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

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «СИСТЕМА СПОСТЕРЕЖЕННЯ, АКТУАЛІЗАЦІЇ ТА ПРЕДИКТИВНОГО АНАЛІЗУ СТАТУСУ ПРИСТРОЇВ ІНТЕРНЕТУ РЕЧЕЙ»

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

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

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

Виконав: здобувач вищої освіти гр. КІД-41
Дмитро ПАЛАМАРЧУК
Ім'я, ПРІЗВИЩЕ
Керівник: Віра БАБЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: _____
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії
Ступінь вищої освіти Бакалавр
Спеціальність 123 Комп'ютерної інженерії
Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
Ім'я, ПРІЗВИЩЕ
«_____» _____ 2026р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Паламарчук Дмитро Євстахович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: Система спостереження, актуалізації та предиктивного аналізу статусу пристроїв Інтернету речей
керівник кваліфікаційної роботи Віра БАБЕНКО,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» 02 2026р. № 51
2. Строк подання кваліфікаційної роботи «08» 05 2026р.
3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література, методи системного аналізу для проєктування багаторівневих архітектур IoT-систем, методи машинного навчання та глибокої аналітики великих даних (Big Data) для побудови моделей предиктивного обслуговування.
Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Огляд сучасних технологій та рішень для контролю, програмного оновлення й оцінки майбутнього стану iot-систем
2. Створення комплексної системи контролю, предиктивної аналітики та віддаленої модернізації пристроїв інтернету речей
3. Створення та налагодження програмних компонентів iot-мережі й автоматизованих devops-платформ
4. Перелік ілюстративного матеріалу: *презентація*
5. Дата видачі завдання «09» 02 2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	25.02-24.03.2026	Виконано
2	Обробка матеріалу	23.03-05.04.2026	Виконано
3	Розробка теоретичної частини	06.04-10.04.2026	Виконано
4	Розробка мережі	11.04-20.04.2026	Виконано
5	Висновки за результатами аналізу	21.04-06.05.2026	Виконано
6	Оформлення пояснювальної записки	07.05-08.05.2026	Виконано
7	Розробка презентації	09.05-11.05.2026	Виконано
8	Передзахист	27.05-29.05.2026	Виконано
9	Здача в деканат	02.06-05.06.2026	Виконано

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

Керівник кваліфікаційної роботи _____ Віра БАБЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 72 стор., 15 рис., 4 табл., 15 джерел.

Мета дослідження – Розробка комплексної системи для безперервного моніторингу, предиктивного прогнозування технічного стану та організації віддаленого оновлення парку IoT-пристроїв.

Об'єкт дослідження – Процеси моніторингу, предиктивного обслуговування та дистанційного програмного оновлення в інфраструктурі Інтернету речей.

Предмет дослідження – Програмно-апаратні рішення, мережеві комунікаційні протоколи та хмарні DevOps-архітектури, що забезпечують управління IoT-системами.

Короткий зміст роботи: В умовах глобальної цифровізації та експоненційного збільшення кількості підключених пристроїв Інтернету речей (IoT) виникає гостра необхідність забезпечення їхньої безперебійної роботи. Класичні методології технічного супроводу (реактивні та превентивні) часто виявляються економічно та технічно неефективними, оскільки ігнорують реальний стан обладнання. З огляду на це, робота пропонує перехід до предиктивного (прогнозувального) обслуговування, яке базується на синергії технологій IoT, Big Data та машинного навчання.

Для ефективного управління життєвим циклом пристроїв досліджено інтеграцію методології DevOps у сферу Інтернету речей. Запропоновано трирівневу архітектурну модель системи, що включає базовий рівень кінцевих пристроїв, проміжний рівень edge-обчислень та вищий рівень хмарних сервісів. В якості апаратної платформи для кінцевого вузла обґрунтовано використання мікроконтролера ESP32, який підтримує сучасні стандарти зв'язку та механізми безпечного оновлення прошивок "по повітрю" (OTA) з можливістю автоматичного відкату (rollback). Для надійної комунікації застосовано гібридну мережеву модель: протокол MQTT для енергоефективної передачі телеметрії та протокол HTTP(S) для розгортання масивних пакетів оновлень.

На основі проведеного дослідження доведено, що симбіоз методології DevOps та технологій IoT формує надійний базис для безпечної та економічно ефективної

експлуатації розподілених інженерних систем.

КЛЮЧОВІ СЛОВА: ІНТЕРНЕТ РЕЧЕЙ (IOT), DEVOPS, ПРЕДИКТИВНЕ ОБСЛУГОВУВАННЯ, OTA-ОНОВЛЕННЯ, ESP32, MQTT, ХМАРНІ ОБЧИСЛЕННЯ, ТЕЛЕМЕТРІЯ.

ABSTRACT

The textual part of the bachelor's qualification paper: 72 pages, 15 figures, 4 tables, 15 sources.

Research Objective – The development of a comprehensive system for continuous monitoring, predictive forecasting of technical conditions, and the organization of remote updates for an IoT device fleet.

Research Focus – Processes of monitoring, predictive maintenance, and remote software updating within the Internet of Things infrastructure.

Subject of Inquiry – Hardware and software solutions, network communication protocols, and cloud DevOps architectures that enable the management of IoT systems.

Thesis Overview: In the context of global digitalization and the exponential growth of connected Internet of Things (IoT) devices, there is an acute need to ensure their uninterrupted operation. Traditional technical maintenance methodologies (reactive and preventive) are often economically and technically inefficient because they ignore the real-time condition of the equipment. Given this, the paper proposes a transition to predictive maintenance, which relies on the synergy of IoT technologies, Big Data, and machine learning.

To effectively manage the device lifecycle, the integration of DevOps methodology into the Internet of Things has been investigated. A three-tier architectural model of the system is proposed, comprising the base tier of end devices, the intermediate tier of edge computing, and the upper tier of cloud services. The use of the ESP32 microcontroller as the hardware platform for the end node is justified; it supports modern communication standards and mechanisms for secure over-the-air (OTA) firmware updates with an automatic rollback feature. A hybrid network model is utilized for reliable communication: the MQTT protocol for energy-efficient telemetry transmission and the HTTP(S) protocol for deploying massive update packages.

Based on the conducted research, it is proven that the symbiosis of DevOps methodology and IoT technologies establishes a solid foundation for the secure and cost-

effective operation of distributed engineering systems.

KEYWORDS: INTERNET OF THINGS (IOT), DEVOPS, PREDICTIVE MAINTENANCE, OTA UPDATE, ESP32, MQTT, CLOUD COMPUTING, TELEMETRY.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

ПОДАННЯ
ГОЛОВІ ДЕРЖАВНОЇ ЕКЗАМЕНАЦІЙНОЇ КОМІСІЇ
ЩОДО ЗАХИСТУ КВАЛІФІКАЦІЙНОЇ РОБОТИ
на здобуття освітнього ступеня бакалавра

Направляється здобувач Паламарчук Д. Є. до захисту кваліфікаційної роботи
(прізвище та ініціали)
за спеціальністю 123 Комп'ютерна інженерія
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)
на тему: «Система спостереження, актуалізації та предиктивного аналізу статусу пристроїв Інтернету речей».
Кваліфікаційна робота і рецензія додаються.

Директор ННІ _____ Катерина НЕСТЕРЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Висновок керівника кваліфікаційної роботи

Здобувач Паламарчук Д. Є. показав відмінну теоретичну підготовку, уміння володіти та розуміння новими комп'ютерними технологіями, користуватися навчальною і науково-технічною літературою. Працюючи над завданнями, які були поставлені керівником, проявив ініціативність, сумлінність та хист до інженерної роботи.
Все це дозволяє оцінити кваліфікаційну роботу студента Паламарчука Дмитра Євстаховича на оцінку «відмінно», та присвоїти йому кваліфікацію «бакалавр з комп'ютерної інженерії».

Керівник кваліфікаційної роботи _____ Віра БАБЕНКО
(підпис) *(Ім'я, ПРІЗВИЩЕ)*

«_» _____ 2026 року

Висновок кафедри про кваліфікаційну роботу

Кваліфікаційна робота розглянута. Здобувач *Паламарчук Д. Є.* допускається до захисту даної роботи в Екзаменаційній комісії.

Завідувач кафедрою Комп'ютерної інженерії _____ Наталія ЛАЩЕВСЬКА
(назва) *(підпис)* *(Ім'я, ПРІЗВИЩЕ)*

ВІДГУК РЕЦЕНЗЕНТА
на кваліфікаційну бакалаврську роботу

здобувача вищої освіти Паламарчук Дмитро Євстахович
(прізвище, ім'я, по батькові)
на тему: «Система спостереження, актуалізації та предиктивного аналізу статусу пристроїв Інтернету речей»

Актуальність:

Бакалаврська кваліфікаційна робота присвячена одній із найгостріших проблем сучасної комп'ютерної інженерії — забезпеченню безперебійної роботи, моніторингу та віддаленому оновленню інфраструктури Інтернету речей (IoT). В умовах, коли класичні реактивні та превентивні методи обслуговування обладнання виявляються неефективними та призводять до непередбачуваних простоїв, тема дослідження є надзвичайно актуальною. Автор пропонує сучасний підхід — перехід до предиктивного (прогнозувального) обслуговування на основі синергії технологій IoT, аналітики даних та практик DevOps, що повністю відповідає світовим трендам у розробці розподілених систем.

Позитивні сторони:
1. Автор вдало аргументував необхідність використання гібридної мережевої моделі, яка поєднує протокол MQTT для безперервного та енергоефективного потоку телеметрії із протоколом HTTP(S) для виконання ресурсоємних операцій, таких як завантаження нових релізів мікрокоду.
2. Автор продемонстрував високу компетентність у проєктуванні апаратно-програмного комплексу, обравши як обчислювальне ядро платформу ESP32.
3. Розроблено гнучку та відмовостійку модульну архітектуру програмної прошивки. Автор вдало імплементував підсистему проактивної самодіагностики, яка ефективно контролює використання системних ресурсів та здатна самостійно реагувати на виняткові ситуації.

Недоліки:
1. У роботі недостатньо уваги приділено питанню реального енергоспоживання запропонованого рішення під час тривалої автономної роботи.
2. Бажано було б розширити експериментальну базу, протестувавши розроблену хмарну архітектуру та брокери повідомлень під час стрес-тестування.

Висновок: *кваліфікаційна бакалаврська робота заслуговує оцінку " відмінно", а здобувач Паламарчук Д. Є. заслуговує присвоєння кваліфікації: бакалавра з комп'ютерної інженерії.*

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

ЗМІСТ

ВСТУП 11
1 ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ТА РІШЕНЬ ДЛЯ КОНТРОЛЮ, ПРОГРАМНОГО ОНОВЛЕННЯ Й ОЦІНКИ МАЙБУТНЬОГО СТАНУ IOT-СИСТЕМ 13
1.1 Дослідження проблематики контролю та технічної підтримки систем Інтернету речей 13
1.2 Побудова процесів DevOps для систем Інтернету речей з використанням новітніх методологій 20
1.3 Використання DevOps-рішень під час створення хмарних середовищ для обробки телеметрії IoT-обладнання 23
2 СТВОРЕННЯ КОМПЛЕКСНОЇ СИСТЕМИ КОНТРОЛЮ, ПРЕДИКТИВНОЇ АНАЛІТИКИ ТА ВІДДАЛЕНОЇ МОДЕРНІЗАЦІЇ ПРИСТРОЇВ ІНТЕРНЕТУ РЕЧЕЙ 31
2.1 Базова архітектура рішення та підбір програмного й апаратного забезпечення 31
2.2 Специфіка впровадження DevOps-практик у розподілені IoT-системи 39
2.3 Способи та програмно-апаратні рішення для збирання, накопичення і опрацювання телеметричної інформації 43
3 СТВОРЕННЯ ТА НАЛАГОДЖЕННЯ ПРОГРАМНИХ КОМПОНЕНТІВ IOT-МЕРЕЖІ Й АВТОМАТИЗОВАНИХ DEVOPS-ПЛАТФОРМ 51
3.1 Практичне створення апаратного забезпечення вузла Інтернету речей 51
3.2 Програмна конфігурація пристроїв Інтернету речей та граничних обчислювальних систем 56
ВИСНОВКИ 68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 70
Додаток А. 72

ВСТУП

В умовах глобальної цифровізації та експоненційного збільшення кількості підключених пристроїв Інтернету речей (IoT) у ключових секторах економіки виникає гостра необхідність забезпечення їхньої безперебійної роботи. Відповідно, проблематика гарантування високої надійності апаратних компонентів, організації цілодобового моніторингу їхнього стану та завчасного прогнозування потенційних технічних збоїв набуває статусу критично важливого завдання для сучасних інженерних систем.

Класичні методології технічного супроводу та експлуатації апаратного забезпечення здебільшого базуються на двох парадигмах: реактивній (відновлення працездатності після поломки) та превентивній (профілактичні роботи за розкладом). Проте обидва ці підходи мають суттєві недоліки, оскільки вони переважно ігнорують реальний, поточний технічний стан обладнання в конкретний момент часу. Як наслідок, такі стратегії часто призводять або до непередбачуваних простоїв, або до невиправданих витрат ресурсів на надмірне обслуговування цілком справних вузлів. Крім того, цільовим середовищем для розгортання програмного забезпечення в IoT виступає величезний, географічно розподілений парк фізичних пристроїв, що часто характеризуються жорстким дефіцитом обчислювальних потужностей та працюють на нестабільних мережах.

З огляду на ці обмеження, актуальні наукові та практичні дослідження дедалі частіше фокусуються на концепції предиктивного (прогнозувального) обслуговування. Цей інноваційний підхід дозволяє діяти на випередження завдяки глибокій аналітиці зібраних телеметричних даних. Для реалізації такої системи постає необхідність впровадження парадигми DevOps безпосередньо у сферу Інтернету речей, що забезпечує безшовне злиття процесів безперервного збору телеметрії, розгортання програмних оновлень (OTA) та постійного нагляду за станом пристроїв. Отже, розробка комплексної системи

спостереження, актуалізації та предиктивного аналізу є вкрай актуальним завданням, яке дозволить підвищити відмовостійкість та ефективність сучасних IoT-інфраструктур..

1 ОГЛЯД СУЧАСНИХ ТЕХНОЛОГІЙ ТА РІШЕНЬ ДЛЯ КОНТРОЛЮ, ПРОГРАМНОГО ОНОВЛЕННЯ Й ОЦІНКИ МАЙБУТНЬОГО СТАНУ ІОТ-СИСТЕМ

1.1 Дослідження проблематики контролю та технічної підтримки систем Інтернету речей

Ось оброблений, перефразований та розширений варіант твого тексту, адаптований під академічний стиль з дотриманням усіх твоїх вимог:

В умовах глобальної цифровізації та експоненційного збільшення кількості підключених пристроїв Інтернету речей (IoT) у ключових секторах економіки — таких як промислове виробництво, сільське господарство, концепції «розумних міст» (Smart City) та сфера медичного обслуговування — виникає гостра необхідність забезпечення їхньої безперебійної роботи. Відповідно, проблематика гарантування високої надійності апаратних компонентів, організації цілодобового моніторингу їхнього стану та завчасного прогнозування потенційних технічних збоїв набуває статусу критично важливого завдання для сучасних інженерних систем.

Класичні методології технічного супроводу та експлуатації апаратного забезпечення здебільшого базуються на двох парадигмах: реактивній (відновлення працездатності вже після фактичного настання поломки) та превентивній (профілактичні роботи, що проводяться за жорстко регламентованим розкладом). Проте обидва ці підходи мають суттєві недоліки, оскільки вони переважно ігнорують реальний, поточний технічний стан обладнання в конкретний момент часу. Як наслідок, такі стратегії часто виявляються економічно та технічно неефективними, призводячи або до непередбачуваних простоїв, або до невиправданих витрат ресурсів на надмірне обслуговування цілком справних вузлів.

З огляду на ці обмеження, актуальні наукові та практичні дослідження дедалі частіше фокусуються на концепції предиктивного (прогнозувального) обслуговування. Цей інноваційний підхід базується на синергії технологій Інтернету речей, методів глибокої аналітики великих даних (Big Data) та передових алгоритмів машинного навчання, що дозволяє діяти на випередження [1].

Процес моніторингу в інфраструктурі IoT є багатоетапним. На початковому етапі відбувається безперервний збір різноманітних телеметричних даних з датчиків, серед яких ключовими є температурні режими, рівні вібрації, параметри споживання електроенергії, а також якість та потужність мережевого сигналу. Наступними кроками є безпечна передача зібраного масиву інформації, його надійне збереження у базах даних, подальша математична обробка та наочна візуалізація для операторів. Поняття «обслуговування» у цій екосистемі набуває ширшого значення: воно охоплює не лише фізичний ремонт, а й дистанційне оновлення програмного забезпечення (firmware), а також ініціацію автоматизованих сценаріїв реагування при фіксації будь-яких відхилень від норми. Своєю чергою, прогнозування стану обладнання полягає в аналізі історичних даних для точного передбачення процесів зношення матеріалів або повної відмови системи ще до того, як вони вплинуть на робочий процес.

Детальний аналіз сучасного наукового дискурсу та останніх галузевих публікацій свідчить про те, що еволюція систем нагляду та підтримки IoT-мереж відбувається переважно у трьох ключових, тісно взаємопов'язаних напрямках:

– проєктування та оптимізація багаторівневих архітектур для ефективного збору, агрегації та глибокого аналізу телеметричної інформації;

– розробка та вдосконалення математичних моделей і алгоритмів, здатних з високою точністю прогнозувати ймовірність виникнення несправностей;

– впровадження надійних автоматизованих інструментів для дистанційного оновлення мікропрограмного забезпечення "по повітрю" (Over-The-Air, OTA), що базуються на сучасних практиках та філософії DevOps.

Стандартна топологія архітектури для систем моніторингу зазвичай будується за трирівневою ієрархічною моделлю, структуру якої наочно продемонстровано на рис. 1.1. До базових рівнів цієї екосистеми входять:

– початковий або периферійний рівень (edge-рівень), що складається безпосередньо з кінцевих IoT-пристроїв та сенсорів, які відповідають за первинну реєстрацію фізичних величин навколишнього середовища;

– проміжний рівень, представлений мережевими шлюзами та локальними контролерами, функція яких полягає у зборі, первинній фільтрації та подальшій маршрутизації агрегованих даних;

– вищий рівень — централізована хмарна інфраструктура, яка бере на себе завдання довгострокового зберігання масивів інформації, виконання складних аналітичних обчислень та надання інструментів візуалізації.

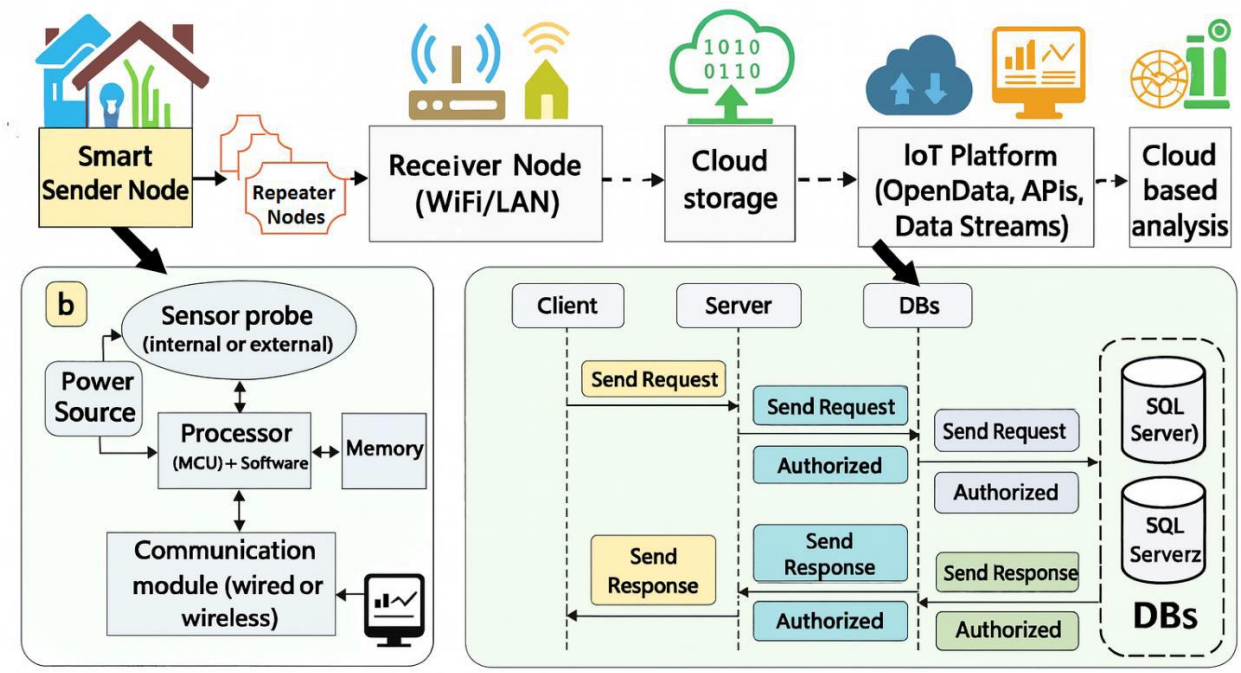


Рисунок 1.1 — Типова структурна схема та ієрархічні рівні моніторингової IoT-системи

Аналізуючи схему, наведену на рис. 1.1, можна детальніше простежити розподіл функцій. Периферійний (edge) рівень охоплює різноманітні датчики та мікроконтролери. Рівень туманних обчислень (fog-рівень), розташований ближче до джерела даних, бере на себе завдання з агрегації потоків інформації та її попередньої обробки, що дозволяє зменшити обсяги трафіку. Водночас хмарний (cloud) рівень виступає глобальним ядром системи, акумулюючи дані для глибинної аналітики, розпізнавання патернів та графічного подання результатів.

Подібні архітектурні концепції ґрунтовно досліджувалися у наукових працях таких вчених, як J. Wettinger, F. Leymann та S. Dustdar. У своїх дослідженнях вони висунули та обґрунтували інноваційну ідею інтеграції методології DevOps безпосередньо у сферу Інтернету речей. Головна суть цієї концепції полягає у безшовному злитті раніше розрізнених процесів: безперервного збору телеметрії, розгортання програмних оновлень та постійного нагляду за станом пристроїв, що формує єдиний життєвий цикл управління системою [2].

Аналогічні підходи до організації мережевої взаємодії пропонують дослідники M. Aazam та A. Rahmani. Їхні роботи роблять особливий акцент на критичній важливості впровадження парадигм туманних (fog) та граничних (edge) обчислень. Перенесення частини обчислювальних потужностей ближче до місця генерації даних дозволяє радикально мінімізувати мережеві затримки, а також суттєво розвантажити магістральні канали зв'язку та центральні хмарні сервери [3-4].

Однією з найбільш вагомих та перспективних тенденцій розвитку галузі протягом останніх років став парадигмальний зсув від застарілої моделі реактивного усунення несправностей на користь проактивного, інтелектуального прогнозувального обслуговування [5-7].

В рамках таких сучасних екосистем мережа розподілених сенсорів здійснює перманентний моніторинг ключових показників: температурних флуктуацій, механічних вібрацій, динаміки енергоспоживання, стабільності

радіосигналу тощо. Зібраний масив "сирих" даних безперервно подається на вхід алгоритмам машинного навчання. Завдяки здатності цих моделей виявляти приховані закономірності, система здатна передбачити неминучість поломки вузла. Це дає можливість завчасно спланувати сервісні роботи, не перериваючи основний процес.

Вагомі результати в цьому напрямку були продемонстровані в наукових дослідженнях [8,9], де розглядалася практична ефективність використання ансамблевого алгоритму AdaBoost. Застосування цієї моделі для ідентифікації ранніх маркерів деградації компонентів промислового устаткування дозволило підвищити точність прогнозування відмов до 90 %.

Окремої уваги заслуговує дослідження [10], автори якого презентували підхід Software-Defined Sensing. Ця концепція пропонує механізм уніфікації гетерогенних потоків даних від датчиків різних типів, що значно підвищує загальну чутливість платформи та дозволяє їй швидше ідентифікувати нештатні ситуації в інфраструктурі.

Специфіка телеметрії в IoT-системах полягає в тому, що дані представляють собою часові ряди. Для ефективної роботи з таким типом інформації дедалі частіше застосовуються архітектури рекурентних нейронних мереж (RNN, зокрема LSTM). Їхня головна перевага — здатність враховувати складну динаміку зміни фізичних параметрів у часі.

Водночас, як підкреслюється в наукових розвідках [11,12], впровадження надзвичайно складних нейромережевих моделей часто є недоцільним на рівні кінцевих вузлів через їхню обмежену енергоємність. Зважаючи на це, на передній план виходить завдання розробки оптимізованих, «легких» аналітичних моделей з функцією адаптивного перенавчання. Концептуальну схему такої архітектури, що ілюструє процеси обробки інформаційних потоків, наведено на рис. 1.2.

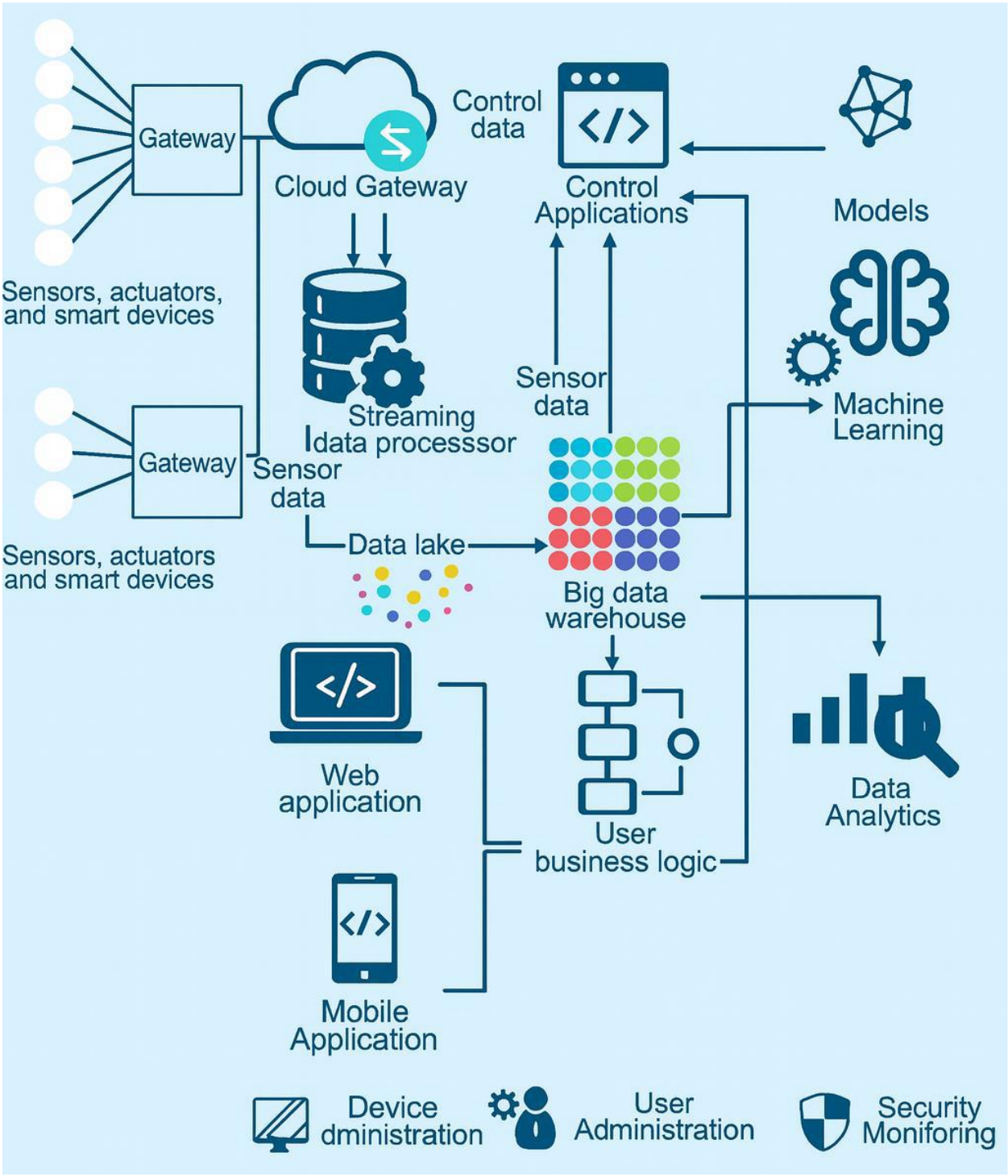


Рисунок 1.2 — Архітектурна модель системи з деталізацією потоків даних та компонентів обробки

Відповідно до структури, зображеної на рис. 1.2, екосистема складається з низки базових компонентів та інформаційних потоків, серед яких:

— Сенсори, актуатори та апаратні засоби: фізичний базис системи, що безпосередньо взаємодіє з середовищем, генеруючи телеметрію або приймаючи команди керування;

— Мережевий шлюз (Gateway): проміжний вузол зв'язку, який слугує мостом для з'єднання локального обладнання із глобальними хмарними платформами;

— Модулі Cloud Gateway та Streaming data processor: хмарні компоненти, що відповідають за безперебійне приймання потокових даних у режимі реального часу та їх попередню фільтрацію;

— Сховища даних (Data lake та Big data warehouse): масштабовані бази для безпечного та довгострокового накопичення великих обсягів історичної телеметрії;

— Модулі штучного інтелекту (AI): аналітичні компоненти, які застосовують машинне навчання для глибокого аналізу даних та генерації прогнозів стану обладнання;

— Сервіси Data Analytics та Control Applications: прикладне програмне забезпечення, що забезпечує візуалізацію бізнес-метрик, аналітику та надає інструменти для дистанційного управління пристроями;

— Web / Mobile Application: клієнтська частина, що представляє інтуїтивно зрозумілі користувацькі інтерфейси для доступу до функціоналу системи;

— Device/User Administration та Security Monitoring: підсистеми адміністрування, які керують правами доступу, життєвим циклом пристроїв та гарантують безпеку платформи.

1.2 Побудова процесів DevOps для систем Інтернету речей з використанням новітніх методологій

У реаліях сучасної комп’ютерної інженерії методологія DevOps закріпила за собою статус фундаментального підходу, що забезпечує глибоку інтеграцію процесів розробки, всебічного тестування, безперебійного розгортання та подальшого технічного супроводу програмного забезпечення. Ключова мета цієї концепції зводиться до формування та підтримки безперервного життєвого циклу будь-якого програмного продукту, що досягається шляхом впровадження високого рівня автоматизації рутинних операцій та суворого, багаторівневого контролю якості на кожному з етапів створення системи.

Коли йдеться про складні екосистеми Інтернету речей (IoT), які можуть об'єднувати у єдину мережу тисячі, а подекуди й мільйони географічно розподілених смарт-пристроїв, застосування практик DevOps набуває критичної значущості. У цьому контексті перед інженерами постає комплексне завдання: потрібно не просто регулярно оновлювати мікропрограмне забезпечення на різнорідних апаратних вузлах, а й беззаперечно гарантувати загальну стабільність функціонування всієї інфраструктури. Крім того, вкрай важливо забезпечити абсолютну безпеку та цілісність пакетів оновлень, що доставляються на пристрої у режимі реального часу, мінімізуючи ризики збоїв[13].

Традиційна парадигма DevOps складається з логічної послідовності взаємопов'язаних етапів, що формують замкнений циклічний конвеєр безперервної інтеграції та безперервного розгортання, відомий як CI/CD (Continuous Integration / Continuous Deployment). Детальну архітектурну структуру цього конвеєрного процесу, адаптовану під вимоги сучасних технологічних підходів, наочно проілюстровано на рис. 1.3.



Рисунок 1.3 — Концептуальна модель та ітеративний цикл DevOps для інфраструктури Інтернету речей з огляду на новітні методології

Спираючись на логіку побудови, що зображена на рис. 1.3, можна виділити такі фундаментальні складові процесу DevOps:

Continuous Integration (CI, Безперервна інтеграція) — етап, що передбачає регулярне та систематичне злиття розробленого програмного коду до спільного репозиторію. Цей процес супроводжується обов'язковою автоматичною перевіркою сумісності внесених модифікацій та запуском серій тестів для раннього виявлення дефектів.

Continuous Deployment (CD, Безперервне розгортання) — фаза, відповідальна за повністю автоматизовану доставку та інсталяцію валідованих оновлень безпосередньо у цільове середовище експлуатації без необхідності ручного втручання.

Monitoring (Моніторинг) — процес безперервного та всебічного нагляду за технічним станом системи в цілому та кожного окремого IoT-вузла зокрема, з особливим фокусом на показники після розгортання нових версій програмного забезпечення.

Feedback (Зворотний зв’язок) — критично важливий механізм збору та глибокого аналізу метрик, який надає розробникам можливість оперативно реагувати на виникнення інцидентів, виправляти програмні помилки та безперервно оптимізувати архітектуру проєкту.

Якщо розглядати розробку класичних веб-сервісів чи монолітних серверних додатків, то організація конвеєра CI/CD зазвичай відбувається у стандартизованих умовах — на базі хмарних платформ або в межах контрольованих локальних центрів обробки даних [13]. Однак, у специфічному середовищі Інтернету речей ця задача суттєво ускладнюється. Головна проблема полягає в тому, що кінцеве розгортання здійснюється не на централізованому потужному сервері, а на величезному парку фізичних апаратних засобів. Такі пристрої, як правило, характеризуються жорстко обмеженими обчислювальними ресурсами, змушені працювати в умовах низької пропускної здатності комунікаційних каналів і часто страждають від нестабільного мережевого з'єднання.

Імплементація практик DevOps у сферу IoT має низку унікальних особливостей, зумовлених специфікою архітектури таких мереж. Першим ключовим фактором є яскраво виражена гібридність операційного середовища. Вона проявляється у тому, що ресурсоємні процеси, такі як оркестрація оновлень та масштабна аналітика, виконуються на рівні хмарних сервісів. Натомість інша частина завдань — збір первинної телеметрії та базова обробка — реалізується безпосередньо на периферії силами самих edge-пристроїв[14].

Наупний критичний аспект стосується жорстких вимог до кібербезпеки. З огляду на специфіку IoT, виникає категорична необхідність обов'язкового криптографічного підпису програмних пакетів цифровими сертифікатами під

час кожної ітерації оновлення. Такий захід гарантує суворий контроль над розповсюдженням коду та слугує надійним захистом від несанкціонованих модифікацій базового мікрокоду (firmware).

Третя суттєва відмінність полягає в обов'язковій підтримці механізмів бездротового оновлення "по повітрю" (Over-The-Air, OTA). Система має бути спроєктована так, щоб стовідсотково забезпечувати можливість віддаленої трансляції та інсталяції найсвіжіших релізів прошивок на всі пристрої парку, повністю виключаючи потребу у фізичному доступі персоналу до обладнання.

Крім того, невід'ємною характеристикою DevOps-інженерії для IoT-середовищ є постійна адаптація до апаратних обмежень пристроїв. Оскільки мікроконтролери мають лімітований обсяг пам'яті, з'являється гостра потреба максимально скорочувати розмір файлів оновлення. Найефективнішим рішенням є використання диференційних пакетів (delta updates), коли завантажується не весь образ прошивки, а виключно той фрагмент коду, який зазнав змін.

Нарешті, повноцінний DevOps-конвеєр для управління IoT-інфраструктурою неможливий без організації надійного каналу зворотної телеметрії. Завдяки цьому, одразу після завершення процедури оновлення, кожен пристрій автоматично відправляє на центральний сервер звіт про свій технічний стан. Цей потік даних надає розробникам об'єктивну картину для аналізу ефективності розгортання та швидкого виявлення потенційних аномалій.

1.3 Використання DevOps-рішень під час створення хмарних середовищ для обробки телеметрії IoT-обладнання

Експоненційне зростання екосистеми Інтернету речей (IoT) неминуче призвело до колосального збільшення масивів даних, які безперервно

генеруються розгалуженими сенсорними мережами, різноманітними вбудованими системами та інтелектуальним периферійним обладнанням. Класичні локальні центри обробки даних (on-premise інфраструктури) все частіше стикаються з критичними обмеженнями: вони виявляються неспроможними оперативно опрацьовувати такі гігантські потоки інформації (Big Data), а також не можуть гарантувати необхідний рівень динамічної масштабованості та цілодобової доступності критично важливих сервісів. З огляду на ці виклики, домінуючу роль у проєктуванні сучасної IoT-архітектури переймають хмарні обчислення (Cloud Computing). Саме синергія хмарних технологій та передових методологій DevOps дозволяє створити максимально гнучке, стійке до відмов та повністю автоматизоване середовище для збору, маршрутизації та глибинної обробки телеметричної інформації[15].

Застосування хмарних платформ надає інженерам можливість налагодити централізоване та безпечне зберігання, складний багатовимірний аналіз і інтерактивну візуалізацію телеметрії, що надходить від IoT-вузлів, повністю уникаючи колосальних капітальних витрат на проєктування та обслуговування власних апаратних серверних потужностей. У цій парадигмі всі інформаційні потоки передаються та опрацьовуються через глобальну мережу Інтернет. Це гарантує безперешкодний доступ до результатів аналітики з будь-якої точки світу, незалежно від типу клієнтського пристрою — чи то спеціалізовані мобільні додатки, чи віддалені edge-контролери на виробництві. Завдяки такій універсальності, розподілена хмарна інфраструктура закріпила за собою статус фундаментального базису для побудови сучасних IoT-рішень, орієнтованих на процеси DevOps.

На сучасному етапі розвитку IT-індустрії сформувалося кілька парадигм надання хмарних ресурсів. У світовій практиці загальноприйнято виділяти три фундаментальні сервісні моделі. Кожна з них характеризується чітко розмежованими зонами відповідальності, ступенем адміністрування та рівнем контролю, що розподіляються між безпосереднім провайдером послуг та кінцевим користувачем (схематично цей розподіл відображено на рис. 1.4):

— Інфраструктура як послуга (Infrastructure as a Service, IaaS);

— Платформа як послуга (Platform as a Service, PaaS);

— Програмне забезпечення як послуга (Software as a Service, SaaS).

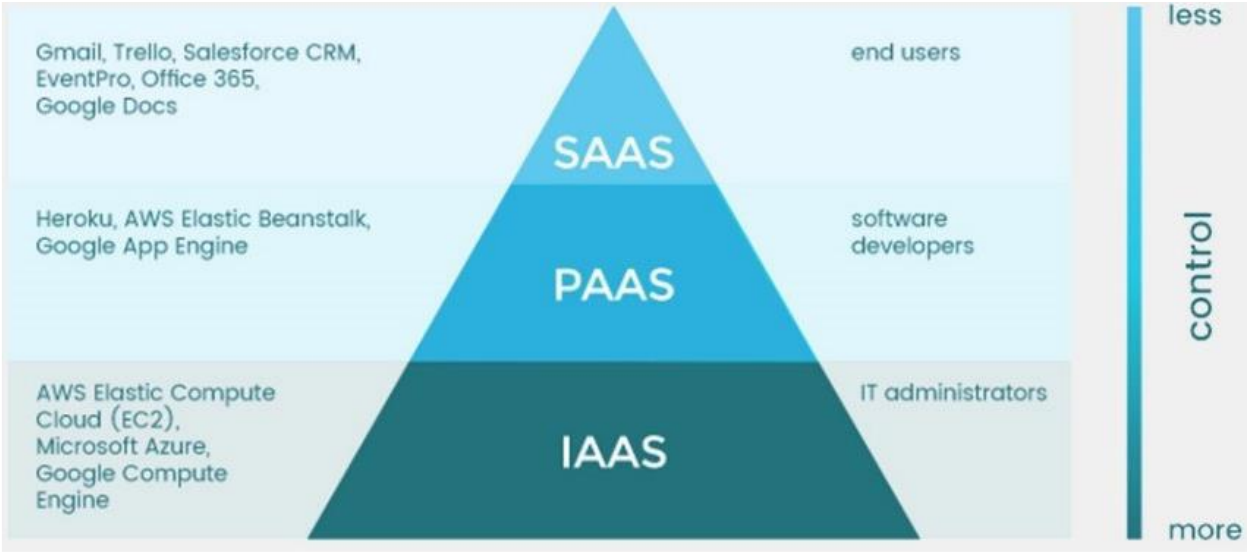


Рисунок 1.4 — Класифікація базових моделей надання хмарних сервісів

Важливо зазначити, що кожна із зазначених архітектурних моделей має власну специфіку інтеграції в існуючі DevOps-конвеєри. Вибір конкретного підходу для опрацювання масивів телеметрії від IoT-обладнання завжди є індивідуальним і диктується унікальними проєктними вимогами щодо необхідної глибини адміністративного контролю, бажаної операційної гнучкості та очікуваних масштабів розширення системи.

Сервісна модель IaaS базується на принципі надання замовнику пулу віртуалізованих обчислювальних потужностей. Сюди входять віртуальні сервери, налаштовані мережеві топології, системи зберігання даних та гіпервізори, які виділяються динамічно за моделлю «на вимогу». У цьому сценарії інженерна команда отримує максимально повний доступ до керування гостьовими операційними системами, інстальованим програмним забезпеченням та брандмауерами. Водночас усі турботи щодо підтримки працездатності апаратного забезпечення (фізичної інфраструктури) повністю лягають на хмарного провайдера.

Як проілюстровано на рис. 1.5, парадигма IaaS вважається еталоном гнучкості з перспективи імплементації DevOps-практик. Її архітектура відкриває широкі можливості для:

— повноцінної реалізації концепції «Інфраструктура як код» (Infrastructure as Code, IaC);

— високорівневої автоматизації процесів швидкого розгортання робочих середовищ;

— проєктування та побудови еластичних, високопродуктивних кластерів, здатних ефективно обробляти великі потоки IoT-даних;

— організації тимчасових тестових або ізольованих експериментальних середовищ для перевірки гіпотез.

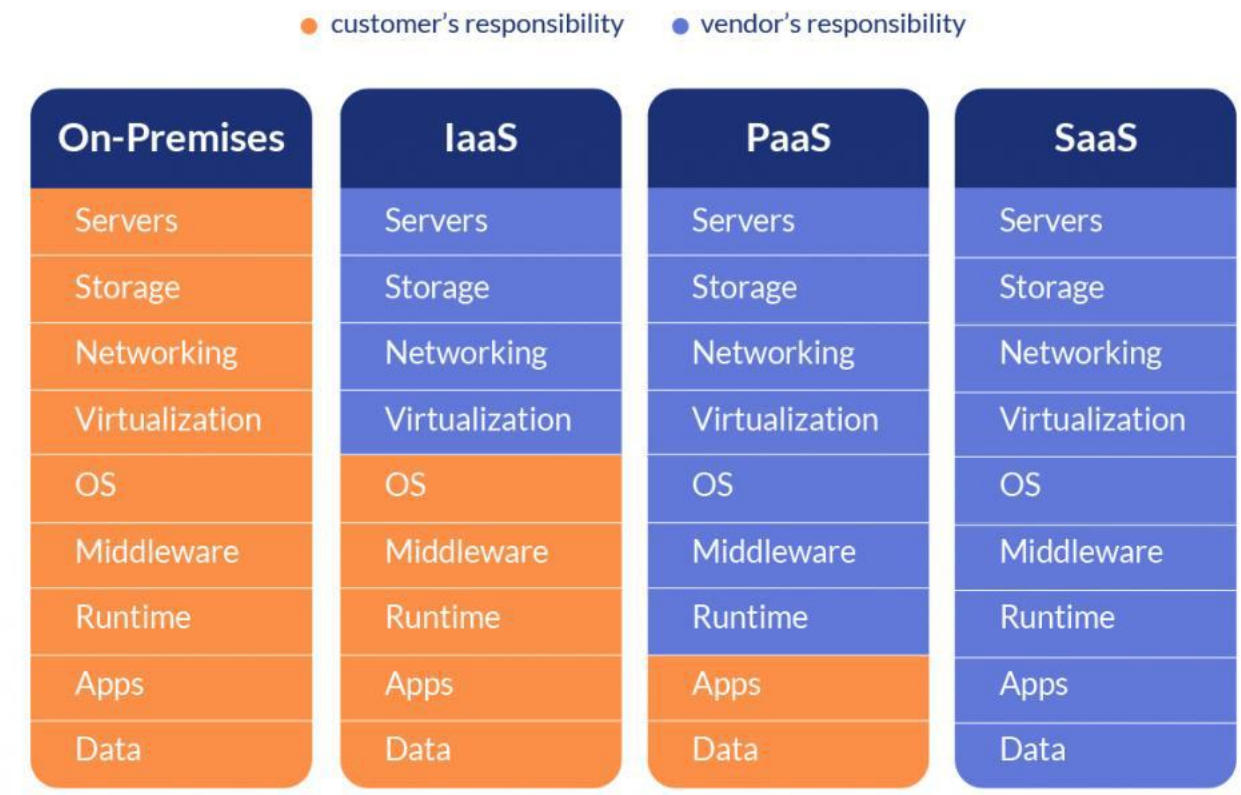


Рисунок 1.5 — Характерні особливості та переваги хмарних обчислювальних моделей

На сучасному ринку домінуючими представниками IaaS-рішень є глобальні платформи: Amazon Web Services (зокрема EC2), Microsoft Azure

Virtual Machines та Google Compute Engine. Саме ця інфраструктурна модель найчастіше застосовується як фундамент для аналізу великих обсягів телеметрії, розгортання систем машинного навчання та побудови алгоритмів прогнозування технічного стану IoT-обладнання.

Проте, незважаючи на очевидні переваги, підхід IaaS має і певні обмеження. До них належать необхідність самостійного адміністрування парку операційних систем, складність налаштування багаторівневих політик безпеки, а також потреба у ретельному контролі фінансових витрат під час пікових навантажень на систему. Загальну структурну логіку та базовий сценарій застосування IaaS для обробки IoT-даних продемонстровано на рис. 1.6.

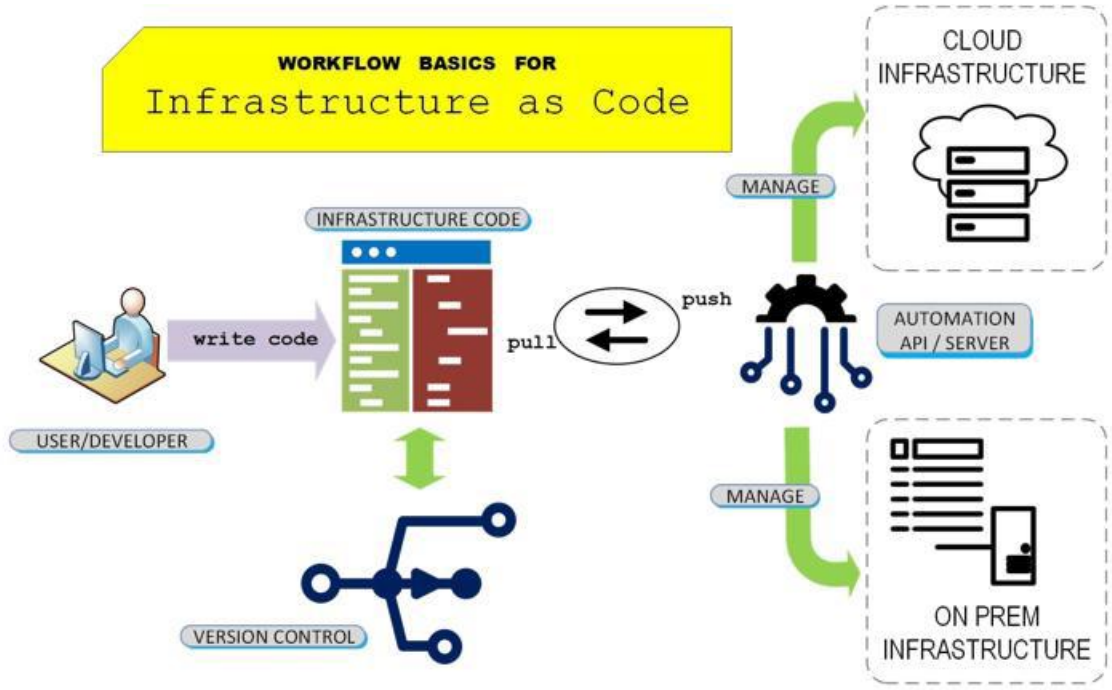


Рисунок 1.6 — Типова схема застосування інфраструктурної моделі (IaaS) в контексті IoT

Наступна модель — PaaS — концептуально спрямована на спрощення та прискорення життєвого циклу розробки і розгортання прикладних систем. У цьому випадку постачальник послуг надає не лише базову інфраструктуру, але й постачає налаштоване середовище виконання, необхідне проміжне

програмне забезпечення (middleware) та інтегровані набори інструментів для програмістів.

Для розробників систем Інтернету речей застосування PaaS є привабливим завдяки:

— швидкому створенню та запуску сервісів для збору і потокової обробки даних;

— значно спрощеній інтеграції з різноманітними API;

— нативній підтримці широкого спектра сучасних мов програмування;

— радикальному зменшенню операційного навантаження на команду DevOps-інженерів.

До типових рішень класу PaaS належать Google App Engine, платформа Heroku та AWS Elastic Beanstalk. Вони часто стають основою для розроблення інтерактивних адміністративних веб-панелей, проєктування REST-API та сервісів управління пристроями.

Головним недоліком PaaS є дещо обмежений контроль над глибинними налаштуваннями інфраструктури та ризик залежності від технологій конкретного постачальника (vendor lock-in). Це може стати перешкодою при спробах інтеграції з нестандартними протоколами або застарілими промисловими IoT-рішеннями.

Модель SaaS (схему наведено на рис. 1.7) передбачає надання клієнту доступу до вже повністю готового програмного продукту, який централізовано розміщується та обслуговується провайдером. Користувач взаємодіє з функціоналом виключно через веб-інтерфейс або API-шлюзи, абсолютно не витрачаючи час на адміністрування інфраструктурної частини.

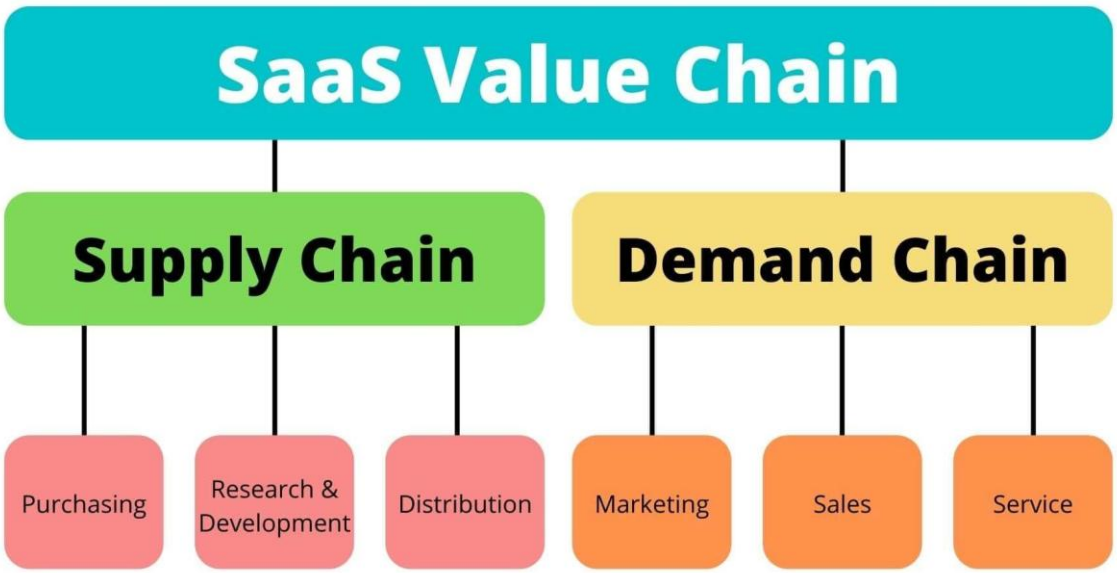


Рисунок 1.7 — Структурна модель "Програмне забезпечення як послуга" (SaaS)

У контексті розгортання IoT-мереж, готові SaaS-платформи найчастіше використовуються для:

— наочної візуалізації зрізів телеметричних даних;

— організації високорівневого моніторингу за загальним станом пристроїв;

— забезпечення середовища для спільної роботи команд та проведення глибокої аналітики.

Прикладами продуктів SaaS є Google Workspace, JIRA, Dropbox, Grafana Cloud, а також численні спеціалізовані IoT-платформи. Основною перевагою цього підходу є мінімальні вимоги до технічного обслуговування з боку замовника. Однак при цьому користувач повністю залежить від стабільності серверів провайдера та його політик надання послуг.

Комплексне поєднання потужностей хмарних сервісів із передовими підходами DevOps дозволяє:

— максимально автоматизувати цикли обробки телеметрії;

— гарантувати безперервне та безпечне оновлення всіх компонентів системи;

— реалізувати еластичне масштабування ресурсів пропорційно до поточного навантаження;

— нативно інтегрувати інструменти проактивного моніторингу, логування подій та аналітики.

Таким чином, вибір оптимальної моделі між IaaS, PaaS та SaaS під час проєктування DevOps-орієнтованих IoT-систем не має єдиного стандарту і залежить від балансу між бажаним рівнем контролю, вимогами до гнучкості розробки та складністю архітектури обробки даних.

2 СТВОРЕННЯ КОМПЛЕКСНОЇ СИСТЕМИ КОНТРОЛЮ, ПРЕДИКТИВНОЇ АНАЛІТИКИ ТА ВІДДАЛЕНОЇ МОДЕРНІЗАЦІЇ ПРИСТРОЇВ ІНТЕРНЕТУ РЕЧЕЙ

2.1 Базова архітектура рішення та підбір програмного й апаратного забезпечення

Процес проєктування високоефективної системи для безперервного моніторингу, предиктивного прогнозування технічного стану та організації віддаленого оновлення парку IoT-пристроїв вимагає розробки комплексної, багатовимірної архітектури. Ця структура повинна органічно та безшовно інтегрувати в єдину екосистему апаратні засоби, програмні комплекси та мережеві комунікаційні протоколи. Головна вимога до такої архітектури полягає у забезпеченні стабільного збору телеметричних показників, їхньої миттєвої обробки в режимі реального часу (real-time processing), а також підтримці надійних механізмів оновлення мікропрограмного забезпечення "по повітрю" (Over-The-Air, OTA). Крім того, архітектурне рішення має бути достатньо гнучким для швидкого масштабування інфраструктури. Не менш важливим є суворе дотримання критеріїв якості: система повинна демонструвати високу відмовостійкість, експлуатаційну надійність та гарантувати абсолютну безпеку всіх комунікаційних каналів від несанкціонованого втручання.

У контексті сучасної DevOps-парадигми, будь-яка повноцінна платформа для нагляду за станом Інтернету речей концептуально поділяється щонайменше на три ключові функціональні ієрархічні рівні.

Базовим фундаментом є рівень кінцевих пристроїв (Device Layer). До його складу входять різноманітні фізичні сенсори, первинні перетворювачі, мікроконтролери, мікрокомп’ютери та виконавчі актуатори. Саме на цьому

етапі відбувається безпосередня взаємодія з середовищем, генеруються інформаційні пакети телеметрії та здійснюється їхня найпростіша первинна обробка.

Другою ланкою виступає периферійний рівень (Edge Layer). Його головна місія — агрегація вхідних інформаційних потоків, проміжна буферизація, проведення попередньої аналітики та фільтрації. Цей рівень також відповідає за інтелектуальний розподіл обчислювального навантаження та локальну обробку критичних подій, що є вкрай важливим для суттєвого зменшення обсягів мережевого трафіку та мінімізації комунікаційних затримок.

Третій, найвищий рівень, базується на потужностях хмарних сервісів (Cloud Layer). Він бере на себе завдання з довгострокового зберігання масивів історичної телеметрії, надає потужний інструментарій для глибокого багатофакторного аналізу та акумулює ресурси для тренування математичних моделей прогнозування. Крім того, саме хмарне середовище є ядром для оркестрації DevOps-процесів, централізованого управління розгортанням оновлень та візуалізації аналітики.

У рамках даного дослідження розроблено та запропоновано комплексну архітектурну модель платформи моніторингу і прогнозування, що ґрунтується на застосуванні практик та інструментарію DevOps. Детальну структурну схему цієї екосистеми наочно проілюстровано на рис. 2.1.



Рисунок 2.1 — Концептуальна багаторівнева структура IoT-системи, орієнтованої на методологію DevOps

Безпосередній збір метрик, вимірювання фізичних величин цільових об'єктів та фіксація параметрів середовища покладається на кінцеві IoT-пристрої. Враховуючи, що проєктована система висуває жорсткі вимоги до постійного моніторингу та безперебійної підтримки процесу OTA-оновлень, процедура вибору апаратної бази має спиратися на чітко визначений перелік критеріїв:

— гарантована підтримка сучасних стандартів зв'язку (Wi-Fi, Ethernet або стільникових мереж LTE);

— наявність достатнього обсягу енергонезалежної пам'яті, необхідної для безпечного завантаження та розпакування нових версій прошивок;

— високий рівень загальної енергоефективності;

— нативна програмна підтримка легковагових протоколів передачі даних (зокрема, MQTT або CoAP);

— достатня обчислювальна потужність для функціонування в edge-режимі.

Аналіз існуючого ринку дозволяє виділити низку популярних рішень. Типові варіанти апаратних платформ для реалізації кінцевих вузлів наведено у табл. 2.1.

Таблиця 2.1 — Поширені апаратні платформи для базового рівня IoT-інфраструктури

Платформа	Основні переваги	Ключові обмеження	Сфери цільового застосування
ESP32	Економічність, вбудовані модулі Wi-Fi/BLE, підтримка OTA, широка екосистема	Лімітований обсяг пам'яті	Базові сенсорні вузли, енергоефективні (low-power) рішення
Raspberry Pi 4/5	Підтримка ОС Linux, запуск Docker-контейнерів, ресурси для ML на edge-рівні	Досить високий рівень енергоспоживання	Локальні мережеві шлюзи, вузли глибокої edge-аналітики
STM32 + LoRaWAN	Ультранизьке споживання енергії, висока промислова стійкість	Підвищена складність розробки програмного забезпечення	Віддалені автономні датчики, розподілені мережі

NXP i.MX	Еталонна промислова надійність (industrial grade), стабільна робота з Linux	Значна вартість компонентів	Промислові логічні контролери, критична інфраструктура

Рівень граничних обчислень (Edge-рівень) виступає стратегічно важливим компонентом у загальній DevOps-архітектурі. Його впровадження дозволяє радикально оптимізувати роботу системи шляхом зменшення навантаження на хмарні сервери. Edge-вузли здатні самостійно здійснювати первинну діагностику та контроль працездатності підключених до них датчиків. Не менш важливою функцією цього рівня є локальна маршрутизація даних, що гарантує автономність функціонування обладнання навіть у випадках деградації якості зв'язку з глобальною мережею.

Проєктування мережевих шлюзів для Edge-рівня вимагає дотримання низки умов. Зокрема, вони повинні підтримувати запуск Docker-контейнерів, локальне розгортання брокерів повідомлень (MQTT), функціонування баз даних часових рядів (Time-Series DB) для буферизації, а також мати механізми транзитної переадресації OTA-оновлень. До платформ, що повною мірою відповідають цим критеріям, належать:

— спеціалізовані модулі Raspberry Pi Compute Module (CM4/CM5) в індустріальному виконанні;

— мікрокомп'ютери серії NVIDIA Jetson (Nano або Xavier NX) для запуску нейромережевих моделей безпосередньо на периферії;

— спеціалізовані промислові шлюзи сімейства Advantech Industrial Gateways;

— високонадійні контролери лінійки Siemens IoT2000 Series.

Повноцінне функціонування хмарного рівня, який є логічним центром системи, спирається на інтеграцію комплексу спеціалізованих інструментальних засобів. Архітектура цього рівня передбачає наявність

брокера повідомлень, масштабованих сховищ для централізованого збереження телеметрії, повноцінного автоматизованого DevOps-конвеєра (CI/CD) та менеджера дистанційного оновлення прошивок. Додатково тут розгортаються математичні моделі прогнозування, що базуються на алгоритмах машинного навчання, а також надаються веб-панелі та відкриті API.

Оптимальний стек сучасних технологій, що забезпечує виконання цих завдань, деталізовано у табл. 2.2.

Таблиця 2.2 — Рекомендований технологічний стек та інструментарій для хмарного рівня

№ з/п	Функціональний напрямок (Технологія)	Популярні інструментальні рішення
1	Брокери повідомлень (MQTT)	EMQX Cloud, Eclipse Mosquitto, AWS IoT Core
2	Бази даних для часових рядів	InfluxDB Cloud, TimescaleDB, Amazon Timestream
3	Сховища "Озера даних" (Data Lake)	Amazon S3, MinIO Object Storage
4	Інструментарій DevOps (CI/CD)	GitHub Actions, GitLab CI/CD, Jenkins
5	Управління оновленнями (OTA)	Mender.io, BalenaCloud, Eclipse hawkBit
6	Моніторинг та візуалізація	Prometheus у зв'язці з Grafana
7	Потокова обробка та аналітика	Apache Spark, Apache Flink, Kafka Streams

Для забезпечення безперебійної та захищеної взаємодії між усіма архітектурними рівнями системи виникає необхідність у побудові стабільних каналів зв'язку. Це передбачає впровадження таких рішень:

— використання протоколу MQTT поверх криптографічних стандартів TLS 1.2 або 1.3 для надійного шифрування трафіку;

— організація захищених віртуальних приватних мереж (VPN-тунелів) для edge-шлюзів;

— логічна сегментація мережевого простору з використанням VLAN для ізоляції критичних компонентів;

— гнучке налаштування рівнів якості обслуговування (QoS 0, 1 або 2 для MQTT) залежно від ступеня важливості даних.

Крім того, при активному використанні механізмів edge-кешування першочерговим завданням стає забезпечення гарантованої доставки повідомлень та впровадження алгоритмів захисту від втрати чи дублювання інформації. Візуалізацію взаємодії всіх описаних компонентів наведено на рис. 2.2.

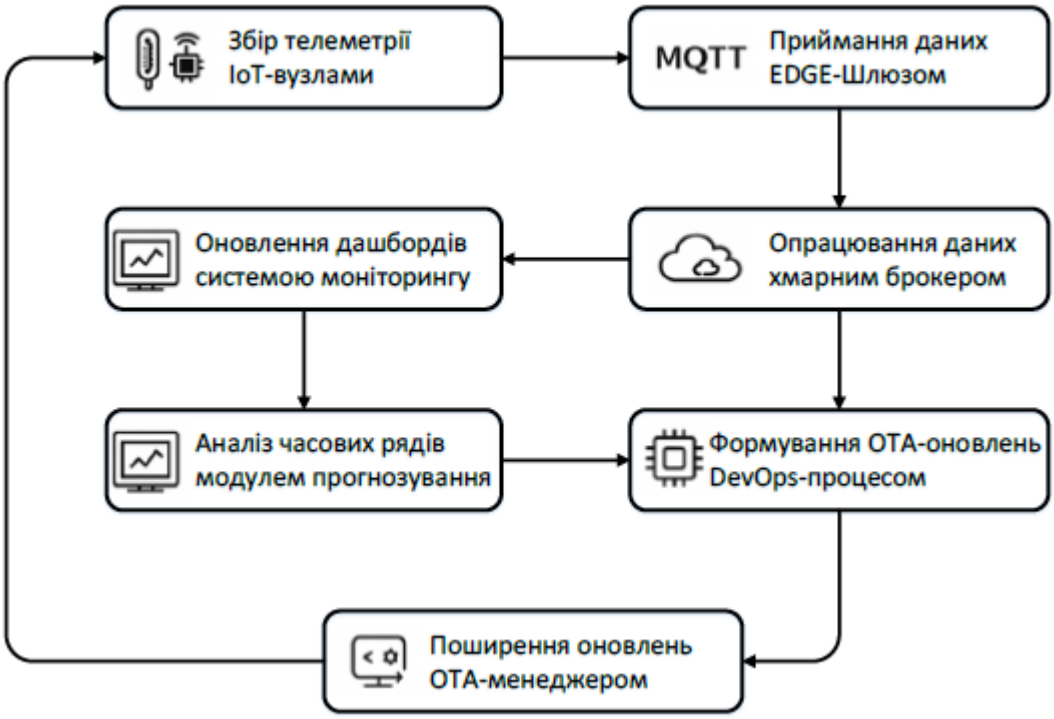


Рисунок 2.2 — Структурна схема взаємодії компонентів платформи моніторингу та прогнозування стану IoT-обладнання

Алгоритм функціонування даної архітектурної моделі можна представити у вигляді такої послідовності взаємопов'язаних етапів:

1. Кінцеві IoT-вузли здійснюють збір показників телеметрії і транслюють їх за допомогою протоколу MQTT.

2. Локальний Edge-шлюз перехоплює потоки, проводить їхню попередню аналітику, тимчасове кешування та подальшу маршрутизацію.

3. Центральний хмарний брокер приймає масиви даних і розподіляє їх між відповідними сервісами.

4. Підсистема моніторингу опрацьовує отримані метрики та в режимі реального часу оновлює графічні дашборди.

5. Модуль предиктивної аналітики аналізує часові ряди для прогнозування можливої деградації вузлів.

6. Автоматизований DevOps-процес ініціює збірку та формування пакетів нового OTA-оновлення.

7. Спеціалізований OTA-менеджер відповідає за безпечне розповсюдження нових версій firmware безпосередньо до цільових пристроїв.

8. Кінцеві вузли завантажують та інсталюють оновлення, після чого генерують зворотну телеметрію, підтверджуючи свій статус.

Підсумовуючи результати аналізу, було концептуально сформовано комплексну архітектуру DevOps-орієнтованої IoT-системи, що чітко розмежовується на три функціональні рівні: базовий рівень кінцевих пристроїв, проміжний рівень edge-обчислень та вищий рівень хмарних сервісів. У ході дослідження обґрунтовано доцільність використання конкретних апаратних платформ, мережевих шлюзів та програмного забезпечення, які в синергії гарантують надійний моніторинг, глибоку потокову аналітику та безпечне розгортання OTA-оновлень. Розроблена архітектура слугує міцним фундаментом для подальшого проєктування DevOps-конвеєрів та створення математичних моделей оптимізації ПЗ кінцевих пристроїв.

2.2 Специфіка впровадження DevOps-практик у розподілені IoT-системи

У сфері розробки класичних веб-орієнтованих або монолітних серверних програмних продуктів конвеєр безперервної інтеграції та розгортання (CI/CD) традиційно розгортається у висококонтрольованих і передбачуваних умовах — на базі потужних хмарних провайдерів або в межах локальних центрів обробки даних (on-premise). Однак при перенесенні цих практик у середовище Інтернету речей (IoT) виникає цілий спектр безпрецедентних інженерних викликів. Головна складність полягає у тому, що цільовим середовищем для розгортання програмного забезпечення виступає не єдиний потужний сервер, а величезний, географічно розподілений парк фізичних пристроїв. Це обладнання найчастіше характеризується жорстким дефіцитом обчислювальних потужностей та оперативної пам'яті, працює в умовах критично низької пропускної здатності комунікаційних каналів і регулярно стикається з нестабільністю мережевого з'єднання або повною його втратою.

З огляду на ці фактори, імплементація методології DevOps у контексті IoT-інфраструктури має низку яскраво виражених специфічних особливостей:

Глибока гібридність операційного середовища: Архітектура вимагає чіткого розподілу відповідальності, де ресурсоємні процеси (наприклад, глобальна оркестрація оновлень, тренування ML-моделей та комплексна аналітика) виконуються на рівні хмари, тоді як критичні до часу відгуку завдання та первинна фільтрація даних реалізуються безпосередньо на периферії (edge-рівень).

Жорсткі імперативи кібербезпеки: Кожен пакет оновлень повинен проходити обов'язкову процедуру криптографічного підпису за допомогою цифрових сертифікатів. Це єдиний надійний механізм запобігання

несанкціонованим втручанням, ін'єкціям шкідливого коду та компрометації базового мікропрограмного забезпечення (firmware) на кінцевих вузлах.

Обов'язкова підтримка технологій OTA (Over-the-Air): Система повинна гарантувати можливість безпечної віддаленої трансляції, верифікації та автоматичної інсталяції нових версій прошивок "по повітрю", повністю виключаючи необхідність дороговартісного фізичного обслуговування обладнання інженерами на місцях.

Адаптація до апаратних обмежень: Враховуючи лімітовані ресурси мікроконтролерів (як енергетичні, так і пам'яті), виникає критична потреба у мінімізації обсягів переданих даних. Для цього застосовуються технології диференційних пакетів (delta updates), що дозволяють завантажувати лише змінені фрагменти бінарного коду, а не весь образ прошивки.

Наявність механізмів зворотної телеметрії: Одразу після завершення циклу інсталяції оновлення, кінцевий пристрій зобов'язаний згенерувати та відправити на сервер детальний звіт про свій поточний стан і працездатність нових компонентів. Цей зворотний зв'язок є основою для прийняття рішень щодо успішності розгортання або необхідності автоматичного відкоту (rollback) до попередньої стабільної версії.

Для забезпечення ефективного функціонування такої складної екосистеми, загальну архітектуру DevOps-інфраструктури в IoT можна декомпозувати на п’ять взаємопов'язаних функціональних шарів, логіку взаємодії яких проілюстровано на рис. 2.1.

Базовий рівень пристроїв (Device Layer): Формує фізичний фундамент системи. Він об'єднує різноманітні сенсори, первинні перетворювачі, виконавчі механізми та керуючі мікрокомп’ютери (наприклад, архітектури ESP32, Raspberry Pi), що безпосередньо відповідають за збір "сирих" даних із зовнішнього середовища.

Комунікаційний рівень (Communication Layer): Відповідає за надійну та безпечну маршрутизацію інформаційних потоків. Цей рівень спирається на використання оптимізованих мережевих протоколів (MQTT, CoAP, HTTP,

LoRaWAN), які забезпечують стабільний обмін телеметрією між кінцевими вузлами, edge-шлюзами та хмарним ядром.

Рівень управління (Management Layer): Виконує роль адміністративного центру. Його інструментарій дозволяє здійснювати глобальний моніторинг інфраструктури, оркеструвати процеси оновлення firmware та гнучко керувати життєвим циклом великих груп (флотів) пристроїв.

Аналітичний рівень (Analytics Layer): Інтелектуальне ядро екосистеми. Включає в себе масштабовані "озера даних" (Data Lakes), системи потокової обробки та інтегровані модулі машинного навчання, які відповідають за предиктивну аналітику та прогнозування технічного стану обладнання.

Рівень візуалізації та адміністрування (Application Layer): Представляє користувацький інтерфейс системи. Надає операторам та інженерам доступ до інтерактивних панелей керування (дашбордів), інструментів побудови звітів та засобів конфігурації платформи.

Практична реалізація описаної архітектури вимагає застосування спеціалізованого програмного забезпечення. Основні класи інструментів DevOps, адаптованих для потреб IoT, зведено у табл. 2.3.

Таблиця 2.3 — Спеціалізований інструментарій DevOps для екосистем Інтернету речей

Компонент інфраструктури	Типові програмні рішення	Ключове функціональне призначення
CI/CD конвеєри	GitHub Actions, GitLab CI/CD, Jenkins	Повна автоматизація процесів компіляції коду, тестування та підготовки релізів
Технології контейнеризації	Docker, Podman	Надійна ізоляція прикладних мікросервісів та їх прискорене розгортання на edge-вузлах

Системи моніторингу	Prometheus, Grafana, Zabbix	Безперервний збір метрик, логування подій та графічна візуалізація телеметрії
Менеджмент OTA-оновлень	BalenaCloud, Mender.io, AWS IoT Device Management	Оркестрація безпечного та масового віддаленого оновлення мікрокоду пристроїв
Управління безпекою	HashiCorp Vault, інфраструктура OpenSSL	Надійне зберігання криптографічних ключів, токенів доступу та цифровий підпис firmware
Оркестрація контейнерів	Kubernetes, полегшені дистрибутиви K3s/MicroK8s	Динамічне керування життєвим циклом контейнеризованих додатків у розподілених edge-середовищах

Системне впровадження методології DevOps у сферу IoT генерує низку стратегічних переваг. Насамперед це забезпечення абсолютного безперервного обслуговування (High Availability) — завдяки інтелектуальним алгоритмам розгортання, оновлення парку пристроїв відбувається без критичної зупинки бізнес-процесів. Крім того, радикально скорочується показник Time-to-Market (час виведення продукту на ринок) за рахунок оптимізації маршруту коду від середовища розробки до кінцевого IoT-обладнання.

Інтеграція цих практик також виводить на новий рівень автоматичний контроль якості. Впроваджені CI-процеси гарантують ретельне автоматизоване тестування кожного рядка коду ще до моменту його потрапляння в робоче середовище. Паралельно з цим забезпечується максимальна прозорість та керованість інфраструктури: розгалужені системи моніторингу надають інженерам вичерпну інформацію про стан здоров'я будь-якого датчика у режимі реального часу. Ще однією беззаперечною перевагою є безшовна масштабованість. Грамотно побудована DevOps-архітектура здатна

автоматично адаптуватися до експоненційного зростання кількості підключених пристроїв, не вимагаючи лінійного збільшення ресурсів на їх обслуговування.

Підсумовуючи, можна стверджувати, що симбіоз методології DevOps та технологій IoT формує абсолютно нову, революційну парадигму управління розподіленими інженерними системами. Вона не лише гарантує безперервну доставку оновлень, глибокий нагляд та точне прогнозування стану обладнання, але й виступає фундаментальною умовою їхньої безпечної та економічно ефективної експлуатації. Вектор подальшого розвитку та адаптації DevOps-практик до специфіки IoT і надалі буде фокусуватися на посиленні криптографічного захисту, підвищенні рівня автономності edge-шлюзів та вдосконаленні алгоритмів потокової аналітики в реальному часі.

2.3 Способи та програмно-апаратні рішення для збирання, накопичення і опрацювання телеметричної інформації

Стабільна та результативна робота екосистем Інтернету речей критично залежить від фундаментальної архітектури процесів збирання, безпечного накопичення та глибокого опрацювання телеметричної інформації. Саме ці масиви "сирих" даних виступають базовим підґрунтям для здійснення безперервного нагляду за технічним "здоров'ям" обладнання, побудови предиктивних моделей для завчасного виявлення потенційних поломок, а також для імплементації сценаріїв повністю автономного сервісного обслуговування. Якщо розглядати ці процеси крізь призму методології DevOps, то конвеєр роботи з даними має відповідати жорстким критеріям абсолютної безперервності, високої еластичності (масштабованості) та відмовостійкості. Це зумовлено тим, що саме телеметрія живить системи потокової аналітики в режимі реального часу і формує життєво важливий

контур зворотного зв'язку (feedback loop), необхідний для коректного та автономного функціонування пайплайнів CI/CD.

Безпосереднє отримання інформації в інфраструктурі IoT ініціюється на найнижчому, апаратному рівні — безпосередньо на сенсорних вузлах, логічних контролерах та мікрокомп'ютерах. Цей комплексний процес структурно поділяється на кілька послідовних стадій:

Первинне локальне опрацювання (на стороні кінцевого вузла): охоплює фізичне вимірювання цільових характеристик навколишнього середовища чи стану механізмів, програмну фільтрацію апаратних шумів та хибних спрацьовувань, а також математичне усереднення отриманих метрик для мінімізації похибки.

Інкапсуляція та пакетування: етап створення стандартизованих інформаційних повідомлень (payloads). Кожен такий пакет обов'язково маркується точною часовою міткою (timestamp), унікальним криптографічним ідентифікатором конкретного пристрою, масивом виміряних значень та необхідними супровідними метаданими.

Транспортування корисного навантаження: фізична та логічна передача сформованих пакетів до вищих архітектурних рівнів з використанням оптимізованих, легковагових мережевих протоколів (зокрема, MQTT, CoAP, AMQP або через класичні архітектури HTTP REST API).

Концептуальну схему одного з поширених підходів до побудови інфраструктури збору IoT-даних наочно проілюстровано на рис. 2.3. Аналізуючи сучасний технологічний ландшафт, можна констатувати беззаперечне лідерство протоколу MQTT (Message Queuing Telemetry Transport). Його архітектура базується на асинхронному патерні обміну повідомленнями «публікатор-підписник» (publish-subscribe). Головною конкурентною перевагою MQTT є його екстремально низькі накладні витрати на передачу службових заголовків та мінімальні вимоги до ширини мережевого каналу. Ці характеристики роблять його де-факто індустріальним

стандартом для інтеграції пристроїв з жорстко обмеженими обчислювальними та енергетичними ресурсами.

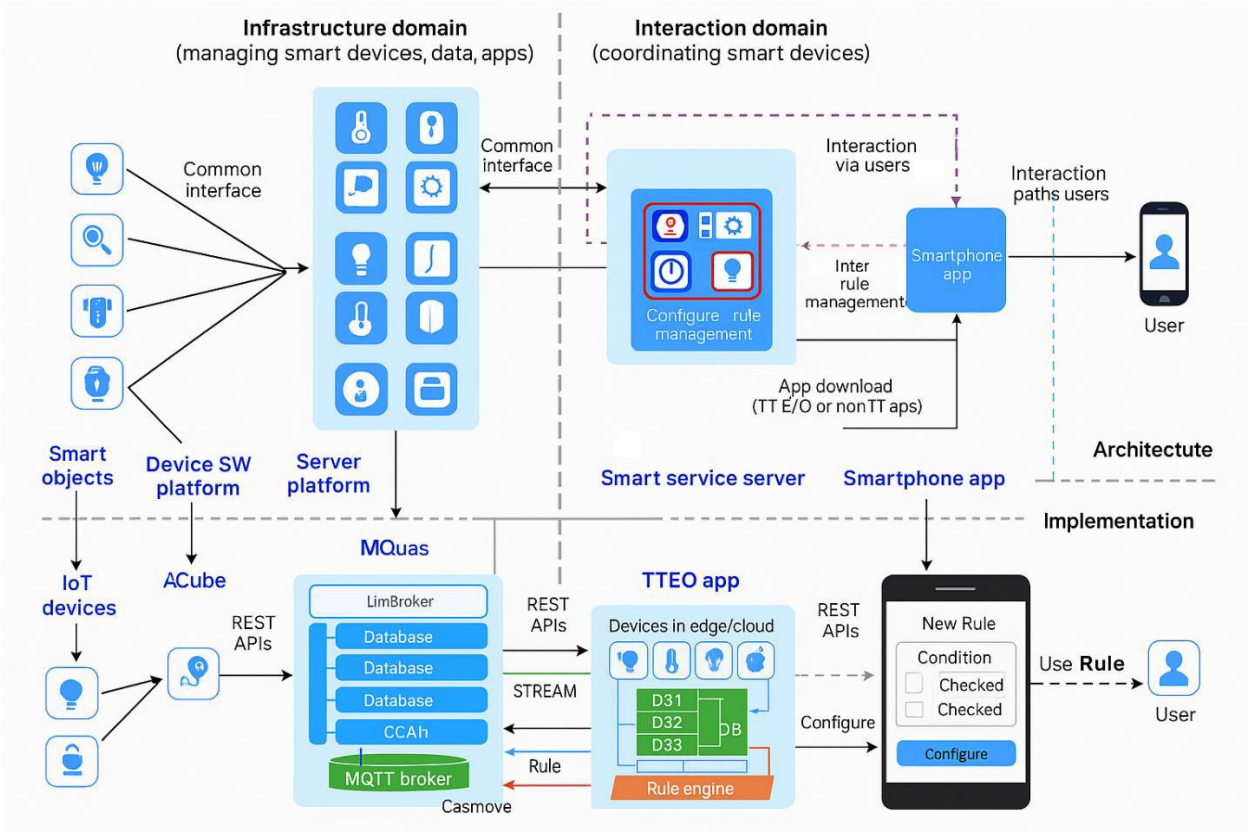


Рисунок 2.3 — Типова архітектурна схема організації інфраструктури для акумулювання телеметрії в мережах IoT

При проєктуванні екосистем, що оперують конфіденційною інформацією або керують об'єктами критичної інфраструктури (де діють посилені стандарти кібербезпеки), базовий протокол модифікується до версії MQTT over TLS. Це розширення гарантує надійне наскрізне шифрування всього мережевого трафіку та строгу взаємну криптографічну аутентифікацію між клієнтськими пристроями і брокером. Детальний порівняльний аналіз ключових комунікаційних протоколів, що домінують у сфері побудови IoT-мереж, систематизовано у табл. 2.4.

Таблиця 2.4 — Порівняльна характеристика базових комунікаційних протоколів в екосистемах Інтернету речей

Назва протоколу	Архітектурний патерн	Ключові технологічні переваги	Головні недоліки та обмеження	Сфери цільового використання
MQTT	Публікація – підписка (Pub/Sub)	Мінімальний трафік, підтримка рівнів QoS, легкість імплементації	Неефективність при трансляції великих бінарних об'єктів	Збір телеметрії, розподілені сенсорні мережі
CoAP	Клієнт – сервер	Мінімалістичність, робота поверх UDP, семантика REST	Вразливості базового рівня безпеки (потребує DTLS)	Системи "розумного дому", малопотужна електроніка
HTTP REST	Клієнт – сервер	Універсальність, безшовна інтеграція з веб-сервісами	Значні накладні витрати на заголовки, ресурсоємність	Взаємодія з хмарними API, адмін-панелі управління
AMQP	Публікація – черга	Висока надійність, сувора гарантія доставки інформації	Значна архітектурна та конфігураційна складність	Фінансовий сектор, складні промислові IoT-рішення

Подолавши етап первинного збору та мережевої трансляції, інформаційні масиви потрапляють до транзитних edge-шлюзів або безпосередньо у хмарні обчислювальні середовища. Саме тут ініціюються процеси глибокої агрегації, тимчасової буферизації та запису в постійну пам'ять для довгострокового зберігання.

У сучасній архітектурі для цих цілей прийнято розмежовувати сховища на дві фундаментальні категорії:

— Бази даних для часових рядів (Time-Series Databases, TSDB): Спеціалізовані програмні рішення (на кшталт InfluxDB, TimescaleDB або Prometheus TSDB), архітектура яких концептуально оптимізована для надшвидкого запису та читання потокових метрик, що жорстко прив'язані до хронологічних міток. Вони є ідеальним інструментом для моніторингу динаміки зміни фізичних параметрів у часі.

— Озера та Сховища даних (Data Lake / Data Warehouse): Масштабовані платформи (наприклад, кластери AWS S3, розподілена файлова система Hadoop HDFS або хмарне аналітичне сховище Snowflake), що слугують централізованим репозиторієм для гігабайтів та петабайтів різнорідної інформації. Їхня головна роль полягає у забезпеченні базису для тренування складних моделей машинного навчання (ML), проведення ретроспективного аналізу та надійного тривалого архівування історичних даних.

Крім того, для запобігання втраті телеметрії в моменти пікових навантажень або під час тимчасових перебоїв мережевого зв'язку, між кінцевими пристроями та базами даних обов'язково розгортається шар проміжного кешування. Цю критично важливу функцію виконують потужні брокери повідомлень та системи управління чергами (зокрема, Apache Kafka, RabbitMQ або Eclipse Mosquitto). Вони акумулюють вхідний трафік у відмовостійких чергах, гарантуючи асинхронну та безперебійну передачу даних до аналітичних сервісів. Комплексна маршрутизація цих інформаційних потоків наочно відображена на рис. 2.4.

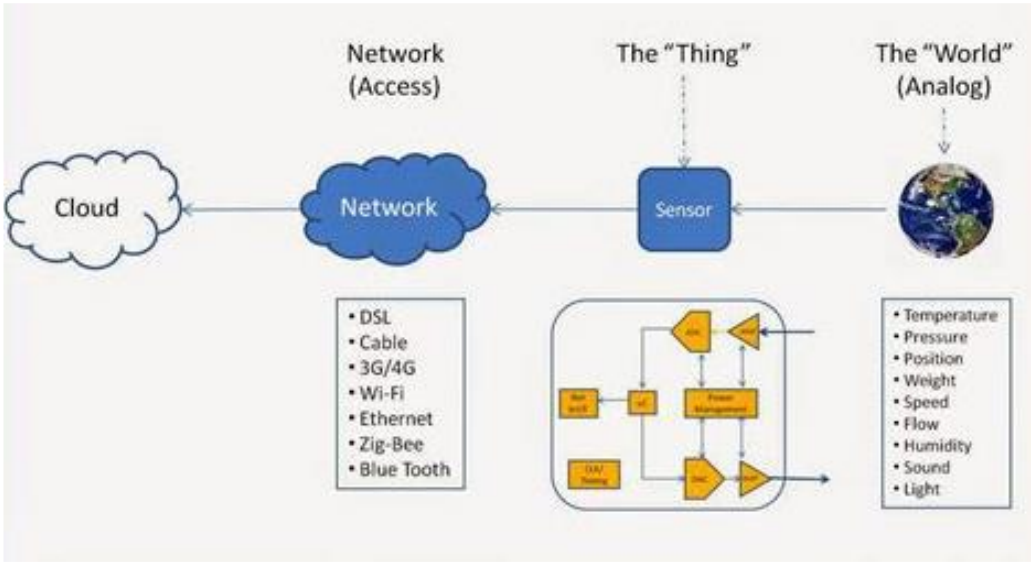


Рисунок 2.4 — Концептуальна модель процесу реєстрації, маршрутизації та транспортування телеметричних метрик від кінцевого обладнання

Процедура безпосереднього опрацювання "сирої" телеметрії стартує з етапів жорсткої програмної фільтрації та математичної нормалізації. Ця стадія передбачає виявлення та відсікання апаратних аномалій (викидів) і уніфікацію гетерогенних даних — приведення різноманітних одиниць вимірювання від датчиків різних виробників до єдиної стандартизованої метричної шкали. Наступним логічним кроком є агрегація, яка зводиться до групування та консолідації атомарних вимірювань у межах заданих часових вікон (наприклад, обчислення медіанного значення температури за хвилину замість зберігання тисяч мілісекундних записів).

Для реалізації складної інтелектуальної аналітики в режимі реального часу інженери залучають високопродуктивні фреймворки потокових обчислень, такі як Apache Spark Streaming, Apache Flink або модулі Kafka Streams. Водночас, для предиктивного прогнозування технічного стану апаратного забезпечення на основі накопичених часових рядів розгортаються алгоритми машинного навчання різного ступеня складності: від класичних статистичних моделей ARIMA до рекурентних нейронних мереж довгої короткострокової пам'яті (LSTM) та потужних ансамблевих методів (Random Forest, Gradient Boosting).

Кінцевим етапом обробки є наочна візуалізація операційних показників, що реалізується за допомогою передових інструментів Business Intelligence та моніторингу (Grafana, Kibana, Microsoft Power BI). Ці інтерактивні платформи безпосередньо інтегруються у загальний DevOps-пайплайн проєкту. У парадигмі DevOps весь цей багаторівневий механізм опрацювання даних розглядається як невід'ємна складова процесу безперервного моніторингу (Continuous Monitoring, CM). Саме він формує об'єктивний зворотний зв’язок для CI/CD-конвеєра, дозволяючи системам оркестрації автоматично ініціювати коригування конфігураційних файлів пристроїв без втручання людини.

З метою подальшої максимізації продуктивності та оптимізації пропускної здатності систем обробки метрик, сучасні інженерні практики пропонують імплементацію низки передових підходів:

— Розгортання Edge-аналітики: перенесення частини обчислювального навантаження (первинної агрегації та фільтрації) безпосередньо на граничні мережеві шлюзи або мікроконтролери самих сенсорів, що радикально розвантажує магістральні канали зв'язку.

— Стиснення та інтелектуальна вибірка (Data compression & sampling): застосування математичних алгоритмів компресії та методів статистичної вибірки, що дозволяє передавати репрезентативні зрізи даних замість повного, надлишкового потоку сирої інформації.

— Гібридна обробка (Batch and stream processing): синергетичне поєднання технологій пакетної (асинхронної обробки гігантських масивів за розкладом) та потокової (в реальному часі) аналітики, що застосовуються диференційовано залежно від специфіки конкретного бізнес-завдання.

— Впровадження конвеєрів виявлення аномалій (Anomaly detection pipelines): створення автоматизованих пайплайнів, що перманентно сканують телеметрію на предмет девіацій, дозволяючи проактивно реагувати на загрози ще до настання фактичної апаратної чи програмної відмови.

Попри значний технологічний стрибок, проєктування таких комплексних систем досі стикається з низкою серйозних перешкод. До

ключових проблем відносяться: безпрецедентні обсяги генерації "сирих" даних, що вимагає імплементації вкрай агресивних механізмів фільтрації; гарантування абсолютної цілісності пакетів під час їхнього транспортування нестабільними бездротовими мережами; складність семантичного узгодження протоколів від гетерогенного обладнання різних вендорів. Окремим масштабним викликом залишається проєктування збалансованої архітектури баз даних та жорстка імплементація багаторівневих політик контролю доступу (IAM) для надійного захисту інформаційних потоків.

Окреслюючи перспективи подальшого розвитку цієї галузі, експерти прогнозують стрімку еволюцію традиційних брокерів повідомлень у напрямку створення "інтелектуальних" мережевих вузлів. Такі смарт-брокери будуть наділені здатністю самостійно виконувати базові скрипти предиктивної аналітики та автоматично блокувати аномальні потоки даних безпосередньо на рівні первинної маршрутизації. Не менш значущим стратегічним вектором є глибинна конвергенція практик DevOps із суміжною методологією DataOps. Таке комплексне об'єднання дозволить сформувати повністю прозорий, автоматизований цикл управління даними: від моменту їхньої фізичної генерації сенсором до фінальної побудови бізнес-аналітики та ініціації оновлень цільової системи.

3 СТВОРЕННЯ ТА НАЛАГОДЖЕННЯ ПРОГРАМНИХ КОМПОНЕНТІВ IOT-МЕРЕЖІ Й АВТОМАТИЗОВАНИХ DEVOPS-ПЛАТФОРМ

3.1 Практичне створення апаратного забезпечення вузла Інтернету речей

Апаратна інфраструктура кінцевого IoT-вузла виступає фундаментальним базисом для всієї багаторівневої системи, орієнтованої на методологію DevOps, яка забезпечує безперервний моніторинг, предиктивну аналітику та віддалену модернізацію пристроїв. Саме від того, наскільки грамотно здійснено підбір фізичної платформи, сенсорної бази та алгоритмів збору телеметричної інформації, безпосередньо залежить валідність отриманих даних, загальна експлуатаційна стабільність обладнання, а також сама технічна можливість безпечного розгортання оновлень мікрокоду (firmware). У межах даного дослідження фізичну архітектуру розумного вузла було спроєктовано з глибоким урахуванням сучасних інженерних стандартів щодо еластичного масштабування мережі, жорсткої енергоефективності, криптографічного захисту комунікацій та повної сумісності з розгорнутою DevOps-інфраструктурою.

В якості центрального обчислювального ядра для побудови цільового IoT-вузла було обрано мікроконтролерну систему на кристалі (SoC) сімейства ESP32. Ця платформа здобула широке визнання та масово впроваджується як у комерційних індустріальних проєктах, так і в передових академічних розробках. Таке архітектурне рішення аргументується оптимальним балансом між високою продуктивністю обчислень, наявністю інтегрованих на апаратному рівні радіомодулів та нативною підтримкою сучасних криптографічних протоколів, рис. 3.1.

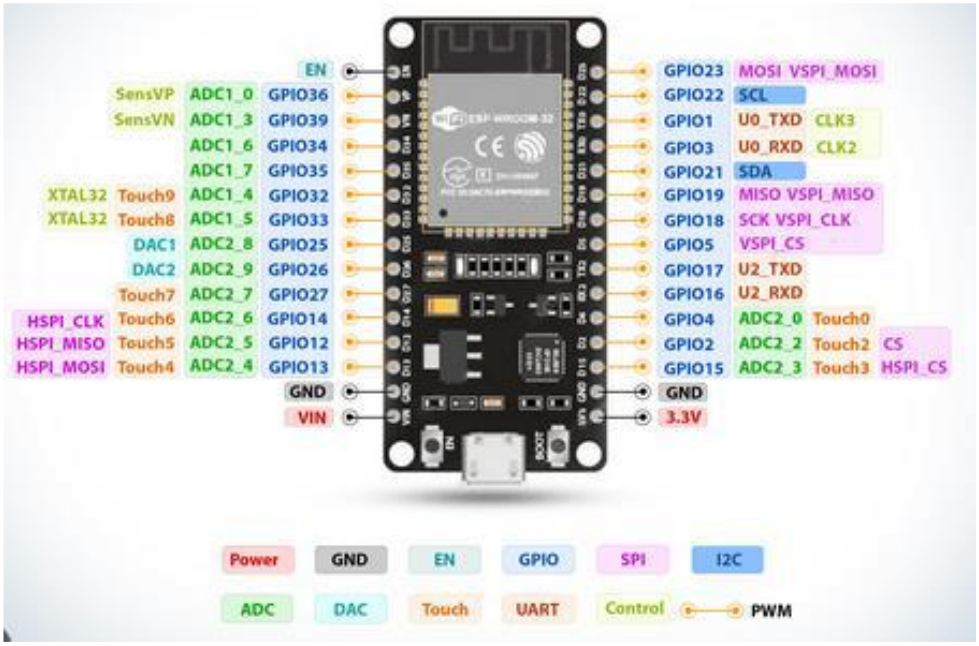


Рисунок 3.1 — Зовнішній вигляд та архітектурні особливості мікроконтролера платформи ESP32

З технічної точки зору, базовий модуль ESP32 являє собою потужний двоядерний мікропроцесор, побудований на архітектурі Xtensa (або RISC-V у новітніх ревізіях). Його беззаперечною перевагою є вбудована підтримка бездротових стандартів Wi-Fi та Bluetooth (зокрема BLE), а також наявність апаратних прискорювачів для алгоритмів шифрування (AES, RSA, SHA). Платформа оснащена достатніми обсягами оперативної (SRAM) та енергонезалежної флеш-пам'яті, що є критично важливим для збереження як основної прошивки, так і завантажених бінарних образів під час процесу оновлення. Особливої уваги заслуговує підтримуваний контролером механізм апаратного сегментування флеш-пам'яті на логічні розділи (зокрема, розділи OTA0 та OTA1). Ця топологія дозволяє реалізувати абсолютно безпечний конвеєр оновлення програмного забезпечення "по повітрю" з гарантованою можливістю автоматичного повернення (відкату або rollback) до попередньої стабільної версії у разі виникнення критичних збоїв під час ініціалізації нового коду.

Для об'єктивного порівняння варто розглянути суміжні архітектурні рішення. Наприклад, популярні мікроконтролери сімейства STM32

здебільшого орієнтовані на створення класичних вбудованих систем (embedded systems) жорсткого реального часу. Для повноцінної роботи в мережі вони найчастіше потребують підключення зовнішніх мережевих трансиверів, що суттєво ускладнює апаратну розробку та організацію надійних OTA-оновлень. З іншого боку, одноплатні мікрокомп'ютери класу Raspberry Pi володіють набагато вищим обчислювальним потенціалом (завдяки повноцінній ОС Linux), проте їхнє високе енергоспоживання робить їх використання недоцільним на рівні автономних кінцевих датчиків; їхня ніша — це потужні edge-шлюзи. Враховуючи ці фактори, вибір ESP32 є найбільш збалансованим і раціональним компромісом між багатим функціоналом, вимогами до автономності та кінцевою вартістю вузла.

Для комплексного моніторингу технічного "здоров'я" IoT-вузла застосовується синергія апаратних та програмних метрик телеметрії. До критичного пулу параметрів, що безперервно реєструються та транслюються до центральної аналітичної системи, входять: тепловий стан кристала мікроконтролера, вольт-амперні характеристики підсистеми живлення, ступінь утилізації обчислювальних потужностей та якісні показники бездротового з'єднання.

Температурний режим мікропроцесора виступає одним із найважливіших індикаторів його стабільності. Хронічний перегрів (thermal throttling) не лише різко знижує продуктивність, але й провокує прискорену фізичну деградацію напівпровідникових структур. В архітектурі ESP32 моніторинг температури реалізується шляхом опитування вбудованих термодатчиків або через непрямий аналіз системних регістрів, що дає змогу операторам відстежувати теплову динаміку в режимі реального часу.

Метрики підсистеми живлення охоплюють поточну напругу на шині, відсоток залишкової ємності акумуляторної батареї (у сценаріях автономного розгортання) та загальну динаміку енергоспоживання. Аналіз цих часових рядів дозволяє системі проактивно ідентифікувати апаратні несправності в

колах живлення, фіксувати хімічну деградацію елементів живлення та запобігати критичним перевантаженням енергетичного тракту.

Статус використання системних ресурсів оцінюється через поточне завантаження обчислювальних ядер, обсяг вільної оперативної пам'яті (Free Heap) та статистику системних винятків (помилок виконання). Для мікроконтролерів ESP32 ці дані легко отримуються через стандартні виклики системного API. Вони є незамінним інструментом для DevOps-інженерів під час профілювання ефективності нового мікрокоду, а також для раннього виявлення витоків пам'яті чи зациклень у нестабільних ділянках алгоритмів.

Якість радіоканалу та мережевого підключення вимірюється через рівень потужності прийнятого сигналу (RSSI), мережевий пінг, затримки (latency) при обміні MQTT-повідомленнями, а також частоту розривів сесій і відсоток втрачених TCP-пакетів. Ці метрики мають вирішальний вплив на успішність доставки OTA-пакетів та гарантію безперебійності телеметричного потоку.

Сам процес збору метрик на кінцевому вузлі архітектурно організований як циклічне полінгове опитування масиву апаратних і програмних сенсорів. Отримані "сирі" дані не відправляються відразу, а проходять етап локальної агрегації, формуючись у структуровані JSON або бінарні повідомлення. Така первинна обробка на рівні самого вузла (мікро-edge обчислення) критично необхідна для оптимізації обсягів вихідного трафіку та розвантаження магістральної мережі.

До процедур попередньої обробки належать: математична фільтрація аномальних спайків (викидів), обчислення середнього ковзного у межах заданих часових вікон та компактизація даних. Для безпосереднього транспортування пакетів застосовується індустріальний IoT-протокол MQTT, який ідеально підходить для таких завдань завдяки своїй асинхронній природі, малим розмірам службових заголовків та підтримці рівнів якості обслуговування (QoS).

Надзвичайно важливим архітектурним нюансом є чітка синхронізація підсистеми телеметрії з процесами OTA-оновлень. На етапі завантаження нового образу прошивки частота генерації та відправки телеметрії свідомо знижується для звільнення пропускної здатності каналу та процесорного часу. Однак, одразу після перезавантаження з новим кодом, частота опитування сенсорів тимчасово переводиться в інтенсивний режим для прискіпливого контролю стабільності. Такий поведінковий патерн дозволяє системі миттєво ідентифікувати фатальні помилки оновлення і автоматично активувати механізм апаратного відкату на попередній розділ пам'яті.

Варто пам'ятати, що екосистема Інтернету речей завжди функціонує в умовах жорсткого дефіциту ресурсів: тактова частота процесора, доступний простір SRAM та ємність акумулятора є суворо лімітованими. Хоча ESP32 є потужною мікроконтролерною базою, вона підпорядковується цим законам, що висуває суворі вимоги до якості написання програмного коду та управління енергозбереженням.

Енергетичний бюджет пристрою прямо пропорційний часу активності радіомодуля, частоті процесора та інтенсивності передачі даних. Для радикального зниження енергоспоживання в архітектуру прошивки імплементуються механізми глибокого (Deep Sleep) та легкого (Light Sleep) сну, алгоритми динамічного масштабування тактової частоти ядра та оптимізація періодів пробудження для виходу в ефір. Ці заходи дозволяють збільшити життєвий цикл автономних датчиків від кількох днів до багатьох місяців або навіть років. Паралельно з цим, обмеження оперативної пам'яті диктують необхідність жорсткого контролю над буферами. Інженери повинні уникати фрагментації пам'яті (шляхом відмови від динамічного виділення пам'яті в циклі) та гарантувати коректну роботу зказників. Це набуває критичної ваги саме під час OTA-сесій, коли системі потрібно виділити значні безперервні блоки пам'яті для розпакування та верифікації криптографічних підписів нового бінарного файлу.

Резюмуючи, можна стверджувати, що апаратна архітектура досліджуваного IoT-вузла спроєктована з глибоким розумінням специфіки сучасних розподілених мереж. Обрана компонентна база, інтегровані сенсорні модулі та оптимізовані алгоритми обробки телеметрії формують надійний, відмовостійкий базис для успішної реалізації високорівневих DevOps-практик: безперервного моніторингу, предиктивного прогнозування життєвого циклу та повністю автоматизованого розгортання оновлень мікрокоду. Практичні аспекти цих процесів будуть детально розкриті у наступних підрозділах.

3.2 Програмна конфігурація пристроїв Інтернету речей та граничних обчислювальних систем

Програмне забезпечення кінцевого IoT-вузла, побудованого на мікроконтролерній базі ESP32, спроєктовано у вигляді складної багатомодульної прошивки. Її архітектура базується на принципах слабкої зв'язності: вона складається з повністю незалежних функціональних компонентів, взаємодія між якими реалізується виключно через безпечні внутрішні черги повідомлень (message queues) операційної системи реального часу та захищені від одночасного доступу спільні структури даних. Застосування такого патерну проєктування дозволяє жорстко ізолювати бізнес-логіку збору метрик від підсистем мережевої комунікації та процесів бездротового оновлення (OTA), що кардинально підвищує загальну експлуатаційну надійність пристрою та унеможливлює каскадні збої.

Структурна декомпозиція прошивки передбачає наявність таких базових модулів:

— підсистема ініціалізації та конфігурування апаратних ресурсів;

— ядро агрегації телеметричної інформації;

— комунікаційний стек для мережевої взаємодії (що підтримує протоколи MQTT та HTTP);

— оркестратор процесу OTA-оновлення мікрокоду;

— підсистема проактивної самодіагностики та обробки виняткових ситуацій.

Візуальне відображення логіки взаємодії цих компонентів наведено на рис. 3.2. Як видно зі схеми, прошивка функціонує як скоординований ансамбль модулів, де кожен елемент несе відповідальність за чітко визначений етап життєвого циклу пристрою.



Рисунок 3.2 — Високорівнева структурна діаграма програмної прошивки на базі мікроконтролера ESP32

Модуль ініціалізації апаратних ресурсів відіграє роль завантажувача (bootstrapper). Він активується першим при подачі живлення та виконує низькорівневе налаштування ядра мікроконтролера, конфігурацію портів вводу/виводу, ініціалізацію підключених зовнішніх сенсорів по шинах (I2C, SPI) та підняття мережевих інтерфейсів. Основна мета цього компонента — гарантувати, що апаратна платформа повністю готова до штатного

функціонування, та завантажити базові конфігураційні параметри з енергонезалежної пам'яті.

Наступним критично важливим компонентом є ядро агрегації телеметрії. Його завдання полягає у систематичному, циклічному опитуванні як фізичних сенсорів, так і внутрішніх програмних лічильників IoT-вузла. Об'єктами моніторингу виступають: тепловий стан кристала процесора, рівень вільної оперативної пам'яті (Heap), ступінь завантаженості обчислювальних ядер та метрики якості бездротового з'єднання. Зібраний масив "сирих" даних проходить первинну обробку та передається до комунікаційного стека для відправки на сервер.

Процес збору телеметрії архітектурно реалізований як відокремлене фонове завдання (Task) під управлінням операційної системи FreeRTOS. Це завдання прокидається із жорстко заданим часовим інтервалом і виконує чітку послідовність дій:

— проводить опитування всіх зареєстрованих датчиків та читання системних регістрів для отримання поточних технічних параметрів вузла;

— здійснює пакування розрізнених змінних у єдину, уніфіковану структуру телеметричного пакета (payload);

— безпечно передає сформований пакет до черги модуля мережевої взаємодії для публікації через брокер MQTT.

Схематичну структуру формату телеметричних даних, що генеруються вузлом, проілюстровано на рис. 3.3.



Рисунок 3.3 — Логічна структура та формат пакета телеметричних даних

Для моніторингу теплового режиму пристрою програмно реалізовано алгоритм звернення до вбудованого в кристал ESP32 температурного сенсора. Логіка роботи цієї функції полягає в ініціалізації аналого-цифрового перетворювача (АЦП), прив'язаного до внутрішнього термодіода, знятті кількох послідовних вимірювань, їхньому усередненні для уникнення шумів та конвертації сирого значення (raw value) у градуси Цельсія за відкаліброваною формулою.

Контроль за станом оперативної пам'яті здійснюється через виклик спеціалізованих системних функцій FreeRTOS. Алгоритм визначає загальний обсяг вільної пам'яті (Free Heap Space) та фіксує мінімальний історичний обсяг вільної пам'яті з моменту останнього перезавантаження (Minimum Ever Free Heap). Ця інформація є критично важливою для виявлення потенційних витоків пам'яті (memory leaks) під час тривалої безперервної роботи пристрою.

Оцінка стабільності радіоканалу реалізується програмним модулем, який періодично звертається до драйвера Wi-Fi інтерфейсу мікроконтролера. Алгоритм зчитує показник Received Signal Strength Indicator (RSSI), який

відображає потужність прийнятого сигналу від точки доступу у децибел-міліватах (дБм). Ці дані дозволяють аналітичній системі будувати карту покриття та прогнозувати можливі розриви зв'язку.

Фонове завдання FreeRTOS, що керує збором метрик, налаштовується з урахуванням пріоритетів планувальника ОС. Йому виділяється власний стек пам'яті, і воно працює у нескінченному циклі while(1). Всередині циклу завдання засинає на визначений час (наприклад, використовуючи функцію vTaskDelay), звільняючи процесорний час для інших процесів, а після пробудження виконує збір метрик і знову переходить у стан очікування. Запуск цього процесу відбувається на етапі загальної ініціалізації прошивки, відразу після успішного підняття мережевих інтерфейсів.

Комунікаційний стек (модуль мережевої взаємодії) виступає сполучною ланкою між фізичним пристроєм, транзитними edge-шлюзами та глобальною хмарною інфраструктурою. Його фундаментальна місія полягає у гарантованій, захищеній та максимально енергоефективній доставці сформованих телеметричних пакетів до центрів обробки. Крім того, модуль забезпечує двосторонній зв'язок: він приймає керуючі команди з хмари, завантажує оновлені конфігураційні файли та керує процесом скачування нових бінарних образів прошивки.

Для досягнення максимальної гнучкості та надійності у прошивці реалізовано гібридну мережеву модель, яка паралельно використовує можливості протоколів MQTT та HTTP. Такий симбіоз дозволяє диверсифікувати мережеве навантаження і підтримувати працездатність навіть у разі деградації каналу зв'язку.

Протокол MQTT визначено як базовий транспорт для безперервного потоку телеметрії. Його вибір зумовлений екстремально низьким розміром службових заголовків (overhead), нативною підтримкою асинхронного патерну "публікація-підписка" та чудовою стійкістю до роботи в мережах з високими затримками (high latency) та втратами пакетів. Для стандартизації маршрутизації повідомлень на брокері впроваджено сувору ієрархічну

структуру топіків (topic tree), яка дозволяє легко ідентифікувати пристрій, його групу та тип переданих даних. Графічне представлення цієї структури топіків наведено на рис. 3.4.



Рисунок 3.4 — Ієрархічна модель адресації та маршрутизації топіків у протоколі MQTT

Процес ініціалізації клієнта MQTT включає налаштування параметрів з'єднання (IP-адреса брокера, порт), конфігурацію сертифікатів безпеки (якщо використовується TLS), визначення унікального Client ID, налаштування облікових даних (логін/пароль) та реєстрацію callback-функцій, які будуть викликатися асинхронно при успішному з'єднанні, розриві зв'язку або надходженні вхідних повідомлень з підписаних топіків.

Відправка зібраної телеметрії реалізується через механізм публікації (publish) MQTT. Алгоритм бере підготовлену структуру даних, серіалізує її у текстовий формат JSON для забезпечення кросплатформної сумісності та відправляє на брокер. При цьому критично важливо, що публікація здійснюється з рівнем якості обслуговування QoS = 1 (At least once). Це означає, що клієнт очікуватиме підтвердження доставки від брокера

(PUBACK), і в разі його відсутності — автоматично повторить відправку, що унеможливлює непомітну втрату даних.

Водночас протокол HTTP (а точніше його захищена версія HTTPS) резервується для виконання "важких" операцій, що вимагають передачі значних обсягів інформації або жорсткої синхронної взаємодії за моделлю "запит-відповідь". Завдяки семантиці REST, він ідеально інтегрується з існуючими хмарними веб-сервісами.

Головні Use Cases (сценарії використання) для протоколу HTTP включають:

— періодичні GET-запити до серверів оновлень для перевірки наявності нових релізів мікрокоду;

— завантаження складних конфігураційних файлів у форматі JSON;

— POST-запити для відправки об'ємних дампів системних логів (crash logs) у разі фатальних збоїв для подальшого аналізу (post-mortem debugging).

З метою забезпечення відмовостійкості (resilience), мережевий стек містить складну логіку обробки помилок. Вона включає: алгоритми експоненційної затримки (exponential backoff) при спробах перепідключення до Wi-Fi чи брокера, тимчасову буферизацію невідправлених повідомлень у кільцевому буфері оперативної пам'яті під час відсутності зв'язку, а також суворий контроль системних тайм-аутів (watchdogs) на мережевих сокетах. Безпека транспортного рівня беззаперечно гарантується використанням сучасних криптографічних стандартів: TLS версії 1.2 або 1.3 для всіх HTTPS-сесій, обов'язковою авторизацією клієнтів на MQTT-брокері за токенами та суворою верифікацією ланцюжка сертифікатів сервера для захисту від атак типу Man-in-the-Middle (MitM).

Архітектурно виокремлений оркестратор OTA-оновлень бере на себе повний цикл відповідальності за безпечну доставку, криптографічну верифікацію та фінальну інсталяцію нових версій firmware. Його ключова особливість полягає у здатності не лише оновлювати систему, але й

гарантувати автоматичний відкат (rollback) до попередньої стабільної версії у випадку, якщо новий код виявиться неробочим, запобігаючи перетворенню пристрою на неробочий шматок пластику ("цеглу").

Цей механізм технічно спирається на можливості платформи ESP32 щодо роботи з розділами пам'яті (OTA partitioning). Енергонезалежна флеш-пам'ять пристрою жорстко сегментується на кілька логічних дисків. Крім завантажувача (bootloader) та розділу даних (NVS), виділяються дві симетричні області для зберігання прошивок: OTA_0 та OTA_1. У будь-який момент часу мікроконтролер виконує код лише з одного, "активного" розділу. Завантаження нового образу завжди відбувається у "пасивний" (сплячий) розділ, що дозволяє пристрою продовжувати виконувати свої основні функції паралельно з процесом скачування оновлення у фоновому режимі.

Життєвий цикл процесу OTA-оновлення регламентується чітким алгоритмом:

Ініціація процесу: вузол перевіряє доступність нової прошивки на сервері (за розкладом або за командою з хмари).

Потокове завантаження: бінарний образ (binary image) частинами скачується через захищене HTTPS-з'єднання.

Багаторівнева верифікація: система перевіряє цілісність скачаного файлу (шляхом порівняння контрольних сум MD5 або SHA256) та валідує цифровий підпис (наприклад, ECDSA), щоб переконатися, що прошивка створена авторизованим розробником і не була модифікована.

Прошивка: бінарні дані записуються у неактивний OTA-розділ флеш-пам'яті.

Зміна конфігурації завантажувача: система маркує оновлений розділ як "готовий для завантаження" при наступному рестарті.

Програмне перезавантаження пристрою.

Фаза тестування: завантажувач запускає нову версію прошивки в тестовому режимі.

Фінальне підтвердження або Rollback: прошивка повинна самостійно підтвердити свою працездатність (наприклад, успішно підключитися до Wi-Fi та відправити пінг на сервер) протягом тайм-ауту. Якщо підтвердження не відбувається (або трапляється критична помилка Kernel Panic), апаратний сторожовий таймер перезавантажить пристрій, а завантажувач автоматично поверне активність попередньому, гарантовано робочому розділу.

Логіка завантаження OTA-образу через протокол HTTPS реалізована за допомогою стандартних бібліотек ESP-IDF. Вона передбачає ініціалізацію HTTP-клієнта, налаштування необхідних заголовків авторизації, встановлення TLS-з'єднання з сервером оновлень та відкриття потоку для читання бінарних даних. Далі алгоритм у циклі вичитує дані блоками фіксованого розміру (наприклад, по 1024 байти) і послідовно записує їх у підготовлений флеш-розділ. Цей процес супроводжується постійним обчисленням поточного хешу для фінальної перевірки.

Критичним етапом безпеки є перевірка цифрового підпису перед тим, як дозволити запис образу в пам'ять. Програмний модуль зчитує відкритий ключ (public key), який жорстко зашитий (hardcoded) у поточній прошивці або безпечно зберігається у захищеному сховищі ключів. Потім він порівнює криптографічний підпис, прикріплений до завантаженого файлу, з розрахованим хешем самого образу. Будь-яка розбіжність призводить до негайного скасування операції оновлення та генерації тривожного повідомлення (Security Alert).

Механізм умовного підтвердження нової прошивки (app_confirm) реалізується через систему таймерів. Відразу після старту нового коду запускається таймер зворотного відліку. Прошивка має виконати низку перевірок життєздатності (sanity checks): успішно ініціалізувати всі сенсори, отримати IP-адресу, встановити сесію з MQTT-брокером та опублікувати перше повідомлення "System Started". Лише після виконання всіх цих кроків програма викликає функцію маркування поточного розділу як "стабільного"

(valid). Якщо ж під час цих спроб мікроконтролер "зависає", апаратний Watchdog Timer (WDT) перезавантажить його, і система відкотиться назад.

У контексті глобальної інфраструктури, цей OTA-модуль є кінцевою точкою розгортання в DevOps-конвеєрі. Процес виглядає так: після коміту в репозиторій, CI-сервер (наприклад, GitHub Actions) автоматично компілює вихідний код, генерує бінарний файл .bin, накладає на нього цифровий підпис за допомогою секретного ключа і пушить цей артефакт у репозиторій OTA-менеджера. Після цього розумні пристрої самостійно, без втручання оператора, виявляють оновлення та проводять його інсталяцію, забезпечуючи концепцію Continuous Deployment (Безперервне розгортання) у світі апаратного забезпечення.

Крім того, результати роботи модуля OTA-оновлення мають прямий вплив на аналітичну систему оцінки стану пристрою. У перші години після переходу на нову версію прошивки система підвищує пріоритет моніторингу метрик: ретельно аналізуються зміни температурного фону (можливо, новий алгоритм сильніше навантажує процесор), стабільність енергоспоживання (чи не з'явилося аномальне використання радіомодуля), а також ведеться суворий облік будь-яких неочікуваних перезавантажень (reboots) та мікрозатримок під час передачі телеметрії. Уся ця інформація формує датасет для моделей машинного навчання, які прогнозують швидкість деградації обладнання.

Фінальним елементом програмної архітектури є підсистема проактивної самодіагностики та обробки винятків (Error Handling). Її завдання — безперервний фоновий моніторинг поведінки як програмних потоків, так і апаратної периферії, оперативна реєстрація інцидентів та формування структурованих діагностичних звітів (tracebacks). Наявність цього модуля є критичною для запобігання фатальним збоям і можливості віддаленої діагностики причин відмови без фізичного доступу до серійного порту пристрою (debug port).

Функціонал модуля самодіагностики сфокусований на ідентифікації ознак ресурсної деградації та точній фіксації тригерів, що спричинили збої.

Він перехоплює події апаратних та програмних перезавантажень (відстежуючи причини через регістри RESET_REASON), реагує на спрацювання Watchdog-таймерів, логує помилки виділення пам'яті (malloc failed) та фіксує тайм-аути при роботі з I2C/SPI шинами. Модуль не лише реєструє ці події в енергонезалежному лозі (у файловій системі SPIFFS або LittleFS), але й самостійно ініціює превентивні дії: може тимчасово знизити частоту відправки телеметрії (throttling) для економії енергії, примусово перезапустити завислий мережевий стек або навіть запросити автоматичний відкат прошивки (OTA rollback), якщо кількість критичних помилок перевищує допустимий поріг.

Внутрішня архітектура цього модуля базується на системі правил (Rule Engine), яка містить жорстко задані порогові значення (thresholds) та логічні умови для різних параметрів. Крім того, до його складу входить реєстратор подій (Event Logger) та алгоритм оцінки загального індексу здоров'я (Health Score) пристрою. Для систематизації логів у коді визначено спеціальну структуру даних "Діагностична Подія" (Diagnostic Event). Ця структура інкапсулює в собі такі поля: мітку часу (Unix timestamp) виникнення помилки, код серйозності (Severity level: Info, Warning, Error, Critical), унікальний ідентифікатор типу помилки (Error Code) та необов'язкове текстове поле з контекстом (наприклад, "MQTT connection timeout").

Програмний механізм виявлення причин останнього перезавантаження пристрою працює на етапі ініціалізації системи. Він зчитує значення спеціального системного регістра (наприклад, rtc_get_reset_reason в ESP-IDF), розшифровує його (визначаючи, чи це було нормальне перезавантаження по живленню (Power-on Reset), спрацювання сторожового таймера (WDT Reset), виняток процесора (Exception Panic) чи вихід з режиму глибокого сну (Deep Sleep Wakeup)), і записує цю інформацію у стартовий лог, який згодом відправляється на сервер.

Логіка визначення правил самодіагностики реалізується через систему конфігураційних параметрів, які можуть бути оновлені "по повітрю". Наприклад, правило може бути сформульоване так: IF (FreeHeap < 10KB) AND

(Duration > 5 min) THEN TRIGGER Event(MEMORY_LEAK_WARNING) AND EXECUTE Action(RestartNetworkStack). Такий підхід дозволяє динамічно адаптувати чутливість пристрою до помилок залежно від умов експлуатації.

Підсумовуючи, запропонована структурна декомпозиція та описані алгоритми програмної реалізації формують гнучку, високо надійну та модульну архітектуру прошивки мікроконтролера. Такий дизайн не лише забезпечує стабільне виконання базових функцій IoT-вузла, але й створює міцний фундамент для безшовної інтеграції з комплексними DevOps-інфраструктурами, гарантуючи ефективний моніторинг, глибоку аналітику та безпечне управління життєвим циклом програмного забезпечення пристроїв.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було досліджено проблематику контролю та технічної підтримки систем Інтернету речей. На основі аналізу встановлено, що однією з найбільш вагомих та перспективних тенденцій розвитку галузі є парадигмальний зсув від моделі реактивного усунення несправностей на користь проактивного, інтелектуального прогнозувального обслуговування.

За результатами дослідження можна зробити наступні висновки:

1. Сформовано комплексну архітектуру DevOps-орієнтованої IoT-системи, що чітко розмежовується на три функціональні ієрархічні рівні: базовий рівень кінцевих пристроїв, проміжний рівень edge-обчислень та вищий рівень хмарних сервісів.

2. Обґрунтовано доцільність використання мікроконтролерної системи ESP32 як центрального обчислювального ядра для кінцевого IoT-вузла. Її застосування дозволяє реалізувати абсолютно безпечний конвеєр оновлення програмного забезпечення "по повітрю" (OTA) з гарантованою можливостю автоматичного відкату до попередньої версії у разі збоїв.

3. Розроблено архітектуру програмної прошивки IoT-пристрою, яка базується на принципах слабкої зв'язності та складається з ізольованих модулів для агрегації телеметрії, мережевої взаємодії та самодіагностики.

4. Визначено оптимальну мережеву модель комунікації, яка паралельно використовує протокол MQTT як базовий транспорт для безперервного потоку телеметрії завдяки його низьким накладним витратам, та протокол HTTP(S) для завантаження масивних бінарних образів нових прошивок.

5. Доведено, що комплексне поєднання потужностей хмарних сервісів із передовими підходами DevOps дозволяє максимально

автоматизувати цикли обробки телеметрії, гарантувати безперервне оновлення компонентів та нативно інтегрувати інструменти проактивного моніторингу.

Таким чином, розроблена система не лише гарантує безперервну доставку оновлень і точне прогнозування стану обладнання, але й виступає фундаментальною умовою безпечної експлуатації розподілених IoT-мереж.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. I. Selinis et al., "Evaluation of the IEEE 802.11ax OFDMA and MU-MIMO in Dense Environments," in 2019 IEEE Global Communications Conference (GLOBECOM), 2019, pp. 1-6. https://doi.org/10.1109/GLOBECOM38437.2019.9013589.
2. K. Kosek-Szott et al., "Coexistence of Wi-Fi 6, 6E, and 7: Challenges and Opportunities," IEEE Network, vol. 35, no. 4, pp. 212-219, 2021. https://doi.org/10.1109/MNET.011.2100234.
3. D. Bankov et al., "OFDMA Uplink Scheduling in IEEE 802.11ax Networks," in 2020 IEEE International Conference on Communications Workshops (ICC Workshops), 2020, pp. 1-6. https://doi.org/10.1109/ICCWorkshops49005.2020.9145228.
4. E. Khorov, A. Kiryanov, A. Lyakhov, and G. Bianchi, "A Tutorial on IEEE 802.11ax High Efficiency WLANs," IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 197-216, 2019. https://doi.org/10.1109/COMST.2018.2871099.
5. D. Lopez-Perez et al., "IEEE 802.11be Extremely High Throughput: The Next Generation of Wi-Fi," IEEE Communications Magazine, vol. 57, no. 9, pp. 113-119, 2019. https://doi.org/10.1109/MCOM.001.1900338.
6. D. Deng et al., "IEEE 802.11ax: Highly efficient WLANs for intelligent information infrastructure," IEEE Wireless Communications, vol. 27, no. 2, pp. 182-189, 2020. https://doi.org/10.1109/MWC.001.1900464.
7. S. Nawaz et al., "A Survey on Rate Adaptation Algorithms in Wireless LANs," IEEE Access, vol. 7, pp. 13444-13468, 2019. https://doi.org/10.1109/ACCESS.2019.2893264.
8. M. Zubair et al., "Interference Management in IEEE 802.11ax WLANs: A Survey," IEEE Access, vol. 9, pp. 11234-11256, 2021. https://doi.org/10.1109/ACCESS.2021.3050123.

9. F. Wilhelmi et al., "Spatial Reuse in IEEE 802.11ax WLANs," IEEE Communications Magazine, vol. 59, no. 5, pp. 84-90, 2021. https://doi.org/10.1109/MCOM.001.2000632.

10. M. Yang et al., "Performance Evaluation of IEEE 802.11ax and 802.11ac in Dense Deployments," in 2022 IEEE International Conference on Communications (ICC), 2022, pp. 1-6. https://doi.org/10.1109/ICC45855.2022.9813583.

11. S. Bhattarai et al., "Throughput Analysis of IEEE 802.11ax Networks with Uplink Multi-User MIMO," IEEE Transactions on Vehicular Technology, vol. 68, no. 10, pp. 10003-10017, 2019. https://doi.org/10.1109/TVT.2019.2936279. R. Karmakar et al., "Quality of Service in Wireless LANs: A Survey," Journal of Network and Computer Applications, vol. 173, p. 102947, 2021. https://doi.org/10.1016/j.jnca.2020.102947.

12. G. Naik et al., "Can Wi-Fi 7 Meet the Requirements of Industrial IoT? A Performance Evaluation," IEEE Access, vol. 10, pp. 11567-11580, 2022. https://doi.org/10.1109/ACCESS.2022.3146201.

13. P. Gallo et al., "Performance Evaluation of Wireless Networks: A Comparative Study of Simulation Tools," IEEE Access, vol. 8, pp. 123456-123470, 2020. https://doi.org/10.1109/ACCESS.2020.3006789.

14. A. Hameed et al., "Coverage and Capacity Analysis of IEEE 802.11ac/ax Networks in Indoor Scenarios," International Journal of Wireless Information Networks, vol. 28, pp. 234-245, 2021. https://doi.org/10.1007/s10776-021-00512-3.

15. I. Kotuliak et al., "Performance comparison of IEEE 802.11ac and IEEE 802.11n standards in indoor environment," in 2018 16th International Conference on Emerging eLearning Technologies and Applications (ICETA), 2018, pp. 293-298. https://doi.org/10.1109/ICETA.2018.8572134.

Додаток А

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

(Презентація)