мати дещо круту криву навчання та певні регіональні обмеження в доступності сервісів.

Google Cloud Platform (GCP) вирізняється своїми провідними інструментами в галузі аналізу великих даних та машинного навчання (*AI/ML*), що робить її привабливою для проєктів, де ці аспекти є ключовими. Ціноутворення часто вважається конкурентним. Однак, суттєвим недоліком стало припинення розвитку нативного сервісу *GCP IoT Core*, що змушує розробників використовувати більш загальні сервіси (наприклад, *Pub/Sub*) і створює враження менш розвиненої IoT-екосистеми порівняно з конкурентами.

IBM Cloud та *Oracle Cloud* також пропонують рішення, орієнтовані на промисловість, гібридні хмари та специфічні завдання, як-от відстеження активів. Проте вони мають меншу спільноту розробників і можуть вважатися дорожчими для малих та середніх проєктів [1].

Для наочного порівняння основних публічних провайдерів, у таблиці 2.1 наведено їхні ключові переваги та недоліки в контексті IoT.

Таблиця 2.1 – Порівняльний огляд публічних хмарних провайдерів для IoT [1]

Провайдер	Плюси	Мінуси
Amazon Web Services (AWS)	• Багаті IoT-сервіси (IoT Core, Greengrass) • Потужна інтеграція аналітики/ШІ	• Складно для початківців • Вища вартість, якщо не оптимізувати
Microsoft Azure	• Потужні інструменти для промислового IoT (Industrial IoT) • Інтеграція з корпоративними системами	• Крута крива навчання • Деякі регіональні обмеження

Продовження таблиці 2.1

Google Cloud Platform (GCP)	• Чудові інструменти для ШІ/МЛ та даних • Конкурентне ціноутворення	• Власний сервіс IoT Core згорнуто • Менша екосистема IoT
IBM Cloud	• Сильні сторони у промисловому та гібридному IoT • Watson IoT для аналітики	• Менша спільнота розробників • Дорого для невеликих рішень
Oracle Cloud	• Інструменти для відстеження активів та промисловості • Вбудована аналітика	• Несучасний/недружній до розробників • Обмежені інструменти для Edge/IoT-пристроїв

Як видно з аналізу, кожен провайдер має свої сильні та слабкі сторони. Вибір конкретної платформи залежить від специфічних вимог проєкту, наявних у команди навичок та бюджетних обмежень. Далі ми детальніше зупинимося на екосистемі лідера ринку – *Amazon Web Services*.

2.2 Amazon Web Services

Як було зазначено в попередньому огляді, *Amazon Web Services* займає провідну позицію на ринку публічних хмарних сервісів для Інтернету речей. Компанія пропонує найбільш зрілу, комплексну та широку екосистему спеціалізованих інструментів, яка дозволяє будувати рішення будь-якої складності – від простих систем моніторингу до глобальних промислових платформ, що обслуговують мільйони пристроїв. Глибока інтеграція IoT-сервісів з іншими потужними інструментами AWS, такими як бази даних, безсерверні обчислення та машинне навчання, робить цю платформу стратегічним вибором для багатьох компаній.

2.2.1 Екосистема сервісів AWS IoT (IoT Core, Greengrass, Analytics та ін.)

Екосистема AWS IoT включає десятки сервісів, призначених для вирішення конкретних завдань на кожному рівні архітектури IoT. Їх можна умовно поділити на сервіси для підключення та управління пристроями, для обробки та аналізу даних, а також для забезпечення безпеки.

Центральним компонентом всієї екосистеми є *AWS IoT Core*. Це керована хмарна служба (*PaaS*), яка виступає в ролі брокера повідомлень (підтримуючи протоколи *MQTT* та *HTTPS*) і дозволяє пристроям легко та безпечно взаємодіяти з хмарними додатками та іншими пристроями. *AWS IoT Core* забезпечує автентифікацію та авторизацію кожного пристрою, а також надає механізм *Device Shadow* (Тінь пристрою). *Device Shadow* – це JSON-документ, що зберігає останній відомий стан пристрою та бажаний майбутній стан. Це дозволяє системі надійно взаємодіяти з пристроями, які можуть тимчасово втрачати зв'язок [1]. Ще одним ключовим елементом *IoT Core* є *Rules Engine* (Механізм правил), який дозволяє фільтрувати, трансформувати та маршрутизувати вхідні повідомлення до інших сервісів AWS (наприклад, *Lambda*, *Amazon S3*, *DynamoDB* або *Kinesis*) для подальшої обробки чи зберігання [1].

Для реалізації граничних обчислень (*Edge Computing*), що було описано в пункті 1.6.3, AWS пропонує сервіс *AWS IoT Greengrass*. Він розширює можливості AWS безпосередньо на кінцеві пристрої або локальні шлюзи. *Greengrass* дозволяє виконувати код (*AWS Lambda*), моделі машинного навчання та контейнери *Docker* локально. Це забезпечує обробку даних ближче до їх джерела, що критично важливо для зменшення затримок, економії трафіку та забезпечення автономної роботи системи при відсутності стабільного інтернет-з'єднання [1].

Для аналізу зібраних даних AWS пропонує набір сервісів. *AWS IoT Analytics* – це керований сервіс, що спрощує запуск та операційне управління

аналітикою для великих обсягів IoT-даних. Він автоматизує етапи очищення, фільтрації, збагачення та зберігання даних, готуючи їх для подальшого аналізу. Для зберігання даних часових рядів (телеметрії) оптимізовано використання сервісу *Amazon Timestream*. Для обробки даних у реальному часі часто використовується *Amazon Kinesis*, що дозволяє аналізувати потоки даних до їх потрапляння у сховище.

Управління життєвим циклом пристроїв та забезпечення безпеки реалізується через *AWS IoT Device Management* та *AWS IoT Device Defender*. Перший сервіс дозволяє реєструвати, організовувати, моніторити та віддалено керувати парком пристроїв, включаючи розгортання оновлень програмного забезпечення "по повітрю" (*OTA*). *Device Defender* забезпечує безперервний аудит конфігурацій безпеки, моніторинг аномальної поведінки пристроїв та надсилання сповіщень у разі виявлення загроз [1].

Ці сервіси є лише частиною великої екосистеми, яка також включає сервіси для зберігання (*Amazon S3*), підготовки даних (*AWS Glue*) та виконання коду (*AWS Lambda*), формуючи гнучкий набір інструментів для побудови IoT-рішень.

Таблиця 2.2 – Ключові сервіси AWS для IoT [1]

Сервіс	Функція	Орієнтовна вартість
AWS IoT Core	Безпечне підключення та масштабоване управління IoT-пристроями.	Підключення: $0.08 за мільйон хвилин з'єднання. Повідомлення: $1 за мільйон повідомлень.
AWS IoT Greengrass	Запуск локальних обчислень, обміну повідомленнями та ML на edge-пристроях.	$0.16 за активний пристрій Greengrass Core на місяць.
AWS IoT Analytics	Аналіз, фільтрація та збагачення IoT-даних для отримання інсайтів.	Вартість залежить від обробки та зберігання даних; див. Калькулятор цін AWS.

Продовження таблиці 2.2

AWS IoT Device Defender	Моніторинг та захист парку пристроїв за допомогою аудиту та сповіщень.	Вартість базується на перевірках аудиту та метриках; див. Калькулятор цін AWS.
AWS IoT Device Management	Реєстрація, організація, моніторинг та віддалене управління пристроями.	Пряме підключення: $0.01 за пристрій на місяць. Хмара-до-хмари: $0.02 за пристрій на місяць.
AWS Lambda	Запуск коду на основі подій без налаштування серверів.	$0.20 за мільйон запитів; додаткова плата за час обчислень.
Amazon Timestream	База даних часових рядів, оптимізована для телеметричних даних IoT.	Вартість базується на прийомі, зберіганні та запитах даних; див. Калькулятор цін AWS.
Amazon Kinesis	Прийом, буферизація та обробка потоків даних у реальному часі.	Вартість базується на пропускній здатності та зберіганні даних; див. Калькулятор цін AWS.
Amazon S3	Надійне, масштабоване сховище для IoT-даних.	Стандартне сховище: $0.023 за ГБ на місяць.
AWS Glue	Підготовка та трансформація IoT-даних для аналітики та звітності.	Вартість базується на одиницях обробки даних (DPU) та тривалості завдання; див. Калькулятор цін AWS.

2.2.2 AWS IoT Lens: Well-Architected Framework

При побудові рішень на базі такої великої екосистеми сервісів, як AWS, розробники та архітектори стикаються з викликами щодо забезпечення надійності, безпеки, ефективності та економічності своїх рішень. Для вирішення цих завдань AWS розробив *Well-Architected Framework* – набір найкращих практик та проєктних принципів для побудови високоякісних систем у хмарі.

Для специфічних завдань Інтернету речей існує спеціалізоване розширення – *AWS IoT Lens* [7]. Цей документ надає конкретні рекомендації та запитання для оцінки архітектури IoT-рішень, допомагаючи переконатися, що система спроєктована з урахуванням усіх ключових аспектів. *IoT Lens*

базується на шести стовпах (*pillars*), які охоплюють повний життєвий цикл рішення [1]:

1. *Операційна досконалість (Operational Excellence)*. Цей стовп фокусується на здатності системи підтримувати робочі процеси та безперервно вдосконалюватися. В контексті IoT це включає автоматизацію розгортання пристроїв та хмарних компонентів, ефективний моніторинг стану парку пристроїв та управління оновленнями програмного забезпечення.
2. *Безпека (Security)*. Забезпечення безпеки є критично важливим для IoT-систем через велику кількість фізичних пристроїв. Цей стовп охоплює примусове впровадження надійного управління ідентифікацією та доступом (*IAM*) для пристроїв і користувачів, захист даних під час передачі (шифрування, *mutual TLS*) та у стані спокою, а також використання сертифікатів для автентифікації пристроїв [1, 8].
3. *Надійність (Reliability)*. Системи IoT мають бути стійкими до збоїв, особливо враховуючи нестабільність мережевих підключень для багатьох пристроїв. Цей принцип вимагає проєктування з урахуванням можливих проблем зі зв'язком (переривчасті або слабкі мережі), впровадження логіки повторних спроб (*retry logic*), буферизації та черг повідомлень.
4. *Ефективність продуктивності (Performance Efficiency)*. Цей стовп стосується оптимального використання обчислювальних ресурсів. Для IoT це часто означає використання подієво-керованої архітектури та безсерверних технологій (наприклад, *AWS Lambda*). Також сюди входить оптимізація протоколів зв'язку (*MQTT*) та обробка даних якомога ближче до джерела за допомогою граничних обчислень (*AWS IoT Greengrass*).
5. *Оптимізація витрат (Cost Optimization)*. Ефективне управління витратами є ключовим для рентабельності IoT-проєктів. Цей принцип передбачає використання моделі оплати за фактом використання (*pay-as-you-go*), зменшення витрат на передачу та зберігання даних шляхом їх фільтрації та агрегації на рівні *Edge*, а також використання багаторівневих сховищ і політик життєвого циклу даних [1].

6. *Стійкість (Sustainability)*. Цей, відносно новий, стовп фокусується на мінімізації впливу рішення на навколишнє середовище. В IoT це означає зменшення споживання енергії як на рівні пристроїв, так і в хмарі, підвищення ефективності використання ресурсів та мінімізацію загального обсягу необхідного обладнання.

Використання *AWS IoT Lens* як керівного принципу дозволяє архітекторам приймати обґрунтовані рішення на етапі проєктування, що значно знижує ризики та підвищує загальну якість кінцевого IoT-рішення.

2.2.3 Моделі ціноутворення та оптимізація витрат (FinOps в AWS)

Одним із ключових аспектів, що визначають вибір хмарної платформи, є її вартість. Моделі ціноутворення *Amazon Web Services* базуються на принципі *pay-as-you-go* (оплата за фактом використання), що є фундаментальною перевагою публічних хмар. Для сервісів IoT це означає, що вартість розраховується на основі конкретних показників: кількості підключень, хвилин з'єднання, кількості надісланих та отриманих повідомлень, обсягу оброблених даних, часу обчислень *Lambda* та гігабайт збережених даних [1].

Хоча така модель виглядає гнучкою, вона несе в собі значну складність при прогнозуванні витрат, особливо для масштабних IoT-рішень з мільйонами пристроїв, що генерують безперервні потоки телеметрії. Неправильне проєктування, наприклад, надсилання "сирих" даних замість агрегованих, може призвести до неконтрольованого зростання витрат. Для попереднього розрахунку вартості AWS надає інструмент *AWS Pricing Calculator* [9], який дозволяє змоделювати очікуване навантаження та оцінити щомісячні витрати.

Через цю складність, у сучасних хмарних практиках виникла окрема дисципліна – *FinOps* (скорочення від *Finance* та *DevOps*). *FinOps* – це операційна та культурна практика, яка об'єднує інженерні, фінансові та

бізнес-команди з метою максимізації бізнес-цінності хмарних технологій та забезпечення фінансової підзвітності [1]. Замість того, щоб просто "платити за рахунками", *FinOps* пропонує ітеративний підхід до управління витратами, який складається з трьох фаз:

1. *Інформування (Inform)*. Це початкова фаза, метою якої є забезпечення повної прозорості та видимості витрат. Команди мають розуміти, які сервіси скільки споживають та з якими бізнес-завданнями це пов'язано. Це досягається шляхом точного маркування (тегування) ресурсів, налаштування панелей моніторингу витрат та бенчмаркінгу.
2. *Оптимізація (Optimize)*. На цій фазі команди вживають заходів для підвищення ефективності витрат. В контексті IoT це може включати: вибір правильних моделей ціноутворення (наприклад, резервування потужностей – *Reserved Instances*), оптимізацію передачі даних (фільтрація та агрегація на рівні *Edge*), автоматичне масштабування ресурсів (*serverless*) та виявлення і видалення невикористаних або надлишкових ресурсів.
3. *Управління (Operate)*. Ця фаза фокусується на безперервному узгодженні хмарних операцій з бізнес-цілями. Команди відстежують динаміку витрат, оцінюють відповідність бюджету та приймають оперативні рішення, керуючись даними, отриманими на перших двох фазах. Це створює культуру, де кожен розробник та архітектор враховує вартість своїх технічних рішень.

Застосування практик *FinOps* є невід'ємною частиною експлуатації IoT-рішень на AWS, дозволяючи компаніям використовувати потужність платформи, не стаючи заручником непередбачуваних витрат, що було також підкреслено в *AWS Well-Architected Framework* [7] як один із ключових стовпів – *Cost Optimization*.

2.2.4 Приклад референтної архітектури IoT-рішення на AWS

Для повного розуміння, як окремі сервіси AWS, розглянуті в попередніх пунктах, поєднуються у єдине рішення, доцільно проаналізувати приклад референтної архітектури. Така архітектура демонструє типовий потік даних та взаємодію компонентів для побудови масштабованого та функціонального IoT-застосунку [1].

На рисунку 2.1 представлена комплексна архітектура, що використовує підхід *serverless* (безсерверні обчислення) та керовані *PaaS*-сервіси.

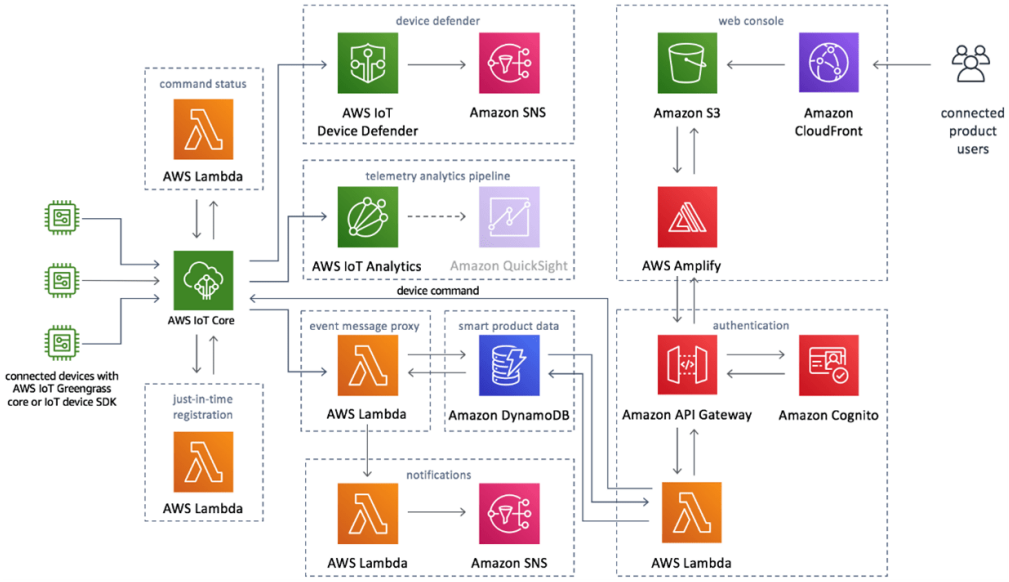

Рисунок 2.1 – Приклад референтної архітектури IoT-рішення на AWS [1]

Розглянемо ключові потоки даних у цій архітектурі:

1. *Підключення та прийом даних*. Кінцеві пристрої (сенсори), включаючи ті, що працюють під управлінням *AWS IoT Greengrass*, безпечно підключаються до *AWS IoT Core*. *IoT Core* виступає як центральний брокер повідомлень, що приймає всю телеметрію. Архітектура також передбачає механізм *just-in-time registration* (реєстрація "в останній момент") з використанням функції *AWS Lambda* для автоматичної реєстрації нових пристроїв.

2. *Обробка та зберігання*. *Rules Engine* (механізм правил) сервісу *IoT Core* аналізує вхідні повідомлення та маршрутизує їх. Наприклад, потік *smart product data* (дані про стан пристрою) через *Lambda* зберігається в базі даних *Amazon DynamoDB* (високопродуктивна *NoSQL* база даних).
3. *Аналітика*. Інший потік даних (*telemetry analytics pipeline*) спрямовується до *AWS IoT Analytics* для фільтрації, збагачення та обробки. Результати аналізу візуалізуються для бізнес-користувачів за допомогою сервісу *Amazon QuickSight*.
4. *Безпека та сповіщення*. *AWS IoT Device Defender* постійно моніторить систему на предмет аномалій. У разі виявлення загрози, він може автоматично надіслати сповіщення (наприклад, адміністратору) через сервіс *Amazon SNS* (Simple Notification Service).
5. *Користувацький застосунок (Frontend)*. Веб-консоль (панель моніторингу) для кінцевих користувачів реалізована як статичний веб-сайт, файли якого зберігаються в *Amazon S3* та розповсюджуються глобально через мережу доставки контенту *Amazon CloudFront* для низької затримки. Фреймворк *AWS Amplify* об'єднує ці компоненти.
6. *Серверна логіка (Backend)*. Для взаємодії веб-консолі з хмарними сервісами використовується *Amazon API Gateway*, який надає захищений *REST API*. Запити від користувача (наприклад, "відправити команду пристрою") обробляються функціями *AWS Lambda*. Автентифікація та управління користувачами веб-консолі реалізовані через *Amazon Cognito*.

Ця референтна архітектура демонструє, як *PaaS*-сервіси AWS дозволяють побудувати складне, надійне та безпечне IoT-рішення, мінімізуючи необхідність в адмініструванні інфраструктури та фокусуючись на бізнес-логіці застосунку.

2.3 Microsoft Azure

Microsoft Azure є другим за часткою ринку гравцем серед публічних хмарних провайдерів і виступає головним конкурентом AWS у корпоративному сегменті. Екосистема Azure для Інтернету речей є надзвичайно потужною та комплексною, маючи чіткий фокус на інтеграції з існуючими бізнес-продуктами Microsoft (такими як *Dynamics 365* та *Power BI*), а також на задоволенні потреб промислового сектору [1].

Як і AWS, Azure надає повний набір *PaaS*-сервісів, що дозволяють розробникам будувати, розгортати та керувати IoT-рішеннями, абстрагуючись від базової інфраструктури.

2.3.1 Ключові сервіси (Azure IoT Hub, IoT Central, IoT Edge)

Екосистема IoT-сервісів Microsoft побудована навколо трьох ключових компонентів, що надають різні рівні абстракції та контролю:

- *Azure IoT Hub*. Це керований хмарний шлюз, який є прямим аналогом *AWS IoT Core*. Він виступає як центральний брокер повідомлень для двонапрямленого зв'язку між мільйонами IoT-пристроїв та хмарним бекендом. *IoT Hub* забезпечує безпечне підключення, автентифікацію на рівні кожного пристрою, прийом телеметрії у великих масштабах та маршрутизацію повідомлень до інших сервісів Azure (наприклад, *Azure Functions* або *Stream Analytics*). Він також підтримує механізм *Device Twins* (двійники пристроїв), аналогічний *Device Shadow* в AWS, для зберігання стану пристроїв.
- *Azure IoT Central*. Це сервіс, що знаходиться на вищому рівні абстракції і фактично є *SaaS*-рішенням (або *aPaaS – application Platform as a Service*). *IoT Central* надає розробникам готовий, майже повністю налаштований бекенд для IoT-рішень. Замість того, щоб збирати рішення з окремих компонентів (*IoT Hub*, бази даних, візуалізація), *IoT Central* пропонує єдину платформу з

готовими шаблонами, веб-інтерфейсом для управління та візуалізації. Це значно прискорює прототипування та розробку, але надає менше гнучкості порівняно з підходом на базі *IoT Hub*.

- *Azure IoT Edge*. Аналогічно до *AWS IoT Greengrass, Azure IoT Edge* є сервісом для реалізації граничних обчислень (*Edge Computing*). Він дозволяє розгортати та виконувати робочі навантаження, такі як моделі ШІ, сервіси Azure або власну бізнес-логіку, безпосередньо на IoT-пристроях у вигляді *контейнерів*. Це забезпечує можливість роботи в режимі офлайн, швидку реакцію (низьку затримку) та зменшення витрат на хмарний трафік.

2.3.2 Специфіка застосування в промисловості (Industrial IoT)

Особливою перевагою платформи Azure, як зазначено в порівняльному аналізі [1], є її сильна орієнтація на промисловий Інтернет речей (*Industrial IoT* або *IIoT*). Microsoft активно розвиває цей напрямок, пропонуючи не лише базові сервіси, але й комплексні рішення для цифрової трансформації виробництва.

Специфіка полягає у глибокій інтеграції з промисловими стандартами та протоколами. Наприклад, Azure підтримує *OPC UA* (*Open Platform Communications Unified Architecture*) – поширений стандарт для безпечного та надійного обміну даними у промисловій автоматизації. Це дозволяє легко підключати до хмари існуюче заводське обладнання (наприклад, *PLC* та *SCADA*-системи), що раніше працювало в ізольованих мережах.

Крім того, Azure пропонує рішення для створення "цифрових двійників" (*Azure Digital Twins*) не просто окремих пристроїв, а цілих середовищ – фабрик, будівель або енергомереж. Це дозволяє моделювати складні взаємозв'язки між процесами та обладнанням, оптимізувати операції та впроваджувати предиктивне обслуговування, що є ключовим завданням для *IIoT*.

2.4 Google Cloud Platform (GCP)

Google Cloud Platform (GCP), третій ключовий гравець на ринку публічних хмар, історично займав дещо іншу позицію в контексті Інтернету речей. На відміну від AWS та Azure, які пропонували комплексні та чітко брендовані IoT-пакети, сильні сторони GCP завжди полягали у неперевершених можливостях обробки великих даних, аналітики та машинного навчання (ML). Це робило платформу привабливим вибором для тих IoT-проєктів, де кінцевою метою була не просто збір телеметрії, а її глибокий інтелектуальний аналіз.

2.4.1 Поточні сервіси для IoT (Pub/Sub, Dataflow)

Важливим фактором, що вплинув на сприйняття GCP в IoT-спільноті, стало рішення компанії про припинення роботи (retirement) свого спеціалізованого керованого сервісу *Google Cloud IoT Core* у 2023 році. Це рішення змусило існуючих та нових користувачів переходити на використання більш загальних, неспеціалізованих сервісів платформи для побудови IoT-рішень.

На сьогодні архітектура IoT-рішень на GCP зазвичай будується на наступних компонентах:

- *Cloud Pub/Sub*: Це глобальний, керований сервіс для асинхронного обміну повідомленнями. В архітектурі IoT він виконує роль основного "вхідного шлюзу" для прийому телеметрії від пристроїв, замінюючи собою брокер *MQTT*, який був інтегрований в *IoT Core*. Пристрої можуть публікувати дані в теми *Pub/Sub* напряму через *HTTPS* або через брокери *MQTT* на рівні *Edge*.
- *Cloud Dataflow*: Це повністю керований сервіс для обробки потокових (stream) та пакетних (batch) даних. *Dataflow* використовується для обробки, трансформації та аналізу потоків телеметрії, що надходять з *Pub/Sub*, в режимі реального часу, перш ніж вони будуть завантажені у сховища даних.

- *Cloud Functions*: Безсерверний сервіс (*FaaS*), аналогічний *AWS Lambda*, який використовується для запуску коду у відповідь на події (наприклад, надходження нового повідомлення в *Pub/Sub*).

Хоча цей підхід "конструктора" надає гнучкість, він вимагає від розробників більшої архітектурної роботи порівняно з використанням готових *PaaS*-рішень, як-от *AWS IoT Core* чи *Azure IoT Hub*.

2.4.2 Сильні сторони в аналітиці та ML

Незважаючи на відсутність єдиного IoT-пакету, головною конкурентною перевагою GCP залишається його екосистема аналітики та штучного інтелекту. Для проєктів, де основний фокус – це не управління мільйонами пристроїв, а глибокий аналіз зібраних даних, GCP пропонує найкращі в своєму класі інструменти.

Сервіси, як-от *BigQuery* (високомасштабоване хмарне сховище даних з вбудованими можливостями ML), *Vertex AI* (єдина платформа для створення та управління моделями машинного навчання) та *Looker Studio* (для візуалізації даних), дозволяють реалізовувати найскладніші сценарії аналізу. Наприклад, побудова моделей предиктивного обслуговування, аналіз поведінкових патернів або оптимізація логістичних ланцюгів на основі IoT-даних.

Таким чином, GCP залишається актуальним вибором для IoT-проєктів, керованих даними (*data-driven*), де команда розробників готова інвестувати час у побудову власної архітектури прийому даних заради доступу до провідних аналітичних інструментів.

2.5 Порівняльний аналіз пропозицій "великої трійки"

Після детального розгляду пропозицій *Amazon Web Services*, *Microsoft Azure* та *Google Cloud Platform* стає очевидним, що кожен із цих провайдерів

пропонує потужну, але ідеологічно різну екосистему для побудови рішень Інтернету речей. Порівняльний аналіз "великої трійки" (зведений у Таблиці 2.1) виявляє ключові відмінності у їхніх стратегіях, сильних сторонах та цільових ринках.

AWS та *Azure* виступають як найбільш прямі та повнофункціональні конкуренти. Обидві платформи пропонують зрілі *PaaS*-рішення для повного циклу IoT: від безпечного підключення пристроїв (*AWS IoT Core* та *Azure IoT Hub*) до розгортання логіки на граничних пристроях (*AWS IoT Greengrass* та *Azure IoT Edge*). Їхні екосистеми є найбільш комплексними, надаючи керовані сервіси практично для будь-якого завдання. Ключова відмінність полягає у фокусі: AWS вирізняється величезною широтою інтегрованих сервісів та гнучкістю, тоді як Azure робить явний акцент на потреби промисловості (*IIoT*) та безшовну інтеграцію з корпоративним програмним забезпеченням Microsoft [1]. Вибір між ними часто зводиться до наявних у компанії компетенцій або специфічних вимог промислової інтеграції.

Google Cloud Platform займає унікальну позицію. Після припинення розвитку *IoT Core*, GCP фактично відійшов від моделі єдиного керованого IoT-хабу, яку пропонують конкуренти. Замість цього, GCP пропонує архітектуру "конструктора" на базі загальних сервісів (*Pub/Sub*, *Dataflow*). Це робить поріг входу для простих IoT-завдань вищим і створює враження менш зрілої IoT-екосистеми [1]. Однак, ця слабкість компенсується провідними на ринку можливостями в аналізі даних та машинному навчанні (*BigQuery*, *Vertex AI*). Тому GCP залишається надзвичайно потужним вибором для проєктів, де основний фокус припадає не на управління пристроями, а на складну інтелектуальну обробку зібраних даних.

Підсумовуючи, не існує єдиного "найкращого" провайдера. *AWS* пропонує найбільш універсальну та багату екосистему. *Azure* є лідером для завдань промислової автоматизації та інтеграції. *GCP* є оптимальним вибором для проєктів, що вимагають поглибленої аналітики та *ML*-обробки даних. Цей аналіз підтверджує, що вибір платформи є компромісним

рішенням, яке має базуватися на конкретних вимогах та пріоритетах проєкту, що буде детально розглянуто в Розділі 3.

2.6 Огляд спеціалізованих платформних сервісів для IoT

Окрім комплексних екосистем від "великої трійки" (AWS, Azure, GCP), існує широкий ринок спеціалізованих IoT-платформ. Ці сервіси, які часто функціонують за моделлю *SaaS* або *PaaS*, орієнтовані на значне спрощення та прискорення процесу розробки IoT-рішень. Вони пропонують готові інструменти для підключення пристроїв, візуалізації даних (веб-панелі) та створення мобільних додатків, часто з мінімальним програмуванням або взагалі без нього (*no-code/low-code*).

Такі платформи є особливо привабливими для швидкого прототипування, освітніх проєктів, малого бізнесу або для розробників, які хочуть зосередитися на апаратній частині та бізнес-логіці, не заглиблюючись у складність адміністрування хмарної інфраструктури [1]. Розглянемо декілька ключових гравців на цьому ринку.

2.6.1 Blynk (хмарна та локальна версії)

Blynk – це одна з найпопулярніших платформ, яка здобула визнання завдяки своїй винятковій простоті та орієнтації на розробників і мейкерів. Ключовою особливістю *Blynk* є конструктор мобільних додатків (*app builder*) – інструмент, що дозволяє методом *drag-and-drop* створювати повноцінні нативні додатки для *iOS* та *Android* без написання мобільного коду [10, 11].

Платформа складається з трьох основних компонентів:

1. Blynk App: Мобільний додаток-конструктор, де користувач збирає інтерфейс із готових віджетів (кнопки, слайдери, графіки, індикатори).
2. Blynk Server: Бекенд, що відповідає за обробку комунікацій між мобільним додатком та апаратним забезпеченням.

3. Blynk Libraries: Набір бібліотек для популярних мікроконтролерів (як-от ESP32, ESP8266, Arduino), які максимально спрощують процес підключення пристрою до сервера.

Blynk пропонує дві моделі розгортання:

- *Хмарна версія (Blynk.Cloud)*: Це *SaaS*-рішення, де користувачі платять за підписку залежно від кількості пристроїв та необхідного функціоналу. Цей варіант забезпечує найшвидший старт, оскільки не вимагає жодного адміністрування сервера.
- *Локальна версія (Blynk.Server)*: Можливість розгорнути власний, *open-source* сервер *Blynk* на своєму обладнанні (наприклад, на *Raspberry Pi* або у приватній хмарі *IaaS*). Цей варіант є безкоштовним та надає повний контроль над даними та системою, але вимагає технічних навичок для налаштування та підтримки сервера.

Завдяки такій гнучкості та низькому порогу входу, *Blynk* ідеально підходить для швидкого прототипування, "розумних будинків" та освітніх проєктів. Водночас, як зазначається в аналізі [1], платні тарифи можуть бути обмежуючими, а вбудовані аналітичні можливості є досить базовими порівняно з великими хмарними провайдерами.

2.6.2 ThingSpeak

ThingSpeak – це ще одна популярна спеціалізована IoT-платформа, яка займає унікальну нішу завдяки своїй тісній інтеграції з *MATLAB*, потужним середовищем для математичних обчислень та аналізу даних, розробленим компанією *MathWorks*. Платформа позиціонується як сервіс для збору, аналізу та візуалізації даних, що надходять від сенсорів [1].

Основна концепція *ThingSpeak* базується на "каналах" (*Channels*), які є сховищами для даних, що надсилаються пристроями. Кожен канал може мати до восьми полів для зберігання різних показників (наприклад, температура,

вологість, тиск). Пристрої, як-от *ESP8266* або *Arduino*, можуть надсилати дані на платформу за допомогою простих *HTTP API* запитів.

Ключовою перевагою *ThingSpeak* є його аналітичні можливості. Платформа дозволяє не просто зберігати дані, але й обробляти їх безпосередньо в хмарі за допомогою скриптів *MATLAB*. Користувачі можуть писати власний код для виконання аналізу даних, їх трансформації, або для реагування на певні події (наприклад, надсилання сповіщень). Ця інтеграція робить *ThingSpeak* ідеальним інструментом для освітніх цілей, наукових досліджень та прототипування проєктів, де потрібен складний математичний аналіз даних [1].

Платформа пропонує безкоштовний рівень (free tier), що робить її доступною для студентів та ентузіастів. Водночас, *ThingSpeak* має суттєві обмеження: його користувацький інтерфейс для візуалізації є досить базовим, а сама платформа не розрахована на великомасштабні комерційні рішення з тисячами пристроїв, поступаючись у масштабованості та гнучкості як великим хмарам, так і іншим спеціалізованим платформам на кшталт *ThingsBoard* [1].

2.6.3 ThingsBoard

ThingsBoard – це потужна *open-source* платформа, призначена для збору, обробки, візуалізації та управління пристроями в IoT-рішеннях. На відміну від *Blynk* чи *ThingSpeak, ThingsBoard* одразу позиціонується як масштабоване та гнучке рішення корпоративного рівня, що забезпечує значно ширшу функціональність [1].

Платформа надає інструменти для вирішення широкого кола завдань:

- *Управління активами та пристроями*: Дозволяє створювати моделі об'єктів (пристроїв, активів) та їх зв'язків.
- *Збір даних*: Нативно підтримує стандартні IoT-протоколи, такі як *MQTT, CoAP* та *HTTP*, для прийому телеметрії.

- *Обробка та аналітика*: Має потужний механізм обробки подій (*Rule Engine*) з графічним інтерфейсом, що дозволяє створювати складні ланцюжки обробки даних (фільтрація, трансформація, аналіз, генерація тригерів) без написання коду.
- *Візуалізація*: Надає гнучкий конструктор веб-панелей (*dashboards*) з великим набором віджетів для візуалізації даних у реальному часі та відображення історії.

Як і Blynk, ThingsBoard пропонує дві основні моделі розгортання:

1. *ThingsBoard Community Edition (CE)*: Це *open-source* версія, яку можна безкоштовно розгорнути на власних серверах (у приватній хмарі *IaaS* або *on-premise*). Цей варіант надає повний контроль над системою та даними і є надзвичайно гнучким, але вимагає значних технічних знань для інсталяції, налаштування, масштабування та подальшої підтримки [1].
2. *ThingsBoard Professional Edition (PE) / Cloud*: Комерційна версія, яка доступна як у вигляді *SaaS*-рішення (хмарна підписка), так і у вигляді ліцензії для розгортання *on-premise*. Вона включає додаткові функції, преміум-віджети та професійну технічну підтримку.

Завдяки своїй масштабованості та високому ступеню кастомізації, *ThingsBoard* добре підходить для побудови складних професійних IoT-рішень, систем моніторингу та управління. Однак, його початкове налаштування та адміністрування є значно складнішим порівняно з простішими *SaaS*-платформами, що підвищує вимоги до кваліфікації команди [1].

2.6.4 Інші платформи (Tuya, Akenza)

Окрім *Blynk, ThingSpeak* та *ThingsBoard*, ринок пропонує й інші платформи, кожна з яких має свою специфічну нішу та переваги.

Tuya є глобальною платформою, що зробила основний акцент на ринку "розумного будинку" (*Smart Home*). Її бізнес-модель орієнтована на *OEM*

(Original Equipment Manufacturer) виробників. *Tuya* надає виробникам готовий хмарний бекенд, модулі зв'язку та інструменти для швидкого створення брендованих мобільних додатків. Це дозволяє компаніям випускати на ринок "розумні" пристрої (лампочки, розетки, камери) з мінімальними витратами на розробку власного ПЗ. Для кінцевого користувача це виливається у величезну екосистему сумісних пристроїв. Однак, суттєвим недоліком цієї платформи є її закритість: розробники переважно обмежені використанням пристроїв з екосистеми *Tuya* і мають менше можливостей для глибокої кастомізації порівняно з *open-source* рішеннями [1].

Akenza – це платформа, що фокусується на підході *low-code* / *no-code* (без або з мінімальним програмуванням) для корпоративних клієнтів. Вона позиціонується як універсальний інструмент для підключення пристроїв, що використовують різні технології зв'язку, з особливим акцентом на *LPWAN* (зокрема *LoRaWAN*) та стільникові технології (*NB-IoT*). Платформа надає гарні інструменти для візуалізації даних та управління парком пристроїв. Водночас, *Akenza* має меншу спільноту розробників порівняно з іншими платформами, а її вартість може бути вищою для невеликих проєктів чи стартапів [1].

2.6.5 Порівняльний аналіз спеціалізованих платформ

Огляд *Blynk, ThingSpeak, ThingsBoard* та інших платформ демонструє, що, на відміну від універсальних екосистем "великої трійки", ринок спеціалізованих сервісів є сильно сегментованим. Кожна платформа має чітку цільову аудиторію та пропонує свій унікальний набір переваг та компромісів.

Blynk та *Tuya* є лідерами у сегменті "розумного будинку" та швидкого прототипування, пропонуючи готові інструменти для створення мобільних додатків, але мають обмежену вбудовану аналітику. *ThingSpeak* міцно закріпився в освітній та науковій ніші завдяки інтеграції з *MATLAB*, проте

його інтерфейс та масштабованість не призначені для комерційних продуктів. *ThingsBoard* , навпаки, є *open-source* рішенням, що орієнтоване на гнучкість та масштабованість, але вимагає значно вищої технічної кваліфікації для розгортання та підтримки порівняно з іншими.

Для узагальнення ключових відмінностей, у таблиці 2.3 наведено порівняльний аналіз розглянутих спеціалізованих платформ.

Таблиця 2.3 – Порівняльний огляд спеціалізованих IoT-платформ [1]

Платформа	Плюси	Мінуси
Blynk	• Зручний конструктор додатків (app builder) • Хмарне та приватне розгортання • Широка підтримка обладнання	• Обмежена аналітика • Деякі функції потребують платних тарифів
ThingSpeak	• Інтеграція з MATLAB • Ідеально для освіти/досліджень • Доступний безкоштовний тариф	• Не підходить для великих масштабів • Базовий інтерфейс користувача
Tuya	• Екосистема розумного будинку • Швидке налаштування для OEM-виробників • Управління пристроями через хмару	• Обмежено пристроями Tuya • Менше можливостей кастомізації
Akenza	• Платформа без коду (No-code) • Підтримує LoRaWAN/HTTP • Чудові інструменти візуалізації	• Вища вартість для невеликих рішень • Менша спільнота
ThingsBoard	• Open-source та масштабована • Підтримка протоколів MQTT, CoAP • Високі можливості кастомізації	• Вимагає технічних знань • Налаштування може бути складним

Цей аналіз підкреслює, що вибір спеціалізованої платформи залежить від балансу між швидкістю розробки, необхідним рівнем кастомізації, вимогами до аналітики та наявними у команди технічними навичками.

2.7 Практичні приклади реалізації IoT-рішень

Для кращого розуміння практичних аспектів впровадження IoT-рішень, переваг та недоліків різних підходів, було детально проаналізовано два

реалізовані проєкти. Обидва проєкти мали на меті вирішення однакового завдання – моніторинг температури та вологості – але використовували кардинально різні підходи до хостингу та обробки даних:

1. Використання готової спеціалізованої *PaaS/SaaS* платформи [10, 11, 12].
2. Використання *кастомного веб-хостингу* [14, 15].

2.7.1 Рішення для моніторингу температури на базі платформи Blynk

Перший практичний приклад демонструє швидкість та простоту розгортання IoT-рішення з використанням платформи *Blynk*, яка була розглянута в пункті 2.6.1. Проєкт базувався на використанні поширених апаратних компонентів: мікроконтролера *ESP8266* (з вбудованим Wi-Fi) та сенсора температури і вологості *DHT11* [11, 12].

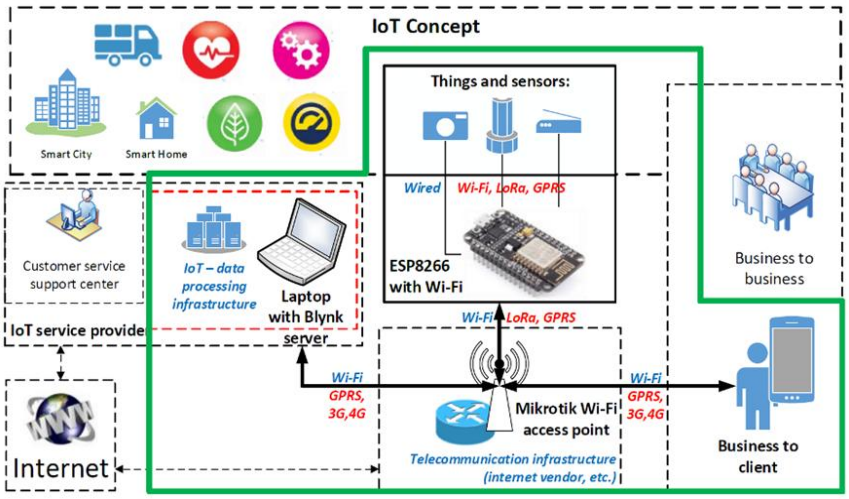

Рисунок 2.2 – Архітектура IoT-рішення на базі Blynk [1]

Процес реалізації був максимально спрощений завдяки інструментам платформи [10]:

1. *Налаштування сервера*: Було обрано хмарну версію *Blynk.Cloud*, що виключило необхідність в адмініструванні сервера. У мобільному додатку

Blynk було створено новий проєкт та отримано унікальний токен автентифікації.

2. *Програмування мікроконтролера*: На *ESP8266* було завантажено код з використанням бібліотеки *Blynk*. Код зводився до ініціалізації підключення до Wi-Fi та сервера *Blynk* (з використанням отриманого токена) та періодичного зчитування даних з сенсора *DHT11*. Дані надсилалися на сервер за допомогою однієї команди Blynk.virtualWrite().
3. *Створення інтерфейсу*: Візуальний інтерфейс користувача (мобільний додаток) було створено за декілька хвилин у конструкторі *Blynk* шляхом простого перетягування необхідних віджетів (графіки, індикатори) на робочу область.

Цей приклад доводить, що використання готових *PaaS/SaaS* платформ, як-от *Blynk*, дозволяє отримати робочий прототип системи моніторингу за надзвичайно короткий час, не вимагаючи глибоких знань у мобільній чи веб-розробці [10].

2.7.2 Рішення для моніторингу температури на базі кастомного веб-хостингу

Другий практичний приклад ілюструє можливості та складності створення власного, незалежного IoT-рішення з використанням стандартного *веб-хостингу*, бази даних *MySQL* та мови сценаріїв *PHP* [14, 15]. Апаратна частина була ідентичною першому прикладу (мікроконтролер *ESP8266* та сенсор *DHT11*).

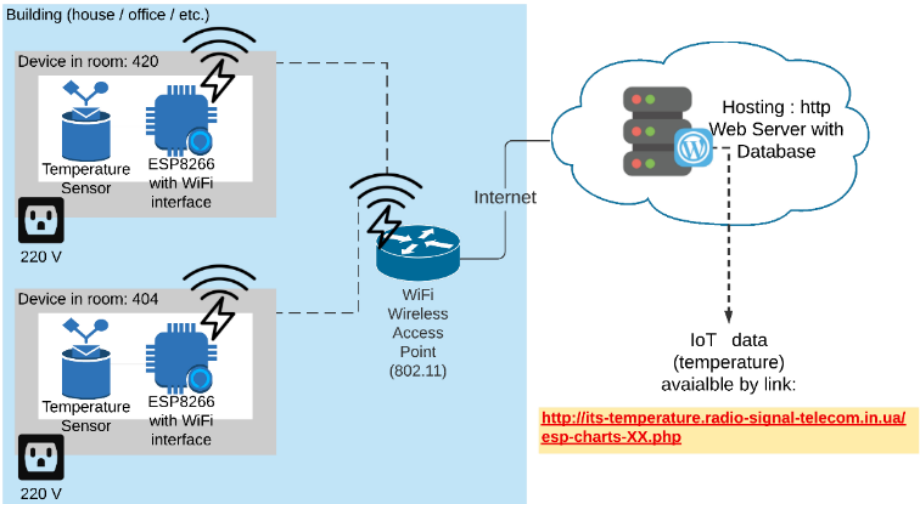

Рисунок 2.3 – Архітектура IoT-рішення на базі кастомного хостингу [1]

На відміну від рішення на *Blynk*, цей підхід вимагав значно більше зусиль з розробки на кожному етапі [14, 15]:

1. *Налаштування сервера*: На веб-хостингу було необхідно вручну створити базу даних *MySQL* з таблицею для зберігання показників. Також було розроблено *PHP-скрипт*, який виступав у ролі *API* – він приймав *HTTP POST* запити від мікроконтролера, валідував дані та записував їх у базу даних.
2. *Програмування мікроконтролера*: Код для *ESP8266* був складнішим. Замість однієї функції бібліотеки, потрібно було вручну формувати *HTTP-запит* (з заголовками та тілом, що містить дані з сенсора) та надсилати його на *URL* розробленого *PHP-скрипта* на хостингу.
3. *Створення інтерфейсу*: Для візуалізації даних було створено окрему веб-сторінку. Ця сторінка використовувала *PHP* для зчитування останніх даних з бази *MySQL* та бібліотеку *JavaScript* (наприклад, *Chart.js*) для побудови графіків температури та вологості у браузері користувача [14, 15].

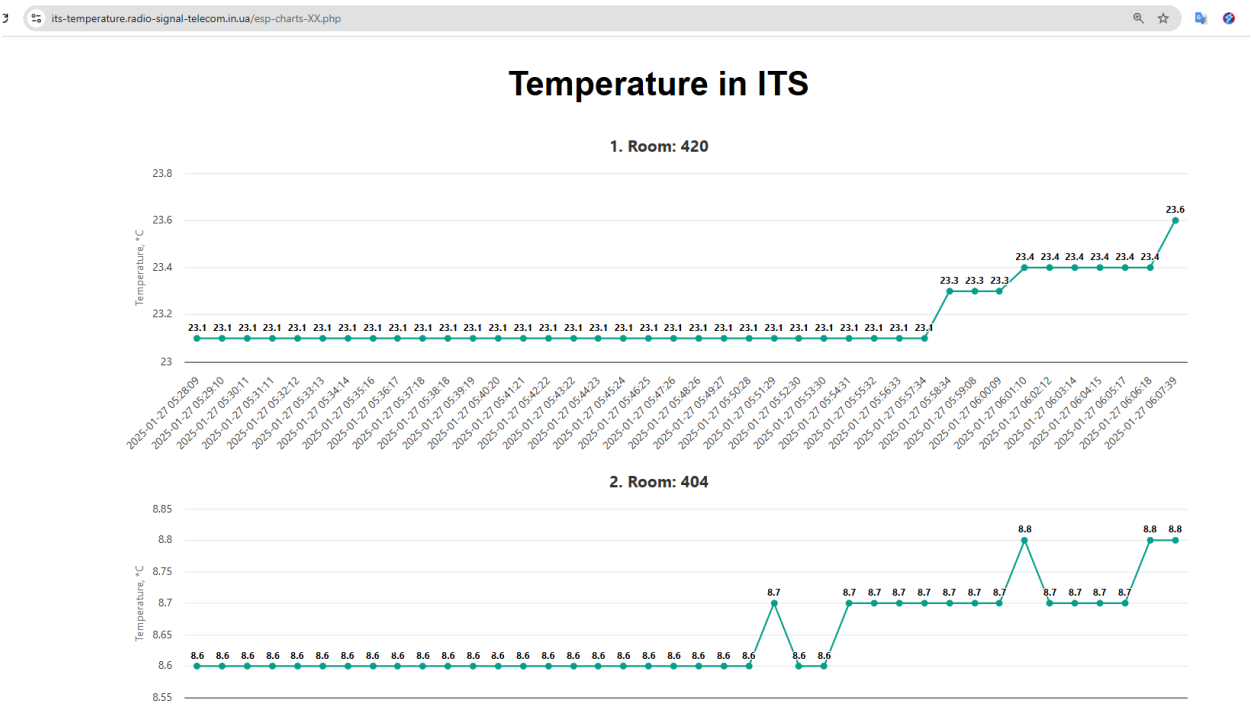

Рисунок 2.4 – Результат візуалізації даних на кастомному хостингу [1]

Порівняння цих двох практичних прикладів наочно демонструє фундаментальний компроміс: *Blynk* (спеціалізована *PaaS/SaaS*) забезпечує максимальну швидкість розробки ціною обмеженої гнучкості [10, 11], тоді як *кастомний хостинг* (*IaaS*-підхід) надає повний контроль над даними та функціоналом, але вимагає незрівнянно більших витрат часу та наявності у команди компетенцій у веб-розробці та адмініструванні баз даних [14, 15].

2.8 Висновки до розділу 2

У другому розділі було проведено комплексний аналіз сучасного ландшафту хмарних сервісів та платформ, призначених для розгортання рішень Інтернету речей, що стало емпіричною основою для подальшої розробки методики оцінювання. Дослідження ринку підтвердило наявність широкого спектра пропозицій, які суттєво відрізняються за архітектурними підходами, функціональними можливостями та моделями надання послуг. Огляд провідних публічних хмарних провайдерів, зокрема Amazon Web

Services, Microsoft Azure та Google Cloud Platform, виявив їхні ключові стратегічні відмінності. Встановлено, що AWS та Azure пропонують найбільш зрілі та всеосяжні екосистеми PaaS, які охоплюють повний життєвий цикл пристроїв через сервіси IoT Core та IoT Hub відповідно. Натомість стратегія Google Cloud Platform змістилася в бік використання загальних інструментів для прийому даних та фокусується на глибокій аналітиці, що вимагає від розробників побудови власної архітектури прийому телеметрії [1].

Особливу увагу в роботі було приділено детальному вивченню екосистеми Amazon Web Services як поточного лідера ринку. Аналіз ключових компонентів, таких як AWS IoT Core для комунікації, Greengrass для граничних обчислень та сервісів аналітики, продемонстрував можливості побудови високомасштабованих систем. Важливим результатом дослідження стало розуміння необхідності дотримання архітектурних стандартів, викладених у Well-Architected Framework, а також критичної важливості впровадження практик FinOps. Це дозволяє ефективно управляти витратами в умовах моделі оплати за фактом використання, яка є стандартною для публічних хмар, але несе ризики непередбачуваних бюджетних витрат при масштабуванні [1].

Поряд із глобальними провайдерами було проаналізовано сегмент спеціалізованих платформ, таких як Blynk, ThingSpeak та ThingsBoard. Дослідження показало, що ці рішення, які часто функціонують за моделями SaaS або спрощеного PaaS, забезпечують значно нижчий поріг входу та вищу швидкість розробки для прототипів або вузькоспеціалізованих завдань. Однак, на відміну від великих публічних хмар, вони часто накладають обмеження на гнучкість налаштувань, можливості глибокої аналітики даних або глобальну масштабованість, що робить їх вибір залежним від специфічних вимог конкретного проєкту [1].

Практичне порівняння двох реалізованих сценаріїв моніторингу температури наочно продемонструвало фундаментальний компроміс, з яким

стикаються архітектори IoT-систем. Рішення на базі платформи Blynk підтвердило переваги використання готових інструментів для мінімізації часу впровадження, тоді як реалізація на базі кастомного веб-хостингу ілюструє підхід, близький до IaaS, що надає повний контроль над системою, але вимагає значно більших трудових витрат та технічних компетенцій. Цей контраст підтверджує головну тезу дослідження про те, що не існує універсального рішення, придатного для всіх задач. Вибір інфраструктури завжди є результатом балансування між функціональними вимогами, бюджетом та часовими обмеженнями, що обґрунтовує необхідність розробки формалізованої математичної моделі для об’єктивного вибору платформи, яка буде представлена у наступному розділі.

3 МЕТОДИКА ОЦІНЮВАННЯ ТА ВИБОРУ ХМАРНИХ СЕРВІСІВ І ПЛАТФОРМ ДЛЯ ІОТ

3.1 Обґрунтування необхідності формалізованого підходу до вибору платформи

Результати, отримані в Розділі 2, наочно продемонстрували, що ринок хмарних сервісів та платформ для Інтернету речей є вкрай насиченим і різноманітним. Він включає як комплексні екосистеми глобальних провайдерів (*AWS, Azure*), що пропонують максимальну функціональність і масштабованість, так і спеціалізовані *PaaS/SaaS* рішення (*Blynk, ThingsBoard*), орієнтовані на швидкість і простоту розгортання. Кожен із цих підходів має унікальні технічні характеристики, моделі ціноутворення та вимоги до кваліфікації команди [1].

Саме ця різноманітність створює суттєву проблему для розробників та архітекторів IoT-систем: вибір оптимальної платформи стає нетривіальним завданням, оскільки він являє собою компроміс між низкою суперечливих критеріїв [1]. Наприклад, вибір на користь *кастомного хостингу* (підхід *IaaS*) забезпечує максимальну гнучкість, але значно збільшує *час впровадження* та *складність* проєкту. Натомість, вибір *Blynk* забезпечує мінімальний *час* та *бюджет* для невеликих рішень, але обмежує *масштабованість* та *аналітичну функціональність* [1].

Таким чином, відсутність універсального рішення та складність порівняння різнорідних пропозицій за множиною технічних та економічних параметрів створює значні труднощі при проєктуванні. Вибір, заснований на неповній інформації або суб'єктивних перевагах, призводить до високих ризиків [1]:

- *Ризик неефективного використання ресурсів* та *перевитрат бюджету* через вибір занадто потужної та дорогої платформи для простого завдання, або, навпаки, занадто слабкої платформи, яка не забезпечить необхідної *масштабованості* та *надійності* (*NFRs*) [1].

- *Ризик невідповідності кінцевого рішення функціональним вимогам* через обмежені можливості обраної платформи, особливо в частині аналізу даних та інтеграції.

Тому, для подолання цієї проблеми, виникає обґрунтована необхідність у розробці *формалізованого, об'єктивного та науково обґрунтованого підходу* [1]. Такий підхід має трансформувати якісний аналіз ринку (виконаний у Розділі 2) у кількісну оцінку, дозволяючи об’єктивно ранжувати платформи-кандидати. Це вимагає створення математичної моделі, яка здатна інтегрувати та зважити ключові критерії вибору – *час, бюджет, функціональну* та *нефункціональну відповідність* [1]. Розробка такої моделі є ключовим науково-практичним завданням, вирішення якого сприятиме підвищенню якості та ефективності впровадження IoT-рішень.

3.2 Розробка математичної моделі на основі функції оцінки корисності

Для вирішення завдання формалізованого порівняння, обґрунтованого в попередньому пункті, пропонується розробка математичної моделі. Ця модель базується на підході *теорії прийняття рішень* та використовує *функцію оцінки корисності* (*utility-based scoring function*). Мета моделі – надати кількісний, інтегральний показник (*Utility Score*), який відображає ступінь придатності кожної платформи-кандидата для конкретного IoT-проєкту з урахуванням його унікальних пріоритетів.

Нехай C – це множина всіх хмарних сервісів та платформ, що розглядаються як кандидати для розгортання IoT-рішення (наприклад, AWS, Azure, Blynk, ThingsBoard). Для кожного сервісу c з цієї множини ($c \in C$) ми визначаємо показник корисності $U(c)$ за наступною формулою:

$$U(c) = w1 \cdot T(c) + w2 \cdot B(c) + w3 \cdot F(c) + w4 \cdot N(c), \quad (3.1)$$

де:

- $T(c) \in [0, 1]$ – нормалізована оцінка *часу впровадження* (1 – найкраще, найшвидше впровадження);
- $B(c) \in [0, 1]$ – нормалізована оцінка *бюджету* (1 – найкраще, найдешевше впровадження);
- $F(c) \in [0, 1]$ – нормалізована оцінка відповідності *функціональним вимогам* (*FRs*) (1 – найкраще, платформа повністю відповідає);
- $N(c) \in [0, 1]$ – нормалізована оцінка відповідності *нефункціональним вимогам* (*NFRs*) (1 – найкраще, платформа повністю відповідає);
- $w1, w2, w3, w4 \in [0, 1]$ – вагові коефіцієнти, що відображають важливість (пріоритет) кожного критерію для конкретного IoT-проєкту.

Після розрахунку показника корисності $U(c)$ для кожної платформи-кандидата, остаточний вибір c^* здійснюється шляхом знаходження сервісу, що максимізує цю функцію:

$$c^* = \arg\max U(c), c \in C \qquad (3.2)$$

Таким чином, запропонована модель дозволяє звести складне багатокритеріальне завдання вибору до чіткої математичної оптимізації. Вона формалізує процес прийняття рішення, роблячи його прозорим, об'єктивним та адаптивним до пріоритетів будь-якого проєкту через налаштування вагових коефіцієнтів [1]. У наступних пунктах буде детально розглянуто зміст кожного критерію (T, B, F, N) та процес їх нормалізації.

3.3 Визначення ключових критеріїв оцінювання

Ефективність запропонованої математичної моделі (3.1) напряму залежить від чіткого та об’єктивного визначення її складових. Було виділено чотири ключові критерії, які комплексно охоплюють основні аспекти, що впливають на вибір платформи: час, бюджет, функціональна відповідність та

нефункціональна відповідність. Ці критерії були обрані, оскільки вони відображають фундаментальні обмеження та вимоги, проаналізовані в Розділі 1, пункт 1.5.

3.3.1 Критерій часу впровадження (T)

Критерій *часу впровадження* (Time-to-Market) є одним із найважливіших у сучасному бізнес-середовищі. Він оцінює, наскільки швидко IoT-рішення може бути розроблене, протестоване та виведене на ринок з використанням обраної платформи. Ця оцінка включає декілька факторів:

- *Швидкість початкового налаштування*: Скільки часу потрібно на створення облікових записів, налаштування середовища та підключення першого пристрою.
- *Простота розробки*: Наявність готових бібліотек (*SDK*), якість документації, наявність інструментів *low-code/no-code* (як у *Blynk*), що прискорюють розробку.
- *Складність інтеграції*: Час, необхідний для підключення платформи до інших сервісів (баз даних, аналітики, зовнішніх *API*).

Платформи типу *SaaS* (наприклад, *Blynk*) зазвичай мають найкращий показник за цим критерієм ($T \rightarrow 1.0$), тоді як рішення *IaaS* (кастомний хостинг), що вимагають ручного налаштування інфраструктури, мають найгірший ($T \rightarrow 0.1$) [1].

3.3.2 Критерій бюджету (B)

Критерій *бюджету* оцінює повну вартість володіння (*Total Cost of Ownership, TCO*) IoT-рішенням на обраній платформі. Важливо враховувати не лише прямі щомісячні платежі за сервіси, але й приховані витрати. Оцінка включає:

- *Початкові витрати (CapEx)*: Вартість ліцензій, плата за розгортання *on-premise* серверів (якщо застосовується).
- *Операційні витрати (OpEx)*: Щомісячна абонентська плата (*SaaS*), плата за фактичне використання ресурсів (*PaaS/IaaS* – трафік, повідомлення, хвилини підключення), вартість зберігання даних.
- *Непрямі витрати*: Вартість робочого часу кваліфікованих розробників та *DevOps*-інженерів, необхідних для підтримки та адміністрування платформи (особливо актуально для *IaaS* та *open-source* рішень, як-от *ThingsBoard*).

Безкоштовні та *open-source* рішення (*ThingSpeak, ThingsBoard CE*) мають високий бал за цим критерієм ($B \to 1.0$), тоді як великі публічні хмари (*AWS, Azure*) при масштабуванні можуть стати дуже дорогими ($B \to 0.1$) [1].

3.3.3 Критерій функціональної відповідності (F)

Цей критерій оцінює, наскільки добре платформа задовольняє специфічні *функціональні вимоги* (FRs), визначені для проєкту (див. пункт 1.5.1). Це оцінка того, *що* система може робити "з коробки" або з мінімальними доналаштуваннями.

Сюди входить наявність та гнучкість таких функцій:

- *Підтримка протоколів*: Можливість підключення пристроїв через необхідні протоколи (*MQTT, HTTP, CoAP, LoRaWAN*).
- *Управління пристроями*: Наявність інструментів для реєстрації, моніторингу стану, оновлення ПЗ (*OTA*).
- *Обробка даних*: Можливості *Rule Engine* (механізму правил) для обробки даних у реальному часі.
- *Візуалізація*: Наявність та гнучкість конструктора веб-панелей (*dashboards*) та мобільних додатків.

Платформи, як-от *AWS* або *ThingsBoard*, пропонують надзвичайно високу функціональність (F → 1.0), тоді як простіші сервіси, як-от *ThingSpeak*, мають базовий функціонал (F → 0.38) [1].

3.3.4 Критерій нефункціональної відповідності (N)

Критерій *нефункціональної відповідності* (*NFRs*) оцінює атрибути якості системи (див. пункт 1.5.2), тобто те, *наскільки добре* вона виконує свої функції. Це критично важливо для промислових та комерційних рішень.

Ключові атрибути цього критерію:

- *Масштабованість*: Здатність платформи обробляти зростаюче навантаження (мільйони пристроїв та повідомлень) без деградації продуктивності.
- *Надійність та Доступність*: Гарантії безвідмовної роботи, що надаються провайдером (*SLA – Service Level Agreement*).
- *Безпека*: Наявність механізмів автентифікації, шифрування, управління доступом, відповідність стандартам безпеки.
- *Гнучкість*: Можливість кастомізації платформи під унікальні бізнес-процеси.

Великі публічні хмари (*AWS, Azure*) забезпечують найвищий рівень *NFRs* (N → 1.0), оскільки вони спроєктовані для глобальних навантажень. Прості платформи, орієнтовані на прототипування (*ThingSpeak*), зазвичай мають низькі показники за цим критерієм (N → 0.18) [1].

3.4 Нормалізація показників критеріїв

Критерії, визначені в попередньому пункті, за своєю природою є різнорідними. *Час впровадження* (T) може вимірюватися в днях або тижнях, *бюджет* (B) – у грошових одиницях, а *функціональна* (F) та *нефункціональна* (N) відповідність часто є якісними експертними оцінками (наприклад,

"висока", "середня", "низька"). Для того, щоб математична модель (3.1) працювала коректно, всі ці показники необхідно привести до єдиної безрозмірної шкали.

Процес *нормалізації* полягає у перетворенні кожного показника в числове значення в стандартизованому діапазоні від 0 до 1. У цій моделі прийнято, що *1.0* відповідає *найкращому* (найбільш бажаному) значенню критерію (найшвидший час, найдешевший бюджет, повна відповідність), а *0* (або близьке до нього значення, наприклад, 0.1) – *найгіршому*.

Для критеріїв F (функціональність) та N (нефункціональність), а також для якісних оцінок T і B (як-от "швидко", "дорого"), пропонується використовувати метод *експертного шкалювання*. У ході аналізу ринку (Розділ 2) ми вже надали якісні оцінки платформам. У Таблиці 3.1 наведено відповідність між цими якісними оцінками та їхніми нормалізованими числовими еквівалентами, які будуть використовуватися в розрахунках [1].

Таблиця 3.1 – Шкалювання якісних оцінок критеріїв T, B, F, N [1]

Якісна оцінка	Кількісне значення часу T(c)	Кількісне значення бюджету B(c)	Кількісне значення відповідності F(c), N(c)
Дуже високий / Дуже швидко / Дуже дешево	1.0	1.0	1.0
Високий / Швидко / Дешево	0.75	0.75	0.75
Середній / Середньо	0.5	0.5	0.5
Низький / Довго / Дорого	0.25	0.25	0.25
Дуже низький / Дуже довго / Дуже дорого	0.1	0.1	0.1

Ця таблиця шкалювання дозволяє об'єктивно перевести експертні висновки, зроблені в Розділі 2, у числові показники, готові для підстановки у функцію оцінки корисності (3.1). Наприклад, платформа *Blynk*, оцінена як

"Дуже швидка" за часом впровадження, отримує оцінку *T(c) = 1.0*, тоді як *AWS*, оцінена як "Середня", отримує *T(c) = 0.5* [1].

3.5 Введення вагових коефіцієнтів для критеріїв

Після того, як всі критерії (T, B, F, N) визначені та приведені до єдиної нормалізованої шкали [0, 1], математична модель (3.1) вимагає ще одного ключового елемента – *вагових коефіцієнтів* (*w1, w2, w3, w4*). Саме ці коефіцієнти надають моделі необхідну гнучкість та роблять її практично застосовною.

Необхідність введення вагових коефіцієнтів обумовлена тим, що кожен IoT-проєкт має унікальні бізнес-цілі та пріоритети. Наприклад, для швидкого прототипування (*MVP – Minimum Viable Product*) або освітнього проєкту найважливішими будуть *час впровадження* (T) та *бюджет* (B). Водночас, для розробки медичної системи моніторингу пацієнтів або промислового рішення (*IIoT*) пріоритети кардинально зміщуються в бік *нефункціональної відповідності* (N) – надійності, безпеки, масштабованості.

Вагові коефіцієнти *w* дозволяють архітектору або особі, що приймає рішення, формалізувати ці пріоритети. Кожному критерію присвоюється вага від 0 до 1, що відображає його відносну важливість для успіху проєкту. При цьому загальна сума всіх коефіцієнтів обов'язково має дорівнювати 1:

$$w1 + w2 + w3 + w4 = 1 \qquad (3.3)$$

Таким чином, якщо для проєкту найважливішим є час виходу на ринок, коефіцієнту *w1* (час) може бути присвоєно значення, наприклад, 0.5 (50%), а решта 0.5 будуть розподілені між іншими критеріями. Якщо ж головне – надійність, то *w4* (нефункціональність) може отримати вагу 0.6 (60%). Цей механізм дозволяє адаптувати одну й ту саму математичну модель до абсолютно різних бізнес-завдань та сценаріїв використання [1].

3.6 Формула розрахунку інтегрального показника корисності (Utility Score)

У попередніх пунктах 3.2–3.5 було детально розібрано компоненти, необхідні для побудови формалізованої методики оцінювання. Було введено саму математичну модель, визначено ключові критерії (T, B, F, N), запропоновано метод їх нормалізації (приведення до шкали [0, 1]) та обґрунтовано введення вагових коефіцієнтів (w) для відображення пріоритетів проєкту.

Таким чином, фінальна формула для розрахунку інтегрального показника корисності (*Utility Score*) для будь-якої платформи-кандидата *c* має наступний вигляд, як це було визначено у формулі (3.1):

$$U(c) = w1 \cdot T(c) + w2 \cdot B(c) + w3 \cdot F(c) + w4 \cdot N(c), \quad (3.1)$$

де *w* – вагові коефіцієнти, сума яких дорівнює 1, а *T, B, F, N* – нормалізовані оцінки відповідних критеріїв.

Кінцеве рішення щодо вибору оптимальної платформи *c** приймається шляхом знаходження кандидата, що максимізує цей інтегральний показник, відповідно до формули (3.2):

$c^* = arg\ max\ U(c)$, (3.2) (де *c* є елементом *C*)

Ця функція корисності [1] є ключовим інструментом запропонованої методики. Вона дозволяє перевести набір різнорідних якісних та кількісних оцінок в єдиний, об’єктивний числовий показник, на основі якого можна приймати обґрунтоване архітектурне рішення. У наступному пункті буде проведено практичну апробацію цієї методики на конкретному сценарії.

3.7 Практична апробація методики оцінювання

Розроблена математична модель (3.1) потребує апробації на практичному прикладі для демонстрації її ефективності. Апробація проводиться шляхом розрахунку інтегрального показника корисності (*Utility Score*) для кількох платформ-кандидатів у рамках чітко визначеного сценарію та різних наборів пріоритетів.

3.7.1 Вибір сценарію для порівняльного аналізу

У якості типового сценарію для апробації було обрано завдання, що розглядалося у практичних прикладах (пункт 2.7) – *створення системи моніторингу температури та вологості для невеликого офісного приміщення*.

Для порівняльного аналізу було обрано три платформи, що представляють різні архітектурні підходи [1]:

1. AWS IoT Core, як представник комплексної *PaaS*-екосистеми "великої трійки".
2. Blynk, як представник спеціалізованої *SaaS/PaaS* платформи, орієнтованої на швидкість.
3. Кастомний хостинг, як представник *IaaS*-підходу з використанням власного *PHP*-скрипта та *MySQL*.

3.7.2 Оцінювання платформ за критеріями T, B, F, N

На основі аналізу, проведеного в Розділі 2, та з використанням таблиці нормалізації (Таблиця 3.1), присвоїмо кожній платформі кількісні оцінки за критеріями T, B, F та N.

AWS IoT Core:

- T (Час): "Середньо" (0.5) – вимагає налаштування кількох сервісів (IoT Core, Lambda, DynamoDB).
- B (Бюджет): "Дорого" (0.25) – хоча є безкоштовний рівень, при масштабуванні вартість швидко зростає; модель оплати складна.
- F (Функціональність): "Дуже висока" (1.0) – надає Rules Engine, Device Shadow, інтеграцію з усіма сервісами AWS.
- N (Нефункціональність): "Дуже висока" (1.0) – гарантує глобальну масштабованість, високу надійність (SLA) та безпеку.

Blynk:

- T (Час): "Дуже швидко" (1.0) – найшвидший варіант, готовий мобільний додаток "з коробки".
- B (Бюджет): "Дешево" (0.75) – для невеликої кількості пристроїв тарифи є дуже низькими або безкоштовними.
- F (Функціональність): "Середня" (0.5) – добре виконує базові завдання (візуалізація, сповіщення), але аналітика та інтеграції обмежені.
- N (Нефункціональність): "Середня" (0.5) – масштабованість та SLA обмежені тарифами; менша гнучкість.

Кастомний хостинг:

- T (Час): "Довго" (0.25) – вимагає найбільше часу на розробку (БД, PHP-скрипт, frontend).
- B (Бюджет): "Дуже дешево" (1.0) – вартість звичайного веб-хостингу є фіксованою та низькою.
- F (Функціональність): "Низька" (0.25) – "з коробки" функціоналу немає, все потрібно розробляти вручну.
- N (Нефункціональність): "Дуже низька" (0.1) – надійність та безпека залежать від найдешевшого хостингу; масштабованість майже відсутня.

Результати кількісного оцінювання зведено у Таблиці 3.2.

Таблиця 3.2 – Кількісна оцінка платформ-кандидатів для сценарію моніторингу

Платформа	T(c) (Час)	B(c) (Бюджет)	F(c) (Функціональність)	N(c) (Нефункціональність)
AWS IoT Core	0.50	0.25	1.00	1.00
Blynk	1.00	0.75	0.50	0.50
Кастомний хостинг	0.25	1.00	0.25	0.10

3.7.3 Призначення вагових коефіцієнтів для різних пріоритетів проєкту

Тепер розглянемо два різні сценарії пріоритетів для одного й того ж завдання:

- Сценарій А: Швидке прототипування (MVP).

Головна мета – максимально швидко і дешево перевірити гіпотезу. Пріоритети: час та бюджет. Функціональність та надійність вторинні.

  - w1 (Час) = 0.5
  - w2 (Бюджет) = 0.4
  - w3 (Функціональність) = 0.1
  - w4 (Нефункціональність) = 0.0
  - *Сума = 1.0*

- Сценарій Б: Надійне корпоративне рішення.

Головна мета – створити систему для довготривалого використання в офісі, яка має бути надійною, безпечною та мати потенціал для розширення (наприклад, інтеграції з іншими системами). Пріоритети: Нефункціональність (w4) та Функціональність (w3).

  - w1 (Час) = 0.1
  - w2 (Бюджет) = 0.1
  - w3 (Функціональність) = 0.4
  - w4 (Нефункціональність) = 0.4
  - *Сума = 1.0*

3.7.4 Розрахунок Utility Score та аналіз результатів

Підставимо значення з Таблиці 3.2 та вагові коефіцієнти для обох сценаріїв у формулу (3.1).

Розрахунок для Сценарію А (MVP: пріоритет T і B):

- *U(AWS)* = (0.5 * 0.50) + (0.4 * 0.25) + (0.1 * 1.00) + (0.0 * 1.00) = 0.25 + 0.10 + 0.10 + 0 = *0.45*
- *U(Blynk)* = (0.5 * 1.00) + (0.4 * 0.75) + (0.1 * 0.50) + (0.0 * 0.50) = 0.50 + 0.30 + 0.05 + 0 = *0.85*
- *U(Hosting)* = (0.5 * 0.25) + (0.4 * 1.00) + (0.1 * 0.25) + (0.0 * 0.10) = 0.125 + 0.40 + 0.025 + 0 = *0.55*

Результат Сценарію А: c* = arg max(0.45; 0.85; 0.55) = Blynk.

Для швидкого прототипування платформа Blynk є беззаперечним лідером, значно випереджаючи конкурентів.

Розрахунок для Сценарію Б (Надійне рішення: пріоритет F і N):

- *U(AWS)* = (0.1 * 0.50) + (0.1 * 0.25) + (0.4 * 1.00) + (0.4 * 1.00) = 0.05 + 0.025 + 0.40 + 0.40 = *0.875*
- *U(Blynk)* = (0.1 * 1.00) + (0.1 * 0.75) + (0.4 * 0.50) + (0.4 * 0.50) = 0.10 + 0.075 + 0.20 + 0.20 = *0.575*
- *U(Hosting)* = (0.1 * 0.25) + (0.1 * 1.00) + (0.4 * 0.25) + (0.4 * 0.10) = 0.025 + 0.10 + 0.10 + 0.04 = *0.265*

Результат Сценарію Б: c* = arg max(0.875; 0.575; 0.265) = AWS IoT Core.

Як тільки пріоритети зміщуються в бік надійності, масштабованості та функціональності, платформа AWS стає оптимальним вибором, незважаючи на вищу вартість та час впровадження.

Ця апробація доводить, що запропонована методика дозволяє об'єктивно та кількісно обґрунтувати вибір хмарної платформи, адаптуючи його до конкретних пріоритетів IoT-проєкту [1].

3.8 Висновки до розділу 3

У третьому розділі було вирішено ключове науково-практичне завдання дисертаційної роботи – розроблено та обґрунтовано формалізовану методику для порівняльного оцінювання та вибору хмарних сервісів і платформ для IoT-рішень. Необхідність такої методики була обґрунтована результатами аналізу ринку (Розділ 2), який продемонстрував значну різноманітність пропозицій та відсутність універсального рішення, що створює ризики неефективного вибору інфраструктури.

Для вирішення цієї проблеми було запропоновано математичну модель, що базується на *функції оцінки корисності* (*Utility Score*). Ця модель інтегрує чотири ключові критерії, що комплексно описують придатність платформи: *час впровадження* (T), *бюджет* (B), *функціональна відповідність* (F) та *нефункціональна відповідність* (N). Було детально розкрито зміст кожного критерію та запропоновано метод *нормалізації* – експертне шкалювання якісних оцінок для приведення їх до єдиного числового діапазону [0, 1], що робить різнорідні показники порівнянними.

Ключовим елементом моделі, що забезпечує її гнучкість, стало введення *вагових коефіцієнтів* (w). Цей механізм дозволяє адаптувати методику до унікальних бізнес-пріоритетів будь-якого IoT-проєкту, надаючи більшої ваги тим критеріям, які є найбільш критичними для конкретного завдання (наприклад, швидкість для MVP або надійність для корпоративного рішення).

Ефективність методики підтверджено практичною апробацією на прикладі моніторингу температури. Порівняння AWS IoT Core, Blynk та кастомного хостингу показало, що зміна вагових коефіцієнтів математично обґрунтовує вибір платформи: Blynk — для швидкого прототипування, AWS — для надійних корпоративних рішень. Це доводить, що методика є дієвим інструментом для вибору архітектури IoT-систем [1].

4 РЕКОМЕНДАЦІЇ ЩОДО СТВОРЕННЯ СТІЙКИХ ТА ЕКОНОМІЧНО ЕФЕКТИВНИХ ІОТ-РІШЕНЬ

4.1 Забезпечення стійкості IoT-систем в умовах нестабільного зв'язку

У сучасних умовах експлуатації, особливо в контексті енергетичної нестабільності та перебоїв у роботі телекомунікаційних мереж, критичною вимогою до систем Інтернету речей стає їхня стійкість. Традиційні архітектури, що покладаються на постійне з’єднання з хмарою для передачі телеметрії та отримання команд, виявляються вразливими. Втрата зв’язку навіть на короткий час може призвести до втрати критичних даних або неможливості керування виконавчими механізмами. Тому архітектура має будуватися за принципом «Offline-first», де здатність пристрою функціонувати автономно є пріоритетною.

Аналіз поведінки досліджуваних платформ (AWS IoT Core, Blynk, кастомний хостинг) в умовах втрати зв'язку дозволив виявити суттєві відмінності у механізмах забезпечення цілісності даних.

Для платформ, що використовують протокол MQTT (AWS IoT Core, ThingsBoard), ключовим механізмом забезпечення надійності є використання рівнів якості обслуговування (QoS – Quality of Service). У сценаріях з нестабільною мережею рекомендується використання рівня QoS 1 («at least once»), який гарантує доставку повідомлення брокеру, вимагаючи підтвердження (PUBACK). Якщо пристрій не отримує підтвердження через втрату зв'язку, він повинен зберігати повідомлення у локальному буфері та повторювати спробу після відновлення з'єднання. AWS IoT Core також надає механізм Device Shadow, який дозволяє асинхронно синхронізувати стан пристрою. Коли пристрій виходить в онлайн, він отримує «різницю» (delta) між своїм поточним станом та бажаним станом, збереженим у хмарі, що дозволяє коректно відновити роботу після блекауту [1].

Для рішень на базі спеціалізованих платформ, таких як Blynk, можливості забезпечення резильєнтності є обмеженими на рівні прошивки.

Бібліотека Blynk має вбудовані механізми перепідключення, проте за замовчуванням вона блокує виконання основного циклу програми (void loop) при спробі відновити з’єднання. Це є критичним недоліком для автономних систем, оскільки пристрій перестає опитувати сенсори або керувати реле. Для вирішення цієї проблеми рекомендується використовувати неблокуючі алгоритми підключення (використання таймерів замість функцій затримки delay) та локальне кешування даних. Оскільки Blynk орієнтований на відображення поточного стану («тут і зараз»), історичні дані, зібрані під час відсутності інтернету, можуть бути втрачені або некоректно відображені на графіках без додаткової логіки пакетного завантаження (batch upload) з мітками часу.

У випадку використання кастомного хостингу на базі протоколу HTTP (REST API), відповідальність за збереження даних повністю покладається на розробника вбудованого ПЗ. Протокол HTTP не має вбудованих черг повідомлень для клієнта. Для забезпечення стійкості необхідно реалізувати алгоритм локальної буферизації на стороні мікроконтролера. Пристрій має записувати виміри у незалежну пам'ять (наприклад, SPIFFS або EEPROM на ESP8266) під час відсутності зв’язку. При відновленні з’єднання алгоритм повинен зчитувати накопичені дані та відправляти їх на сервер пакетами. Важливо, щоб серверний скрипт (PHP/Python) був налаштований на прийом масиву даних з історичними мітками часу (timestamp), а не присвоював час отримання пакету як час вимірювання, що є поширеною помилкою при проєктуванні простих систем.

Порівняльний аналіз стратегій забезпечення резильєнтності наведено в таблиці 4.1.

Таблиця 4.1 – Стратегії забезпечення стійкості IoT-рішень на різних платформах

Характеристика	AWS IoT Core (PaaS)	Blynk (SaaS)	Кастомний хостинг (IaaS)
Протокол передачі	MQTT (підтримка QoS)	Власний бінарний / MQTT	HTTP / HTTPS
Поведінка при втраті мережі	Збереження повідомлень у черзі (при QoS>0)	Автоматичне перепідключення (може блокувати loop)	Помилка з'єднання, втрата даних без буфера
Синхронізація стану	Device Shadow (автоматично)	Останнє значення на сервері	Реалізується вручну (запит до БД)
Збереження даних офлайн	Вимагає реалізації локального буфера (напр., Greengrass)	Обмежено, дані можуть бути втрачені	Вимагає написання коду для запису у Flash/SD
Складність реалізації стійкості	Середня (наявні інструменти SDK)	Висока (обхід обмежень бібліотеки)	Дуже висока (розробка всієї логіки з нуля)

Отже, для критично важливих систем, що працюють в умовах нестабільного енергопостачання та зв'язку, перевагу слід надавати платформам з нативною підтримкою MQTT QoS та механізмами тіньових копій пристроїв (AWS, Azure, ThingsBoard). Використання HTTP-рішень вимагає значних трудовитрат на розробку власних механізмів буферизації та синхронізації, що підвищує ризики втрати даних.

4.2 Аналіз енергоефективності та стратегії "Green IoT"

Ключовим фактором стійкості IoT-рішень, особливо тих, що працюють від автономних джерел живлення (батарей, акумуляторів), є енергоефективність. Стратегія "Green IoT" передбачає мінімізацію вуглецевого сліду та подовження життєвого циклу пристроїв без заміни елементів живлення. Для досягнення цієї мети необхідно комплексно оптимізувати як апаратну частину, так і програмні алгоритми передачі даних.

Аналіз протоколів передачі даних показує, що вибір між HTTP та MQTT має вирішальний вплив на енергоспоживання. Протокол HTTP, який використовується в архітектурі кастомного хостингу (Розділ 2.7.2), є текстовим і вимагає встановлення нового TCP-з'єднання для кожного запиту (заголовки, "рукостискання"). Це створює значні накладні витрати (overhead) трафіку та тримає радіомодуль активним довше. Натомість, протокол MQTT (використовується в AWS IoT Core та ThingsBoard) працює поверх постійного з'єднання, використовує бінарний формат та мінімальні заголовки. Дослідження показують, що для передачі однакового обсягу корисного навантаження MQTT витрачає у 10–20 разів менше енергії та трафіку порівняно з HTTP [13].

Для мікроконтролера ESP8266, використаного у практичній частині, критично важливим є використання режимів сну. У активному режимі передачі даних через Wi-Fi споживання струму сягає 170 мА, тоді як у режимі глибокого сну (Deep Sleep) воно знижується до 20 мкА. Для платформи *Blynk* та *кастомного хостингу* типовий алгоритм роботи передбачає постійне з'єднання, що швидко виснажує батарею. Для забезпечення енергоефективності рекомендується змінити алгоритм на циклічний: "Пробудження – Вимірювання – Передача – Глибокий сон".

Для підвищення "інтелектуальності" енергозбереження доцільно використовувати методи машинного навчання на периферії (Edge ML). Як зазначається у дослідженнях [16], впровадження простих моделей на мікроконтролері дозволяє адаптувати частоту передачі даних залежно від динаміки зміни показників. Якщо температура в приміщенні стабільна, пристрій може не виходити на зв'язок, накопичуючи дані локально, і передавати їх лише при суттєвих змінах або раз на добу для звіту.

У Таблиці 4.2 наведено розрахункове порівняння часу автономної роботи пристрою від стандартного акумулятора ємністю 2500 мАг для різних платформ та режимів роботи.

Таблиця 4.2 – Розрахунок часу автономної роботи IoT-пристрою (ESP8266, 2500 мАг) [13, 18]

Платформа / Режим	Протокол	Середній струм (активність/сон)	Частота передачі	Орієнтовний час роботи
Кастомний хостинг (Always On)	HTTP	~80 мА (без сну)	Безперервно	~31 година (1.3 доби)
Blynk (Default)	Власний/TCP	~70 мА (без сну, Blynk.run)	Безперервно	~35 годин (1.5 доби)
AWS IoT / ThingsBoard (Deep Sleep)	MQTT	170 мА (3 сек) / 20 мкА (597 сек)	1 раз на 10 хв	~90 днів (3 місяці)
AWS Greengrass (Edge ML)	MQTT	170 мА (3 сек) / 20 мкА (змінна)	Адаптивна (по події)	> 120 днів (4+ місяці)

Як видно з розрахунків, використання архітектурних можливостей платформ, що підтримують асинхронну передачу та не вимагають постійного з'єднання (AWS, ThingsBoard), у поєднанні з протоколом MQTT, дозволяє збільшити час автономної роботи на порядки. Для кастомних рішень та Blynk досягнення аналогічних показників вимагає значного ускладнення коду для реалізації логіки Deep Sleep та коректного відновлення сесії після сну [13].

4.3 Економічне моделювання сукупної вартості володіння (TCO)

Окрім технічних аспектів архітектури та енергоефективності, критичним фактором вибору платформи є економічна доцільність. Для об'єктивного порівняння варіантів реалізації IoT-рішення недостатньо оцінювати лише прямі витрати (вартість підписки або хостингу). Необхідно використовувати модель сукупної вартості володіння (*TCO – Total Cost of Ownership*), яка враховує як капітальні витрати (*CapEx*), так і операційні витрати (*OpEx*) на горизонті життєвого циклу проєкту.

У рамках дослідження було розроблено економічну модель для оцінки вартості експлуатації системи моніторингу середнього масштабу (наприклад, для складського приміщення або агропромислового комплексу).

Вхідні параметри для моделювання:

1. *Кількість пристроїв:* 100 одиниць.
2. *Частота передачі даних:* 1 повідомлення кожні 5 хвилин (288 повідомлень на добу на пристрій).
3. *Загальний обсяг повідомлень:* приблизно 864 000 повідомлень на місяць.
4. *Термін експлуатації:* 3 роки.

Для розрахунку було обрано три типові сценарії, проаналізовані у Розділі 2:

- *Сценарій 1: SaaS (Blynk).* Використовується комерційний тарифний план *PRO*, який покриває потреби до 100 пристроїв. Вартість становить фіксовану щомісячну плату. Важливою особливістю цієї моделі є ступінчасте ціноутворення: вартість послуг може різко зростати при масштабуванні. Якщо потреби бізнесу вийдуть за межі лімітів плану *PRO* (наприклад, знадобиться 101-й пристрій або функція брендування додатку *White Label*), необхідним стане перехід на тариф *Business*. Це призведе до миттєвого зростання операційних витрат приблизно у 6 разів (з 99 доларів до 600 доларів на місяць), що є суттєвим фінансовим ризиком для проєкту з обмеженим бюджетом [9].
- *Сценарій 2: PaaS (AWS IoT Core).* Використовується модель оплати за фактом використання (*Pay-as-you-go*). Вартість складається з плати за підключення (*Connectivity*), обмін повідомленнями (*Messaging*), виконання правил (*Rules Engine*) та зберігання даних (*DynamoDB/Timestream*). Згідно з калькулятором AWS, для вказаного обсягу трафіку витрати є лінійними та гнучкими [8].
- *Сценарій 3: IaaS (Кастомний хостинг).* Використовується віртуальний приватний сервер (*VPS*). До прямих витрат (оренда сервера, доменне ім'я) додаються *непрямі витрати* на адміністрування системи (оновлення ОС, налаштування безпеки, резервне копіювання). У моделі

прийнято, що підтримка працездатності власного сервера займає мінімум 4 години на місяць роботи кваліфікованого інженера.

Результати порівняльного розрахунку наведено в таблиці 4.3.

Таблиця 4.3 – Прогноз сукупної вартості володіння на 3 роки для 100 пристроїв

Стаття витрат	SaaS (Blynk PRO)	PaaS (AWS IoT)	IaaS (Кастомний хостинг)
Ліцензії / Підписка	~$99 / міс	$0	$0
Інфраструктура (Сервер/Трафік)	Включено в підписку	~$15 / міс	~$10 / міс (VPS + Backup)
Адміністрування (Human Labor)	0 годин (включено)	~1 година / міс	~4 години / міс
Вартість людської праці ($20/год)	$0	$20 / міс	$80 / міс
Разом за місяць	~$99	~$35	~$90
TCO за 3 роки	~$3 564	~$1 260	~$3 240

Результати моделювання демонструють, що для заданого сценарію платформа *AWS IoT Core* є найбільш економічно ефективною, забезпечуючи найнижчу сукупну вартість володіння ($1260 за 3 роки) завдяки відсутності фіксованих платежів та мінімальним витратам на адміністрування.

Варіанти *Blynk* (план PRO) та *Кастомний хостинг* демонструють схожі показники TCO ($3564 та $3240 відповідно), але мають різну структуру витрат та ризиків. Кастомний хостинг вимагає постійних вкладень у людські ресурси для підтримки працездатності. Платформа *Blynk* знімає навантаження з персоналу, але несе ризик "фінансового стрибка": незначне розширення проєкту понад ліміти поточного плану призведе до кратного збільшення бюджету, роблячи це рішення найдорожчим із розглянутих.

Таким чином, для довгострокових проєктів з планами на масштабування, використання архітектури *Serverless/PaaS* є найбільш економічно виправданим та прогнозованим [1].

4.4 Аналіз чутливості розробленої математичної моделі

Важливим етапом валідації запропонованої у Розділі 3 математичної моделі є проведення аналізу чутливості (*Sensitivity Analysis*). Цей метод дозволяє дослідити, наскільки стабільним є оптимальне рішення (c^*) при зміні вхідних параметрів, зокрема вагових коефіцієнтів (w), що відображають пріоритети стейкхолдерів.

Для проведення аналізу було змодельовано динамічну зміну пріоритетів між двома конфліктуючими критеріями: *Бюджетом (B)* та *Якістю/Надійністю (N).* Інші критерії (Час та Функціональність) були зафіксовані на базовому рівні.

Умови експерименту:

- Вагові коефіцієнти для *Часу (w1)* та *Функціональності (w3)* зафіксовані: $w1 = 0.1, w3 = 0.1$.
- Сумарна вага *Бюджету (w2)* та *Надійності (w4)* становить 0.8 ($w2 + w4 = 0.8$).
- Проведено серію розрахунків *Utility Score*, де вага Бюджету ($w2$) змінюється від 0.1 (бюджет неважливий) до 0.7 (бюджет єдиний пріоритет), а вага Надійності ($w4$) відповідно зменшується.

Результати розрахунку інтегрального показника наведено в таблиці 4.4.

Таблиця 4.4 – Результати аналізу чутливості моделі

Сценарій пріоритетів	Вага Бюджету (w2)	Вага Надійності (w4)	AWS IoT (PaaS)	Blynk (SaaS)	Кастомний хостинг (IaaS)	Переможець
1. Пріоритет Якості	0.1	0.7	0.875	0.575	0.220	AWS
2. Збалансований	0.4	0.4	0.650	0.650	0.490	AWS / Blynk
3. Економія	0.5	0.3	0.575	0.675	0.580	Blynk
4. Жорстка економія	0.7	0.1	0.425	0.725	0.760	Хостинг

Аналіз отриманих даних дозволяє зробити наступні висновки:

1. *Точка перемикання (Switching Point):* Модель чітко демонструє зміну лідера при зміні пріоритетів. При переході від сценарію "Пріоритет Якості" до "Збалансованого" ($w2 \approx 0.4$), лідерство переходить від дорогого, але надійного *AWS* до більш доступного *Blynk*.
2. *Ефект низької бази:* При екстремальному пріоритеті бюджету ($w2 = 0.7$, Сценарій 4), лідером стає *Кастомний хостинг*. Це підтверджує адекватність моделі: якщо замовник готовий ігнорувати ризики надійності заради економії на ліцензіях, математика вказує на найдешевший варіант.
3. *Стійкість рішення AWS:* Платформа AWS демонструє найвищий *Utility Score* у широкому діапазоні вагових коефіцієнтів, де вимога до надійності (N) перевищує 0.4. Це свідчить про те, що для критично важливих систем вибір на користь хмарних гігантів є математично обґрунтованим, навіть незважаючи на вищу вартість.

Таким чином, проведений аналіз чутливості підтверджує, що розроблена у Розділі 3 методика є гнучким інструментом, який коректно реагує на зміну бізнес-вимог і дозволяє знаходити оптимальне архітектурне рішення для широкого спектра вхідних умов [1].

4.5 Практичні рекомендації щодо вибору платформи для типових сценаріїв

На основі проведеного комплексного аналізу ринку (Розділ 2), розробленої методики оцінювання (Розділ 3) та економічного моделювання (пункт 4.3), сформульовано практичні рекомендації щодо вибору хмарної інфраструктури для найбільш поширених класів IoT-задач.

Сценарій 1: "Розумний стартап" (MVP та вихід на ринок)

- *Характеристика*: Обмежений бюджет, критично важливий час запуску (Time-to-Market), невелика команда (1-3 розробники), невизначена кількість майбутніх користувачів.
- *Рекомендація*: Використання спеціалізованих SaaS-платформ, зокрема *Blynk*.
- *Обґрунтування*: Аналіз показує, що хоча вартість підписки може бути вищою за "чисту" інфраструктуру, економія на розробці мобільного додатку та бекенду (які надаються "з коробки") перекриває операційні витрати на етапі до 1000 пристроїв. Це дозволяє команді фокусуватися на продукті, а не на адмініструванні серверів.
- *Застереження*: Необхідно закладати в архітектуру пристрою можливість зміни прошивки (OTA) для потенційної міграції на власну інфраструктуру у разі різкого масштабування, щоб уникнути ефекту "Vendor Lock-in".

Сценарій 2: "Корпоративна система моніторингу" (Enterprise/Industrial IoT)

- *Характеристика*: Високі вимоги до безпеки та надійності, стабільна кількість пристроїв (сотні або тисячі), потреба в інтеграції з існуючими бізнес-системами, довгострокова експлуатація (3+ роки).
- *Рекомендація*: Використання PaaS-рішень великих провайдерів, насамперед *AWS IoT Core* або *Azure IoT Hub*.
- *Обґрунтування*: Розрахунок TCO (пункт 4.3) довів, що на горизонті 3 років вартість володіння професійним хмарним рішенням є найнижчою завдяки моделі оплати за трафік та відсутності витрат на адміністрування фізичних серверів. Крім того, аналіз чутливості (пункт 4.4) підтвердив, що при пріоритеті надійності та безпеки AWS є безальтернативним лідером.
- *Архітектурна порада*: Використання протоколу MQTT та "тіньових копій" (Device Shadows) є обов'язковим для забезпечення роботи в умовах нестабільного зв'язку.

Сценарій 3: "Локальна автоматизація" (Бюджетний сегмент / DIY)

- *Характеристика:* Критично обмежений бюджет, відсутність вимог до глобальної масштабованості, допустимість періодичних простоїв, наявність кваліфікованого інженера для підтримки.
- *Рекомендація*: Розгортання *кастомного рішення* (наприклад, ThingsBoard CE або власний стек MQTT+Node-RED) на віртуальному приватному сервері (VPS).
- *Обґрунтування*: Це рішення забезпечує повний контроль над даними та фіксовану, прогнозовану вартість інфраструктури, яка не залежить від кількості повідомлень (на відміну від AWS). Однак, власник системи повинен бути готовим інвестувати власний час у налаштування безпеки та резервне копіювання, що є "прихованою" вартістю цього підходу.

Таблиця 4.5 – Зведена матриця вибору

Критерій вибору	Blynk (SaaS)	AWS IoT (PaaS)	Кастомний хостинг (IaaS)
Швидкість запуску	* * * * *	* * *	*
Масштабованість	* * *	* * * * *	* *
Економічна ефективність (малий масштаб)	* * * *	* * * * *	* * *
Економічна ефективність (великий масштаб)	* *	* * * * *	* * * *
Вимоги до кваліфікації команди	Низькі	Високі	Середні/Високі

Таким чином, вибір платформи не є статичним рішенням. Рекомендується починати з SaaS-рішень для перевірки гіпотез, а при досягненні економічної доцільності (точки беззбитковості) планувати міграцію на PaaS-архітектуру для забезпечення масштабування та оптимізації витрат.

4.6 Висновки до розділу 4

У четвертому розділі розроблено практичні рекомендації щодо створення стійких, енергоефективних та економічно обґрунтованих IoT-рішень, що базуються на результатах попереднього аналізу та моделювання.

Встановлено, що забезпечення резильєнтності (стійкості) системи в умовах нестабільного зв'язку вимагає архітектурного підходу «Offline-first». Порівняльний аналіз показав переваги протоколу MQTT (підтримується AWS, ThingsBoard) над HTTP (кастомний хостинг) завдяки механізмам QoS та меншим накладним витратам, що дозволяє зберігати цілісність даних при перебоях мережі.

Розрахунок енергоефективності довів, що використання режимів глибокого сну (*Deep Sleep*) та оптимізованих протоколів на платформі AWS дозволяє подовжити час автономної роботи пристрою (на базі ESP8266) від 1,5 доби до декількох місяців порівняно з постійним з'єднанням на простих платформах .

Проведене економічне моделювання сукупної вартості володіння (*TCO*) на горизонті 3 років спростувало міф про безумовну дешевизну власної інфраструктури. Врахування прихованих витрат на адміністрування показало, що професійне *PaaS*-рішення (AWS) може бути втричі вигіднішим за кастомний хостинг ($1260 проти $3240) та конкурентним з комерційними *SaaS*-тарифами при масштабуванні.

Аналіз чутливості підтвердив надійність розробленої математичної моделі вибору, продемонструвавши логічне перемикання оптимального вибору між AWS, Blynk та кастомним рішенням в залежності від зміни пріоритетів бюджету та надійності. Це дозволяє рекомендувати розроблену методику як дієвий інструмент для стратегічного планування IoT-проєктів.

ВИСНОВКИ

У магістерській дисертації вирішено актуальне науково-практичне завдання підвищення ефективності та обґрунтованості вибору хмарної інфраструктури для розгортання та експлуатації систем Інтернету речей. В умовах стрімкого зростання ринку IoT, який характеризується значною фрагментацією технологій та різноманітністю архітектурних підходів, відсутність формалізованих методик вибору призводить до ризиків створення неоптимальних, дорогих в обслуговуванні або технічно обмежених рішень.

В ході дослідження було проведено комплексний аналіз предметної області, який дозволив систематизувати теоретичні знання щодо еволюції, життєвого циклу та архітектури сучасних IoT-систем. Визначено, що для ефективного проєктування необхідно спиратися на п'ятирівневу архітектурну модель, в якій хмарні сервіси та платформи відіграють центральну роль у зборі, обробці та аналізі даних . Детальний розгляд моделей обслуговування (IaaS, PaaS, SaaS) дозволив класифікувати існуючі рішення за ступенем гнучкості та рівнем відповідальності розробника, що стало базисом для подальшого порівняння.

Аналіз ринку хмарних сервісів виявив фундаментальні відмінності у стратегіях провідних провайдерів. Встановлено, що глобальні екосистеми (AWS, Azure) пропонують найбільш зрілі PaaS-рішення, орієнтовані на повний цикл управління пристроями та даними, тоді як спеціалізовані платформи (Blynk, ThingsBoard) фокусуються на зниженні порогу входу та прискоренні розробки. Практичне порівняння реалізованих прототипів системи моніторингу наочно продемонструвало ключовий компроміс: використання SaaS-платформ забезпечує миттєвий старт, але обмежує функціональність, тоді як кастомні рішення на базі IaaS надають повний контроль, але вимагають значно більших ресурсів на розробку та підтримку .

Центральним науковим результатом роботи стала розробка та обґрунтування методики багатокритеріального вибору платформи. Створена

математична модель на основі функції корисності (Utility Score) дозволила інтегрувати різнорідні фактори – час впровадження, бюджетні обмеження, функціональні та нефункціональні вимоги – в єдиний інтегральний показник. Введення механізму вагових коефіцієнтів забезпечило адаптивність методики до унікальних бізнес-пріоритетів кожного проєкту. Валідація моделі, проведена через аналіз чутливості, підтвердила її здатність коректно визначати оптимальне рішення при зміні пріоритетів замовника, перемикаючись між AWS, Blynk та кастомним хостингом.

Практична цінність роботи суттєво посилена розробленими рекомендаціями щодо створення стійких та економічно ефективних рішень. На основі моделювання сукупної вартості володіння (TCO) доведено, що для довгострокових проєктів використання безсерверних (Serverless) технологій AWS є економічно вигіднішим за утримання власної інфраструктури, завдяки мінімізації прихованих витрат на адміністрування. Технічний аналіз енергоефективності підтвердив критичну перевагу протоколу MQTT та режимів глибокого сну для автономних пристроїв, дозволяючи подовжити час їх роботи на порядки. Сформульовані архітектурні принципи забезпечення резильєнтності (стійкості) до втрати зв'язку є особливо актуальними для забезпечення надійної роботи IoT-систем в сучасних умовах нестабільності мереж.

Таким чином, результати дисертації утворюють цілісний інструментарій, що дозволяє архітекторам та менеджерам проєктів переходити від інтуїтивного вибору технологій до інженерно та економічно обґрунтованих рішень, мінімізуючи ризики на всіх етапах життєвого циклу IoT-продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Yadav, P., Osypchuk, S., Bidikar, M., & Rodichev, I. (2025, April 14-18). *Cloud services and platforms research for internet of things applications deployment*. The 19th International Scientific and Technical Conference "Modern Challenges in Telecommunications" (MCT-2025), Kyiv, Ukraine.

2. Ilchenko, M., Uryvsky, L., & Osypchuk, S. (2019, September 9-13). *World trends of modern information and telecommunication technologies development*. The Fourth International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo-2019), Odesa, Ukraine. IEEE.

3. Ilchenko, M., Uryvsky, L., & Osypchuk, S. (2020). The main directions of improving information and communication technologies in the global trends. In *Advances in Information and Communication Technology and Systems* (pp. 3-22).

4. Uryvsky, L., Moshynska, A., & Osypchuk, S. (2019). Internet of things solutions usage perspectives in Ukraine and their applications specifics. In *The Potential of Modern Science* [Collective monograph].

5. Yadav, P., Li, Q., & Mortier, R. (2018). *IoT network behaviours and dependencies* (Series No. CDBB_WP_005). University of Cambridge.

6. Kumar, V., Yadav, P., et al. (2023). *Challenges in the design and implementation of IoT testbeds in smart-cities.*

7. Amazon Web Services, Inc. (n.d.). *AWS Well-Architected Framework: IoT Lens*. Retrieved October 20, 2025

8. Kolcun, R., Popescu, D. A., Safronov, V., Yadav, P., Mandalari, A. M., Mortier, R., & Haddadi, H. (2021). *Revisiting IoT device identification*.

9. Amazon Web Services, Inc. (n.d.). *AWS Pricing Calculator*. Retrieved October 20, 2025

10. Blynk Inc. (n.d.). *Blynk IoT Platform*. Retrieved October 20, 2025.

11. Chekunov, M., Osypchuk, I., Kyrashchuk, V., & Osypchuk, S. (2018, April 16-20). *Theoretical research and practical design of IoT testbed solution based on ESP8266 SoC and Blink components*. 12th International Scientific Conference

"Modern Challenges in Telecommunications" (PT-2018), Kyiv, Ukraine. Conference materials, 466-468.

12. Osypchuk, S., Moshinska, A., & Kirashchuk, V. (2019). *Applied aspects of implementation of information transmission solutions in the IoT technologies*. Proceedings of the International Scientific and Technical Conference "Perspectives of Telecommunications" (PT-2019), Kyiv, Ukraine, 17-21.

13. Kumar, V., Yadav, P., & Indrusiak, L. S. (2023). Resilient Edge: Building an adaptive and resilient multi-communication network for IoT Edge using LPWAN and WiFi. *IEEE Transactions on Network and Service Management, 20*(3), 3055-3071.

14. Iatsyshin, O., Maltsev, A., Dykyy, A., & Osypchuk, S. (2020). *Research and implementation of IoT temperature automated metering project based on telecommunication systems institute*. Proceedings of the International Scientific and Technical Conference "Perspectives of Telecommunications" (PT-2020), Kyiv, Ukraine, 343-345.

15. Uryvsky, L. O., Gerstacker, W. H., Moshynska, A. V., Osypchuk, S. O., & Yatsyshyn, O. V. (2020). Research and implementation of IoT projects for environment parameters and energy resource metering. *Information and Telecommunication Sciences, 11*(1), 27-34.

16. Yadav, P., Ngai, E. C., et al. (2024). Machine Learning on Edge in Sensor Systems (SenSys-ML 2024). *2024 IEEE 3rd Workshop on Machine Learning on Edge in Sensor Systems (SenSys-ML)*, 1-2. IEEE.

17. Kolcun, R., Yadav, P., et al. (2020). *The case for retraining of ML models for IoT device identification at the Edge*.

18. Espressif Systems. ESP8266EX Datasheet. Version 7.1, 2025.11