Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету )
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

# КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

**магістра**
(освітній ступінь))

на тему: **Методи і DevOps-засоби моніторингу, оновлення та прогнозування стану IoT-пристроїв**

Виконав: студент (ка) 6 курсу, групи СІм-61
спеціальності 123 «Комп'ютерна інженерія»
(шифр і назва спеціальності)

| | | |
|---|---|---|
| | (підпис) | Комарницький В.В. (прізвище та ініціали) |
| Керівник | (підпис) | Луцків А.М. (прізвище та ініціали) |
| Нормоконтроль | (підпис) | Луцик Н.С. (прізвище та ініціали) |
| Завідувач кафедри | (підпис) | Осухівська Г.М. (прізвище та ініціали) |
| Рецензент | (підпис) | Готович В.А. (прізвище та ініціали) |

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Факультет комп'ютерно-інформаційних систем і програмної інженерії
Кафедра комп'ютерних систем та мереж

**ЗАТВЕРДЖУЮ**
Завідувач кафедри *Осухівська Г.М.*

«*17*» *листопада* 2025 р.

# З А В Д А Н Н Я

## НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 «Комп'ютерна інженерія»*
(шифр і назва спеціальності)

студенту *Комарницькому Владиславу Володимировичу*
(прізвище, ім'я, по-батькові)

1. Тема проекту (роботи) *Методи і DevOps-засоби моніторингу, оновлення та прогнозування стану IoT-пристроїв*

Керівник проекту (роботи) *Луцків Андрій Мирославович, к.т.н., доц.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*14*» *листопада* 2025 року №*4/7-987*

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи *Принципи та особливості функціонування IoT-пристроїв, види IoT-пристроїв, протоколи передачі даних в IoT мережах, DevOps практики безперервної Доставки та оновлення програмного забезпечення*

4. Зміст роботи (перелік питань, які потрібно розробити)
*Вступ. 1. Аналіз методів і засобів монторингу, оновлення та прогнозування стану IoT-пристроїв. 2. Проєктування системи моніторингу, прогнозування та віддаленого оновлення стану IoT -пристроїв. 3. Програмна реалізація та налаштування IoT-інфраструктури та DevOps засобів 4. Охорона праці та безпека в надзвичайних ситуаціях. Висновки*

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
*1. Актуальність і мета дослідження. 2. Задачі дослідження, об'єкт і предмет, наукова новизна і практична цінність дослідження. 3. Класична архітектура системи моніторингу IoT-пристроїв 4. Роль хмарного рівня при організації IoT інфраструктури. 5. Архітектурні рівні та їх складові проєктованої системи. 6. Схема взаємодії компонентів архітектури. 7. Структура прошивки IoT -пристроїв. 8. Висновки*

6. Консультанти розділів роботи

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдання прийняв |
| *Охорона праці та* | *Осухівська Г.М., зав. каф. КС* | | |
| *безпека в надзвичайних* | | | |
| *ситуаціях* | *Стручок В.С., ст. викладач каф. ОХ* | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

7. Дата видачі завдання *17.11.2025 р.*

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Термін виконання етапів роботи | Примітка |
|---|---|---|---|
| *1.* | *Огляд літературних джерел. Постановка завдання на кваліфікаційну роботу.* | *17.11.2025р. – 23.11.2025р.* | *виконано* |
| *2.* | *Проєктування системи моніторингу, прогнозування та віддаленого оновлення стану IoT-пристроїв* | *24.11.2025р. – 02.12.2025р.* | *виконано* |
| *3.* | *Програмна реалізація та налаштування IoT-інфраструктури та devops засобів* | *03.12.2025р. – 10.12.2025р.* | *виконано* |
| *4.* | *Охорона праці та безпека в надзвичайних ситуаціях* | *04.12.2025р. – 10.12.2025р.* | *виконано* |
| *5.* | *Оформлення пояснювальної записки і графічного матеріалу* | *11.12.2025р. – 19.12.2025р.* | *виконано* |
| *6.* | *Попередній захист кваліфікаційної роботи магістра* | *16.12.2025р.* | *виконано* |
| *7.* | *Захист кваліфікаційної роботи магістра* | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

Студент _____ *Комарницький В.В.*
(підпис) (прізвище та ініціали)

Керівник проекту (роботи) _____ *Луцків А.М.*
(підпис) (прізвище та ініціали)

# АНОТАЦІЯ

Комарницький В.В. Методи і DevOps-засоби моніторингу, оновлення та прогнозування стану IoT-пристроїв: робота на здобуття кваліфікаційного ступеня магістра: спец. 123 — комп'ютерна інженерія / наук. кер. А.М. Луцків. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2025. — 83 с.

Ключові слова: метод, засіб, моніторинг, оновлення, прогнозування, DevOps, IoT.

У кваліфікаційній роботі проведено аналітичний огляд сучасних підходів до моніторингу, обслуговування та прогнозування стану IoT-пристроїв, проаналізовано архітектури побудови IoT-систем із використанням edge-, fog- та cloud-рівнів та досліджено особливості застосування DevOps-підходів у середовищах Інтернету речей.

Запропоновано та обґрунтовано багаторівневу архітектуру DevOps-орієнтованої IoT-системи, яка поєднує рівень IoT-пристроїв, edge-обчислень і хмарних сервісів. Розроблено метод визначення пріоритетів OTA-оновлення та прогнозування деградації, який поєднує поточний стан пристрою, час від останнього оновлення та кількість зафіксованих помилок. Запропоновано систему метрик для оцінювання ефективності роботи платформи, що включає середню доступність, середній час оновлення та точність прогнозування стану

Розроблено механізм збору та передавання телеметрії з використанням протоколів MQTT та HTTP, а також запропоновано DevOps-орієнтований процес безперервного оновлення IoT-вузлів, який включає CI/CD-конвеєр, формування підписок OTA-артефактів, доставку оновлень через edge-шлюзи та автоматичні механізми їх скасування.

# ABSTRACT

Komarnytskyi V.V. Methods and devops tools for monitoring, updating, and predicting the state of IoT devices. Master's Graduation Thesis: speciality 123 - Computer engineering/supervisor A.M. Lutskiv. Ternopil: Ternopil Ivan Puluj National Technical University, 2025. – 83 p.

Keywords: method, tool, monitoring, updating, forecasting, DevOps, IoT.

In the qualification thesis, an analytical review of modern approaches to monitoring, maintenance, and state prediction of IoT devices is carried out. Architectures of IoT systems based on edge, fog, and cloud layers are analyzed, and the specifics of applying DevOps approaches in Internet of Things environments are investigated.

A multi-layer DevOps-oriented IoT system architecture is proposed and substantiated, combining the IoT device layer, edge computing, and cloud services. A method for determining OTA update priorities and predicting device degradation is developed, which integrates the current device state, the time since the last update, and the number of detected errors. A system of metrics for evaluating the platform's performance is proposed, including average availability, average update time, and state prediction accuracy.

A mechanism for collecting and transmitting telemetry using MQTT and HTTP protocols is developed. In addition, a DevOps-oriented continuous update process for IoT nodes is proposed, which includes a CI/CD pipeline, generation and subscription of OTA artifacts, delivery of updates via edge gateways, and automatic rollback mechanisms.

ЗМІСТ

# ВСТУП

**Актуальність теми.** Сучасна тенденція розвитку інформаційних технологій демонструє стрімке поширення пристроїв Інтернету речей (IoT), які формують надзвичайно масштабні розподілені системи збору та обробки даних. За оцінками аналітичних агентств Gartner і Statista, до 2030 року кількість під'єднаних IoT-пристроїв сягне понад 30 мільярдів, що зумовлює потребу в нових методах їхньої підтримки, керування та технічного обслуговування.

Ефективне функціонування таких систем вимагає не лише безперебійного обміну даними, але й можливості оперативного оновлення програмного забезпечення, віддаленого моніторингу та прогнозування працездатності вузлів. Традиційні підходи до супроводу апаратно-програмних комплексів не забезпечують необхідної гнучкості, тому на передній план виходить методологія DevOps, яка інтегрує процеси розроблення, тестування, розгортання та контролю у єдиний безперервний життєвий цикл.

В останні роки спостерігається активне наукове і практичне вивчення проблем інтеграції DevOps із IoT-інфраструктурами. Серед провідних дослідників цієї тематики варто відзначити J. Wettinger, F. Leymann, S. Dustdar, D. Merkel, B. Fitzgerald, L. Atzori, A. M. Rahmani та M. Aazam, які розвивають концепції хмарно-периферійних архітектур, контейнеризації, CI/CD-автоматизації та інтелектуальної аналітики для систем Інтернету речей.

В українському науковому середовищі питання інтелектуальних і мультиагентних платформ, а також кіберфізичних систем досліджуються у працях А. Паламаря, А. Луцківа, М. Паламаря, С. Лупенка, Н. Шаховської, Р. Буція та ін. Їхні результати, опубліковані у виданнях CEUR, Springer CCIS, ITTAP та IEEE Xplore, закладають теоретичні основи для створення адаптивних систем управління та аналізу даних у середовищі IoT.

Разом із тим, інтеграція апаратного моніторингу, прогнозування технічного стану та автоматизованого оновлення в єдиний DevOps-цикл поки що залишається малодослідженою. Невирішеним є також питання розроблення гібридної хмарної

платформи, здатної забезпечувати узгоджену роботу сотень розподілених вузлів, оптимізацію оновлень і підтримку високої доступності системи.

Тому розроблення методів і DevOps-засобів для моніторингу, оновлення та прогнозування стану IoT-пристроїв є актуальною науково-технічною задачею, розв’язання якої сприятиме підвищенню ефективності, надійності та масштабованості сучасних комп’ютерно-інженерних рішень.

**Мета кваліфікаційної роботи** полягає у розробленні та дослідження методів і DevOps-засобів моніторингу, оновлення та прогнозування стану IoT-пристроїв, що забезпечують підвищення надійності, керованості та безперервності функціонування розподілених апаратно-програмних систем.

Для того, щоб досягнути мети роботи необхідно виконати розв’язок наступних **задач**:

– проаналізувати сучасні підходи до впровадження DevOps-методології в системах Інтернету речей, зокрема методи CI/CD-автоматизації, контейнеризації та моніторингу;

– дослідити існуючі програмні та апаратні засоби збору телеметрії, обробки даних і віддаленого керування IoT-вузлами;

– обґрунтувати математичну модель оцінювання технічного стану IoT-пристроїв, що враховує комплекс телеметричних параметрів;

– запропонувати метод визначення пріоритету оновлення пристроїв на основі поєднання поточного стану, часу від останнього оновлення та кількості виявлених збоїв;

– реалізувати модуль візуалізації та аналітики даних для спостереження за ключовими параметрами системи;

– провести експериментальні дослідження ефективності розроблених методів і засобів та сформулювати рекомендації щодо їх застосування.

**Об’єкт дослідження**: процеси функціонування, моніторингу і технічного обслуговування IoT-пристроїв у розподілених апаратно-програмних системах, що взаємодіють у хмарно-периферійному середовищі.

**Предмет дослідження**: методи та DevOps-засоби організації безперервного життєвого циклу IoT-пристроїв, які забезпечують автоматизований моніторинг телеметрії, прогнозування технічного стану та віддалене оновлення програмного забезпечення.

**Методи дослідження:** Поставлені у кваліфікаційній роботі задачі розв'язано із застосуванням таких методів:

– аналіз та узагальнення – при дослідженні сучасних підходів до моніторингу, оновлення та управління IoT-інфраструктурами, при аналізі DevOps-концепцій

– методи проєктування розподілених апаратно-програмних систем – при розробленні загальної архітектури системи моніторингу IoT-пристроїв;

– DevOps-методи та інструменти автоматизації – при побудові CI/CD-конвеєрів для збирання, тестування та розгортання прошивок IoT-пристроїв;

– – методи обробки та аналізу часових рядів – при аналізі телеметричних даних IoT-пристроїв;

– програмування та експериментальне моделювання – при реалізації програмного забезпечення IoT-вузлів, edge-шлюзів і хмарних сервісів, а також при тестуванні роботи системи в умовах, наближених до реальної експлуатації.

**Наукова новизна отриманих результатів.** Наукова новизна полягає в наступному:

– уперше запропоновано формалізовану модель інтегральної оцінки технічного стану IoT-пристрою в DevOps-орієнтованій системі, яка враховує одночасний вплив апаратних, програмних і комунікаційних параметрів та дозволяє формалізувати показник працездатності IoT-пристроїв для прийняття рішень щодо їх обслуговування;

– уперше запропоновано адаптовану формалізовану модель управління ризиками програмного забезпечення в Agile-проєктах на основі класифікації SEI, яка інтегрує ризики вимог, архітектурних рішень та процесів розробки в єдину інформаційну модель і дозволяє здійснювати їх системну ідентифікацію та оцінювання на ранніх етапах життєвого циклу програмного продукту;

– удосконалено метод оцінювання пріоритетності ризиків у гнучких методологіях розробки шляхом поєднання експертних оцінок, метрик якості вимог та результатів аналізу критичного шляху проєкту, що дало змогу підвищити обґрунтованість прийняття управлінських рішень щодо пом’якшення ризиків без порушення ітеративної природи Agile-процесів;

– набули подальшого розвитку підходи до автоматизації процесів управління ризиками програмного забезпечення за рахунок розроблення багаторівневої архітектури програмного засобу, яка забезпечує трасування ризиків, вимог і метрик якості між ітераціями Agile-проєкту, що сприяє підвищенню прозорості процесу розробки та зниженню ймовірності критичних відхилень під час реалізації програмних проєктів.

**Практичне значення одержаних результатів.** Практична цінність результатів кваліфікаційної роботи полягає у розробці та реалізації комплексної DevOps-орієнтованої системи моніторингу, оновлення та прогнозування стану IoT-пристроїв, яка може бути використана в реальних IoT-інфраструктурах різного масштабу.

**Публікації.** Результати кваліфікаційної роботи апробовані на XIV міжнародній науково - технічній конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (11-12 грудня 2025 р.) Тернопільського національного технічного університету імені Івана Пулюя та на XIII науково-технічній конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (17-18 грудня 2025 року) як тези конференцій.

1. Луцків А.М., Комарницький В.В. DevOps-підхід до автоматизації CI/CD у розподілених IoT-системах. Матеріали XIV міжнародної науково - технічної конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (11-12 грудня 2025 р.) Тернопільського національного технічного університету імені Івана Пулюя. Тернопіль: ТНТУ. 2025. С. 296.

2. Луцків А.М., Комарницький В.В. Проєктування системи моніторингу та віддаленого оновлення IoT-пристроїв. Матеріали XIII науково-технічної

конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (17-18 грудня 2025 року). Тернопіль: ТНТУ. 2025. С. 131.

**Структура роботи.** Кваліфікаційна робота містить пояснювальну записку та графічний матеріал. До складу записки входить вступ, 4 розділи, загальні висновки, список використаних джерел і додатки. Обсяг роботи: пояснювальна записка – 83 арк. формату А4, графічна частина – 6 аркушів формату А1.

# РОЗДІЛ 1
# АНАЛІЗ МЕТОДІВ І ЗАСОБІВ МОНТОРИНГУ, ОНОВЛЕННЯ ТА ПРОГНОЗУВАННЯ СТАНУ ІОТ-ПРИСТРОЇВ

## 1.1. Аналіз проблеми моніторингу та обслуговування IoT-систем

У контексті стрімкого зростання кількості IoT-пристроїв у різних галузях, зокрема, промисловості, аграрного сектору, “розумних міст”, охорони здоров’я, питання надійності, безперервного моніторингу та прогнозування відмов стають критично важливими.

Традиційні підходи до технічного обслуговування апаратних пристроїв, які включають реактивне обслуговування після збою або профілактичне, що виконується за графіком, не враховують поточний стан обладнання і можуть бути неефективними або надмірними.

Сучасні дослідження все більше зосереджуються на прогнозувальному обслуговуванні, яке поєднує IoT, аналітику даних і машинне навчання [1].

Моніторинг у IoT-системах включає збір телеметрії, наприклад, значення показників температури, вібрації, споживання струму, рівня сигналу. Після цього виконується передача даних, їх зберігання, обробка та візуалізація. Термін «обслуговування» у такому контексті інтерпретується як оновлення та проведення автоматизованих дій реагування на виявлені аномалії. Прогнозування стану асоціюється з передбаченням деградації або відмови на основі минулих даних.

Огляд останніх публікацій показує, що напрям моніторингу та обслуговування IoT-систем розвивається у трьох взаємопов’язаних площинах:

– архітектура систем збору й аналізу телеметрії;

– алгоритми прогнозування відмов;

– автоматизовані механізми оновлення програмного забезпечення (OTA-оновлення) у рамках DevOps-підходів.

У класичному вигляді архітектура системи моніторингу включає три рівні, як показано на рис.1.1. До них належать:

– пристрої IoT (edge-рівень), що вимірюють фізичні параметри;
– шлюзи або контролери, які агрегують і передають дані;
– хмарну платформу, де виконується аналітика, зберігання та візуалізація.

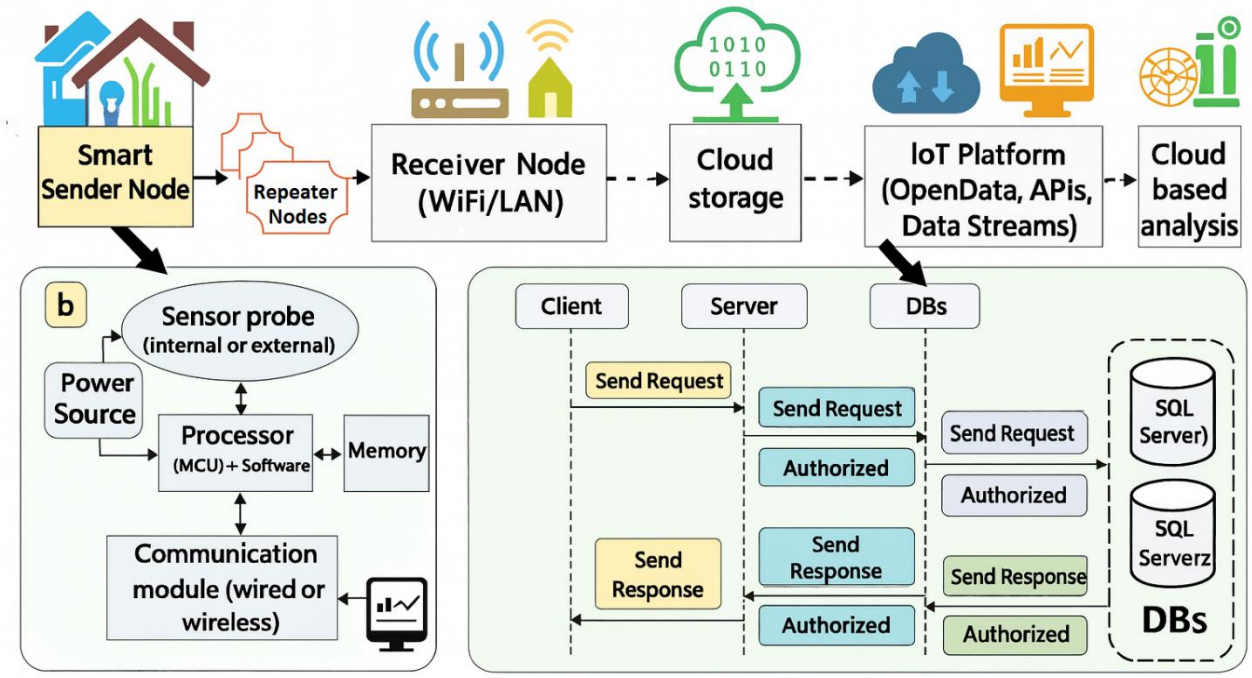


Рис. 1.1. Приклад архітектури системи моніторингу IoT

Як видно з рис. 1.1, edge-рівень містить сенсори й контролери, fog-рівень відповідає за агрегацію даних і їх попередню обробку, а cloud-рівень використовується для задач аналітики, зберігання та візуалізації.

Такі рішення описано в роботах J. Wettinger, F. Leymann та S. Dustdar, які запропонували концепцію DevOps-інтеграції для IoT. Вона передбачає об’єднання процесів збору даних, оновлення й моніторингу в єдиний життєвий цикл системи [2].

Подібні принципи застосовують M. Aazam і A. Rahmani, акцентуючи на використанні fog-та edge-обчислень для мінімізації затримок і зниження навантаження на хмарну інфраструктуру [3-4].

Важливою тенденцією останніх років є перехід від реактивного обслуговування до інтелектуального прогнозувального [5-7].

У таких системах сенсори постійно збирають телеметрію температуру, вібрацію, споживання енергії, рівень сигналу тощо. Отримані дані аналізуються алгоритмами машинного навчання, що дозволяє прогнозувати наближення відмови й планувати обслуговування без зупинки процесу.

Дослідження [8,9] показали ефективність застосування моделі AdaBoost для виявлення ознак деградації промислового обладнання, що дало змогу підвищити точність прогнозу до 90 %.

У роботі [10] запропоновано підхід Software-Defined Sensing, який уніфікує різнорідні сенсорні дані й забезпечує високу чутливість системи до нештатних ситуацій.

Для часових рядів, характерних для IoT-моніторингу, дедалі ширше застосовуються рекурентні нейронні мережі (RNN, LSTM), здатні враховувати динаміку зміни параметрів.

Разом з тим, у працях [11,12] наголошується, що надто складні моделі не завжди сумісні з енергообмеженими пристроями, тому актуальними стають «легкі» моделі з адаптивним перенавчанням. Приклад архітектури з потоками даних представлено на рис. 1.2.

Основними компонентами та потоками даних представленими на рис. 1.2 є:

– сенсори, актуатори та інші апаратні засоби, які генерують або отримують дані із фізичного середовища;

– шлюз, що забезпечує з’єднання пристроїв із хмарними сервісами;

– компоненти Cloud Gateway та Streaming data processor, що виконують приймання потокових даних і їх попередню обробку;

– сховища даних типу Data lake та Big data warehouse забезпечують довготривале збереження телеметрії;

– модулі штучного інтелекту виконують задачі аналізу даних та побудови прогнозів;

– прикладні сервіси Data Analytics та Control Applications забезпечують візуалізацію, аналітику й управління пристроями;

– Web/Mobile Application представляють користувацькі інтерфейси доступу до системи;

– Device/User Administration, Security Monitoring — підсистеми безпеки та адміністрування.

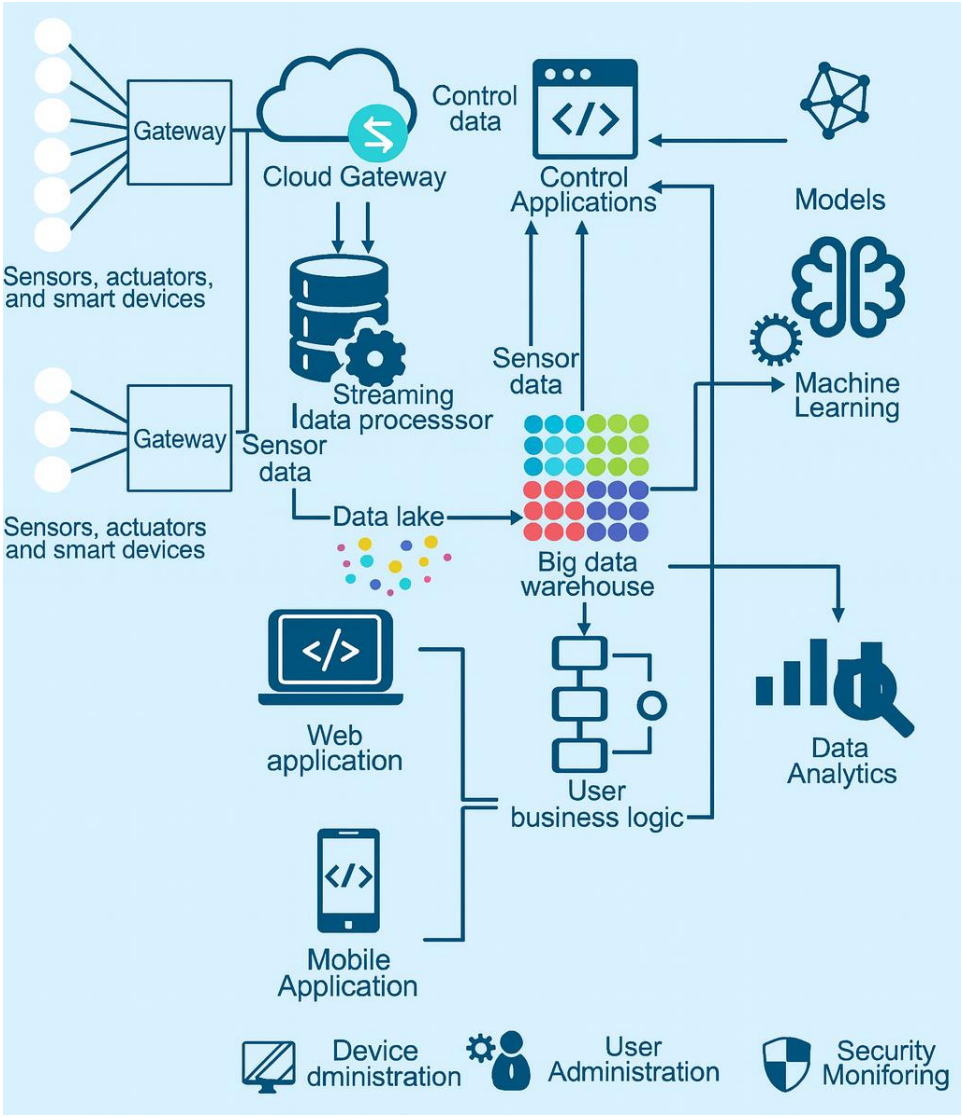


Рис. 1.2. Приклад архітектури та потоків даних у системах моніторингу та прогнозування стану IoT-пристроїв

Таким чином, рис. 1.2 демонструє повний цикл обробки IoT-даних починаючи від сенсорів до аналітики та прийняття рішень у хмарному середовищі.

Ключовим аспектом підтримки IoT-інфраструктури є безпечне дистанційне оновлення прошивок (OTA — Over-the-Air).

Розв’язання цього завдання пропонують такі платформи, як Mender, Balena, AWS IoT Device Management та Azure IoT Hub. Вони дозволяють DevOps-командам централізовано розгортати нові версії програмного забезпечення, перевіряти цілісність оновлення та виконувати автоматичне відновлення (rollback) у разі помилки.

Завдяки інтеграції з CI/CD-сервісами (GitHub Actions, GitLab CI, Jenkins) забезпечується безперервне розгортання в гібридних середовищах.

Узагальнені результати використовуваних підходів, методів і засобів до моніторингу, прогнозування та оновлення IoT-систем представлені у табл. 1.1.

*Таблиця 1.1*

**Сучасні методи і засоби моніторингу IoT-інфраструктури**

| Методи та інструменти | Сфера застосування | Основні результати |
|---|---|---|
| ML (AdaBoost) | Промислові системи | Підвищення точності прогнозу до 90 % |
| Software-Defined Sensing | Гібридні мережі IoT | Зменшення кількості помилкових сповіщень |
| DevOps підхід | Хмарно-периферійні системи | Автоматизація оновлень у розподілених середовищах |
| Легкі моделі ML | Багатогалузеві IoT-системи | Оптимізація використання ресурсів пристрою |

Результати проведеного аналізу показують, що попри значну кількість робіт, лише незначна їх частина пропонує комплексні рішення, у яких поєднано моніторинг, прогнозування й DevOps-оновлення в єдиній системі.

Серед викликів щодо реалізації таких технологій є обмеження ресурсів, безпека, масштабованість та адаптація моделей.

Це підтверджує доцільність розробки і впровадження DevOps-рішень, які забезпечать автоматизований моніторинг, прогнозування стану та безпечне оновлення IoT-пристроїв у реальному часі.

## 1.2. Організація DevOps-процесу для IoT-систем на основі сучасних підходів

У сучасній комп'ютерній інженерії DevOps виступає основним підходом до інтеграції розроблення, тестування, розгортання та підтримки програмних систем. Його головна мета полягає у забезпеченні безперервного життєвий циклу програмного продукту з високим рівнем автоматизації та контролю якості.

Для IoT-систем, які поєднують тисячі або навіть мільйони розподілених пристроїв, DevOps набуває особливого значення. Тут необхідно не лише підтримувати актуальність програмного забезпечення на різних вузлах, але й гарантувати стабільність роботи та безпеку оновлень у реальному часі.

Класичний DevOps включає низку етапів, які утворюють циклічний процес CI/CD. Структуру цього процесу з використанням сучасних підходів проілюстровано на рис. 1.3.

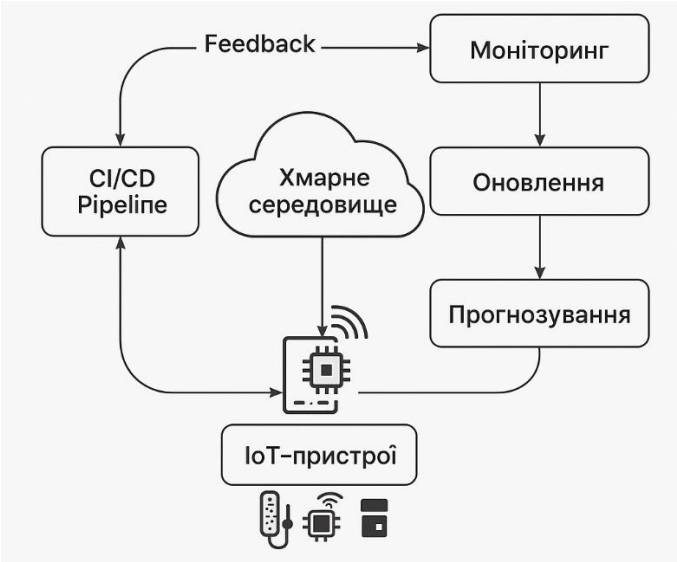


Рис. 1.3. Узагальнена схема DevOps-процесу для IoT-систем на основі сучасних підходів

Як видно з рис. 1.3, основними складовими DevOps-процесу є:

– Continuous Integration (CI) – безперервне об’єднання коду, перевірка сумісності нових змін і автоматичне тестування;

– Continuous Deployment (CD) – автоматизоване розгортання оновлень у робочому середовищі;

– Monitoring – постійне відстеження стану системи та пристроїв після оновлення;

– Feedback – зворотний зв’язок, який дозволяє швидко реагувати на помилки й удосконалювати систему.

Для традиційних веб- або серверних застосунків CI/CD процес реалізується у хмарі або локальному дата-центрі [13]. В IoT-середовищі ситуація ускладнюється тим, що розгортання відбувається не на одному сервері, а на великій кількості фізичних пристроїв, часто з обмеженими ресурсами, низькою пропускною здатністю каналів зв’язку та нестабільною мережею.

Особливості DevOps у контексті IoT включає ряд факторів. Перша особливість полягає у гібридності середовища, що передбачає виконання частини процесів у хмарі, зокрема, керування оновленнями та аналітика, а частина процесів виконується на периферії edge-пристроями.

Інший фактор передбачає особливі вимоги до безпеки при використанні IoT та полягає у необхідності підпису цифровим сертифікатом в процесі оновлення прошивок. Це дозволяє забезпечити контрольований процес оновлення та запобігання несанкціонованим змінам firmware.

Ще одна особливість полягає у підтримці OTA-оновлень, тобто повинна бути забезпечена можливість віддалено передавати й встановлювати нові версії прошивки без фізичного втручання.

Важливою особливістю процесу DevOps в IoT інфраструктурі є обмежені ресурси пристроїв. Це породжує необхідність мінімізувати обсяг оновлень, використовуючи диференційні пакети (delta updates).

DevOps процес керування IoT інфраструктурою передбачає зворотню телеметрію. Після оновлення пристрій надсилає інформацію про свій стан, що дозволяє аналізувати ефективність оновлень.

## 1.3. DevOps-платформи при організації хмарних сервісів обробки даних з IoT пристроїв

Стрімкий розвиток Інтернету речей зумовив зростання обсягів даних, що надходять від розподілених сенсорних мереж, вбудованих систем і периферійних пристроїв. Традиційні локальні обчислювальні інфраструктури дедалі частіше не здатні ефективно обробляти такі обсяги інформації, забезпечувати масштабованість та безперервну доступність сервісів. У зв’язку з цим ключову роль у сучасних IoT-системах відіграють хмарні обчислення, які в поєднанні з підходами DevOps формують гнучке та автоматизоване середовище обробки даних.

Хмарні сервіси дозволяють організувати централізоване зберігання, аналіз та візуалізацію телеметрії IoT-пристроїв без необхідності розгортання власних центрів обробки даних. Дані передаються та обробляються через мережу Інтернет, що забезпечує доступ до них з будь-якого географічного розташування, включно з мобільними платформами та edge-вузлами. Завдяки цьому хмарна інфраструктура стала базовим елементом сучасних DevOps-орієнтованих IoT-рішень.

На сьогодні існує кіклька моделей надання хмарних сервісів. У практиці хмарних обчислень виокремлюють три базові моделі надання сервісів, кожна з яких передбачає різний рівень відповідальності між постачальником послуг і користувачем (рис. 1.4):

– Infrastructure as a Service (IaaS);
– Platform as a Service (PaaS);
– Software as a Service (SaaS).

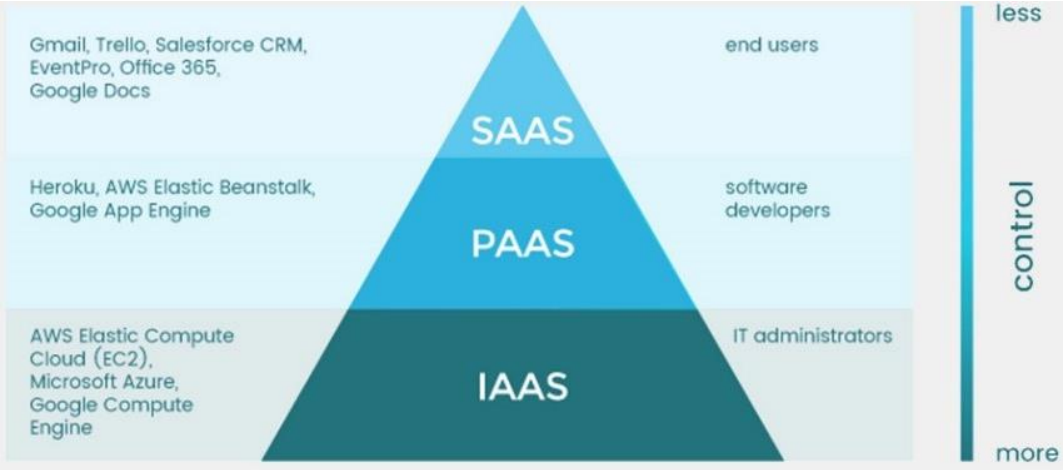


Рис. 1.4. Типи сервісів як послуг

Кожна з цих моделей по-різному інтегрується з DevOps-процесами та використовується для обробки даних від IoT-пристроїв залежно від вимог до керування, гнучкості та масштабування.

Модель IaaS передбачає надання користувачеві віртуалізованих обчислювальних ресурсів: серверів, мережі, систем зберігання та засобів віртуалізації у режимі «на вимогу». Користувач отримує повний контроль над операційними системами, програмним забезпеченням та мережевими конфігураціями, тоді як фізична інфраструктура обслуговується провайдером.

IaaS (рис. 1.5) є найбільш гнучкою моделлю з точки зору DevOps, оскільки дозволяє:

– реалізувати Infrastructure as Code (IaC);

– автоматизувати розгортання середовищ;

– створювати масштабовані середовища для обробки великих потоків IoT-даних;

– організовувати тестові та експериментальні середовища.

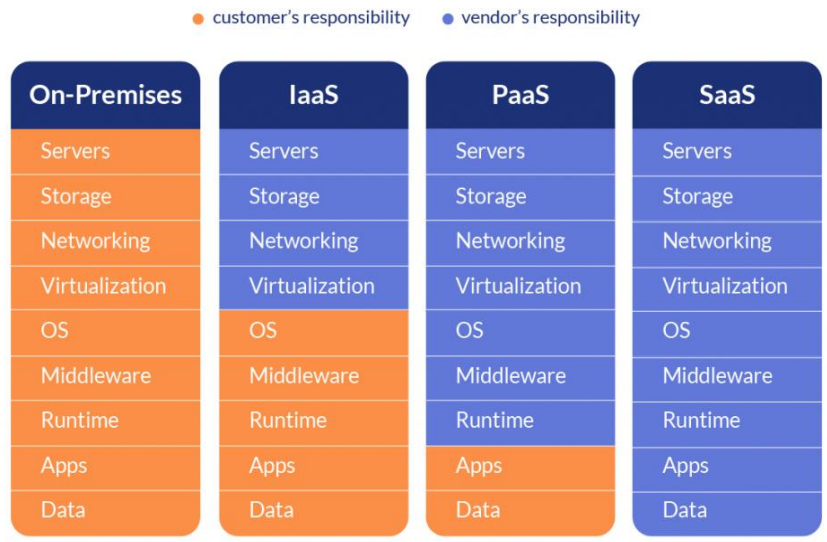


Рис. 1.5. Особливості хмарних сервісів

Типовими прикладами IaaS-платформ є Amazon Web Services (EC2), Microsoft Azure Virtual Machines, Google Compute Engine. Саме ця модель широко застосовується для аналізу великих обсягів телеметрії, машинного навчання та прогнозування технічного стану IoT-пристроїв.

Водночас IaaS має і низку обмежень: необхідність самостійного адміністрування операційних систем, налаштування безпеки та контролю витрат у періоди пікових навантажень. Базовий принцип використання IaaS для обробки IoT-даних показано на рис. 1.6.

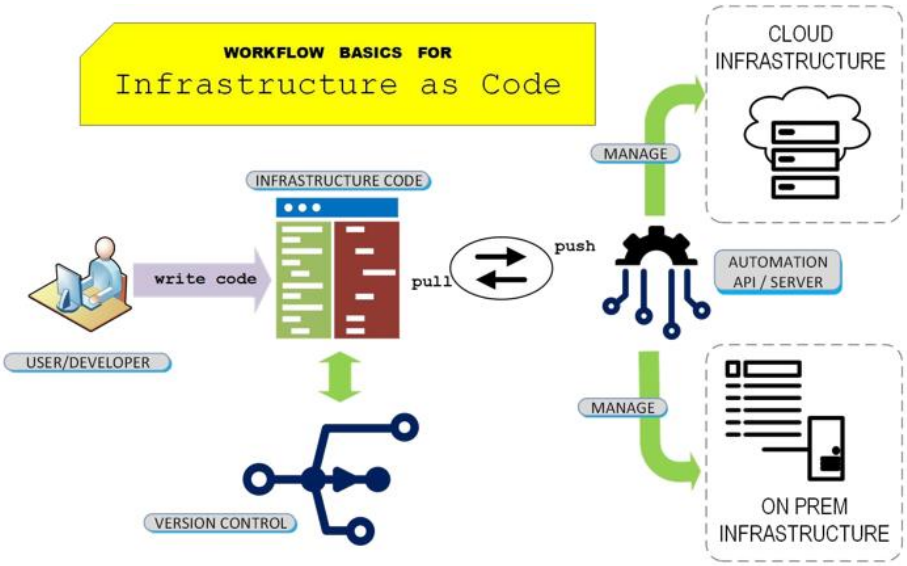


Рис. 1.6. Базовий сценарій використання IaaS

Модель PaaS орієнтована на спрощення процесу розроблення та розгортання прикладних систем. Постачальник PaaS надає не лише інфраструктуру, але й середовище виконання, проміжне програмне забезпечення та інструменти розробки.

Для IoT-систем PaaS є привабливою через:

– швидке створення сервісів збору та обробки даних;
– спрощену інтеграцію API;
– підтримку популярних мов програмування;
– зменшення операційного навантаження на команду DevOps.

До типових PaaS-рішень належать Google App Engine, Heroku, AWS Elastic Beanstalk. Вони часто застосовуються для розроблення веб-панелей, REST-API та сервісів управління IoT-пристроями.

Недоліком PaaS є обмежений контроль над інфраструктурою та залежність від конкретного постачальника, що може ускладнювати інтеграцію з нестандартними або промисловими IoT-рішеннями.

Модель SaaS (рис. 1.7) передбачає надання користувачеві повністю готового програмного рішення, яке розміщується та обслуговується провайдером. Користувач отримує доступ до функціоналу через веб-інтерфейс або API без необхідності адміністрування інфраструктури.

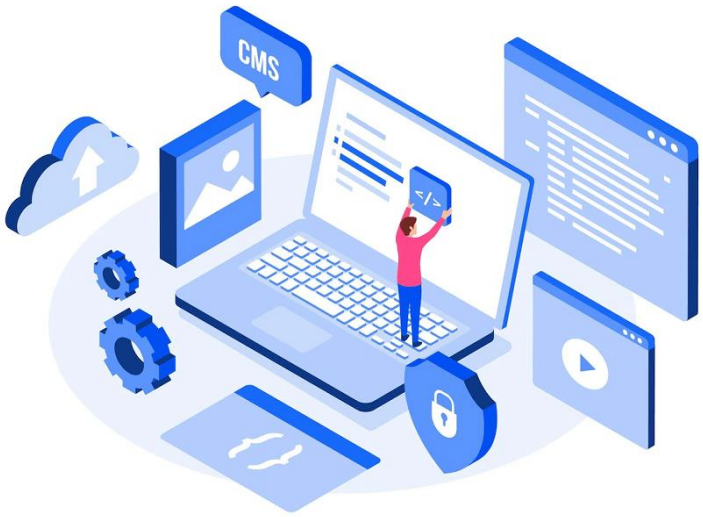


Рис. 1.7. SaaS

SaaS-платформи широко використовуються для:

– візуалізації IoT-даних;
– моніторингу стану пристроїв;
– спільної роботи та аналітики.

Прикладами SaaS є Google Workspace, JIRA, Dropbox, Grafana Cloud, а також спеціалізовані IoT-платформи. Основною перевагою SaaS є мінімальні вимоги до обслуговування, однак користувач повністю залежить від стабільності та політик постачальника сервісу.

Поєднання хмарних сервісів із DevOps-підходами дозволяє:

– автоматизувати обробку телеметрії;
– забезпечити безперервне оновлення компонентів системи;
– реалізувати масштабування залежно від навантаження;
– інтегрувати моніторинг, логування та аналітику.

Таким чином, вибір між IaaS, PaaS та SaaS у DevOps-орієнтованих IoT-системах залежить від рівня контролю, необхідної гнучкості та складності обробки даних.

## 1.4. Висновки до розділу

1. Проведено аналітичний огляд сучасних підходів до моніторингу, обслуговування та прогнозування стану IoT-пристроїв, що дозволило встановити обмеження традиційних методів та обґрунтувати доцільність переходу до прогнозувального обслуговування на основі аналізу телеметричних даних.

2. Проаналізовано архітектури побудови IoT-систем із використанням edge-, fog- та cloud-рівнів, а також моделі потокової обробки даних, що дало змогу визначити ключові вимоги до масштабованості, затримок передачі, обробки великих обсягів даних і безперервного моніторингу технічного стану пристроїв.

3. Досліджено особливості застосування DevOps-підходів у середовищах Інтернету речей, включаючи CI/CD-процеси, OTA-оновлення та зворотну

телеметрію, що дозволило обґрунтувати необхідність автоматизації життєвого циклу IoT-пристроїв з урахуванням обмежених ресурсів і вимог до безпеки.

4. Проаналізовано хмарні моделі надання сервісів IaaS, PaaS та SaaS у контексті DevOps-орієнтованих IoT-систем, що дало змогу визначити їх переваги та обмеження при організації обробки телеметрії, моніторингу, аналітики та оновлення програмного забезпечення.

5. Встановлено, що комплексне поєднання хмарних сервісів, DevOps-інструментів і методів аналізу даних є найбільш перспективним підходом, який забезпечує автоматизований моніторинг, прогнозування технічного стану та безпечне оновлення IoT-пристроїв.

# РОЗДІЛ 2
# ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ, ПРОГНОЗУВАННЯ ТА ВІДДАЛЕНОГО ОНОВЛЕННЯ СТАНУ ІОТ-ПРИСТРОЇВ

## 2.1. Загальна архітектура системи та вибір програмно-апаратних компонентів

Проєктування системи моніторингу, прогнозування та віддаленого оновлення стану IoT-пристроїв потребує формування цілісної архітектури, що поєднує апаратну, програмну та мережеву складові.

Вона повинна забезпечувати безперервний збір телеметрії, обробку даних у режимі реального часу, можливість виконання OTA-оновлень та зручне масштабування. Окрім цього, система моніторингу повинна забезпечувати такі критерії якості як висока надійність і відмовостійкість, а також безпека комунікацій.

Будь-яка система моніторингу стану IoT-пристроїв у DevOps-парадигмі складається щонайменше з трьох функціональних рівнів.

Перший рівень – це рівень пристроїв (Device Layer), який включає сенсори, мікроконтролери, мікрокомп’ютери та виконавчі механізми. На цьому рівні формуються пакети телеметричних даних та виконується початкова обробка.

Наступний рівень – периферійний рівень. Він реалізує агрегацію даних, буферизацію, попередню аналітику, розподілення навантаження та обробку подій, що важливо для зниження трафіку та затримок.

Третій рівень пов’язаний з хмарними сервісами. Він забезпечує зберігання великих обсягів даних, містить набори інструментів для проведення глибокої аналітики, підтримує побудову моделей прогнозування, DevOps-процеси, керування оновленнями та візуалізацію.

У роботі пропонується архітектура платформи моніторингу та прогнозування стану IoT пристроїв з використанням DevOps практик та інструментів, яка представлена на рис. 2.1.


Рис. 2.1. Узагальнена багаторівнева архітектура DevOps-орієнтованої IoT-платформи

Пристроями, які безпосередньо виконують вимірювання показників певних об’єктів чи параметрів стану середовища є IoT-пристрої.

Оскільки система повинна підтримувати моніторинг і можливість OTA-оновлення, критерії вибору кінцевих пристроїв включають:

- наявність Wi-Fi / Ethernet / LTE;
- достатній обсяг пам’яті для прошивки і її оновлення;
- енергоефективність;
- підтримку легковагових протоколів (MQTT/CoAP);
- можливість роботи в edge-режимі.

Типові варіанти апаратних рішень для кінцевих пристроїв представлені у табл. 2.1.

*Таблиця 2.1*

**Типові варіанти платформ на рівні IoT-пристроїв**

| Платформа | Переваги | Обмеження | Рекомендоване застосування |
|---|---|---|---|
| ESP32 | Дешевий, Wi-Fi/BLE, OTA, багато бібліотек | Обмежена пам’ять | Сенсорні вузли, low-power IoT |
| Raspberry Pi 4/5 | Повноцінний Linux, Docker, ML-edge | Високе енергоспоживання | Шлюзи, edge-аналітика |
| STM32 + LoRaWAN | Низьке споживання, промислові модулі | Складніша розробка | Віддалені вузли |
| NXP i.MX | Промислова надійність, Linux | Висока ціна | Промислові контролери |

Edge-рівень відіграє ключову роль у DevOps-архітектурі, оскільки він дає змогу зменшити навантаження на хмару та забезпечити виконання первинної діагностики стану сенсорів, які підключені до мікроконтролера. Окрім цього, даний рівень здійснює локальну маршрутизацію даних і підтримує автономність системи при поганому зв’язку.

При організації шлюзів на Edge-рівні необхідно, щоб забезпечувалась підтрика Docker-контейнерів, MQTT-брокерів, локальної бази часового ряду, а також OTA-переадресації. Серед платформ, які відповідають таким критеріям варто відмітити настпуні:

- Raspberry Pi CM4/5 у промисловому виконанні;
- Jetson Nano / Xavier NX – у випадку необхідності виконання локальної ML-аналітики;
- Advantech Industrial Gateways;
- Siemens IoT2000 Series.

Функціонуванн хмарного рівня при організації системи моніторингу, прогнозування та оновлення IoT пристроїв забезпечують наявність таких інстурментальних засобів як брокер повідомлень, сховища централізованого

зберігання телеметрії, DevOps-конвеєр (CI/CD), менеджер оновлення прошивок кінцевих пристроїв та моделі прогнозування на основі алгоритмів машинного навчання. Окрім цього, хмарний рівень надає веб-панелі та API.

Технологічний стек хмарного рівня забезпечують засоби, представлені у табл. 2.2.

*Таблиця 2.2*

**Типовий стек технологій та інструментів хмарного рівня**

| № з/п | Технологія | Інструменти |
|---|---|---|
| 1 | MQTT-брокери | EMQX Cloud, Mosquitto, AWS IoT Core |
| 2 | Зберігання часових рядів | InfluxDB Cloud, Timescale, AWS Timestream |
| 3 | Data Lake | S3 / MinIO |
| 4 | DevOps CI/CD | GitHub Actions, GitLab CI, Jenkins |
| 5 | OTA-оновлення | Mender, BalenaCloud, Eclipse hawkBit |
| 6 | Моніторинг | Prometheus + Grafana |
| 7 | Аналітика | Spark, Flink, Kafka Streams |

Для коректної роботи п’ятирівневої архітектури необхідно забезпечити стабільний і захищений канал передачі даних, що включає в себе:

– MQTT over TLS 1.2/1.3;
– VPN-тунелі для шлюзів;
– сегментація мережі VLAN для безпеки;
– QoS-рівні MQTT (0,1,2) залежно від критичності даних.

Особливо важливо забезпечити гарантовану доставку повідомлень та анти-збитковість даних при використанні edge-кешування.

На рис. 2.2 наведено схему функціонування системи моніторингу та прогнозування стану IoT-пристроїв із використання DevOps-процесу.

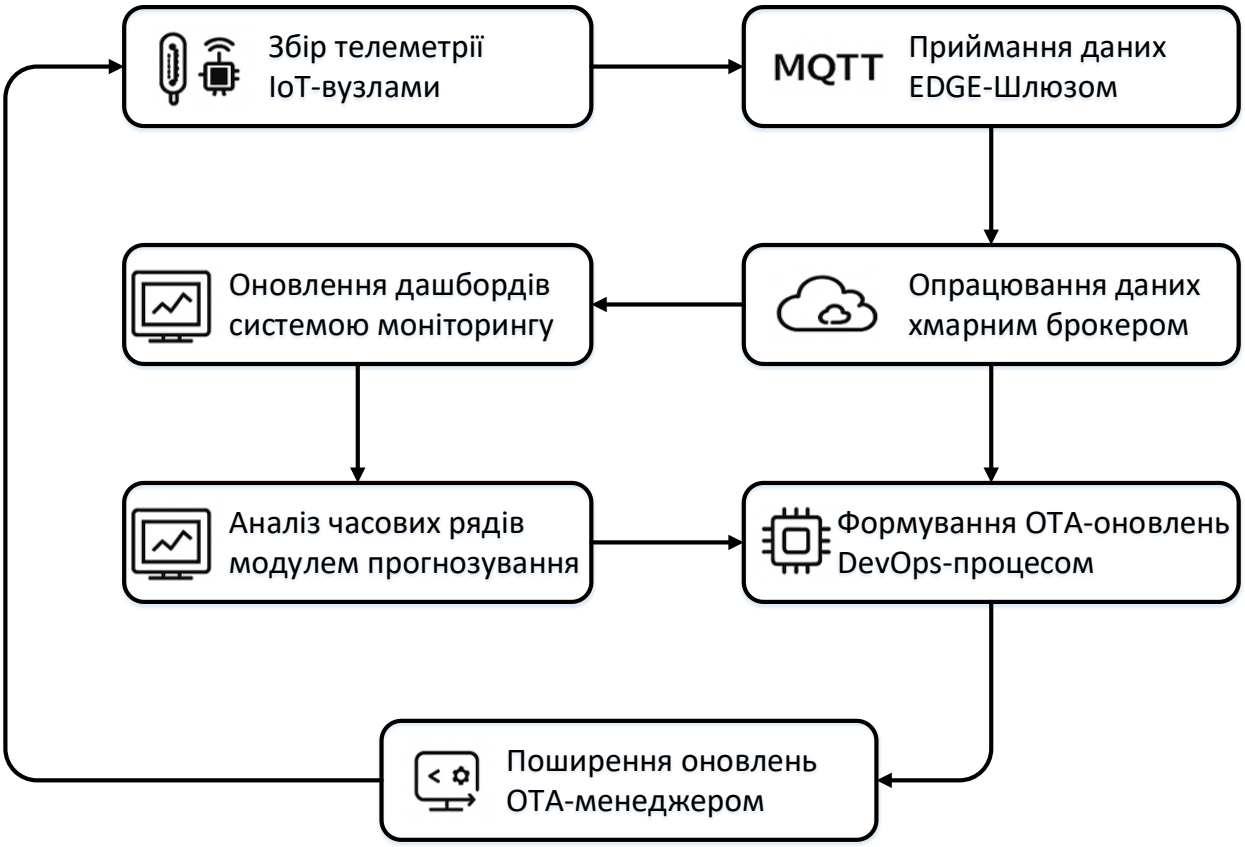


Рис. 2.2. Схема взаємодії компонентів архітектури системи моніторингу та прогнозування стану IoT-пристроїв

Опис алгоритму, який відповідає структурі моделі системи наступний:

– IoT-вузли збирають телеметрію та надсилають через MQTT;

– Edge-шлюз приймає дані, виконує попередню аналітику, кешування та відправку даних;

– хмарний брокер обробляє дані;

– система моніторингу оновлює дашборди;

– модуль прогнозування аналізує часові ряди;

– DevOps-процес формує OTA-оновлення;

– OTA-менеджер розповсюджує firmware назад до всіх IoT-вузлів;

– вузли встановлюють оновлення та надсилають зворотну телеметрію.

У результаті проведеного аналізу сформовано загальну архітектуру DevOps-орієнтованої IoT-системи, яка включає три основні функціональні рівні – рівень IoT-пристроїd, рівень edge-обчислень та хмарні сервіси. Обґрунтовано вибір

апаратних засобів, шлюзів і хмарних сервісів, що забезпечують надійний моніторинг, аналітику й безпечні OTA-оновлення. Отримана архітектура задає основу для проєктування DevOps-процесу та математичної моделі прогнозування стану та оновлення ПЗ кінцевих пристроїв.

## 2.2. Адаптація DevOps до IoT-середовища

Для традиційних веб- або серверних застосунків CI/CD процес реалізується у хмарі або локальному дата-центрі.

В IoT-середовищі ситуація ускладнюється тим, що розгортання відбувається не на одному сервері, а на великій кількості фізичних пристроїв, часто з обмеженими ресурсами, низькою пропускною здатністю каналів зв’язку та нестабільною мережею.

Особливості DevOps у контексті IoT включають:

- гібридність середовища – частина процесів відбувається у хмарі (керування оновленнями, аналітика), а частина на периферії (edge).
- вимоги до безпеки – оновлення повинні бути підписані цифровим сертифікатом, щоб запобігти несанкціонованим змінам firmware;
- підтримка OTA-оновлень (Over-the-Air) – можливість віддалено передавати й встановлювати нові версії прошивки без фізичного втручання;
- обмежені ресурси пристроїв – необхідність мінімізувати обсяг оновлень, використовуючи диференційні пакети (delta updates);
- зворотна телеметрія – після оновлення пристрій надсилає інформацію про свій стан, що дозволяє аналізувати ефективність оновлень.

Загальну структуру DevOps-інфраструктури для IoT можна подати у вигляді п’яти функціональних шарів (рис. 2.1).

Рівень пристроїв (Device Layer) складається з сенсорів, контролерів, мікрокомп’ютерів (ESP32, Raspberry Pi), які виконують збір даних.

Рівень комунікації (Communication Layer) представляє протоколи MQTT, CoAP, HTTP, LoRaWAN, що забезпечують обмін даними між пристроями та шлюзами.

Рівень управління (Management Layer) включає в себе сервіси моніторингу, оновлення firmware, керування групами пристроїв.

Рівень аналітики (Analytics Layer) представляється сховищами даних, інструментами машинного навчання та прогнозування стану.

Рівень візуалізації та адміністрування (Application Layer) надає панелі керування, дашборди, інтерфейси користувача.

Основні інструменти DevOps для IoT представлено у табл. 2.3.

*Таблиця 2.3*

**Інструменти DevOps для IoT**

| Компонент | Приклади інструментів | Функціональне призначення |
|---|---|---|
| CI/CD | GitHub Actions, GitLab CI, Jenkins | Автоматизація збирання, тестування й оновлення |
| Контейнеризація | Docker, Podman | Ізоляція застосунків і швидке розгортання |
| Моніторинг | Prometheus, Grafana, Zabbix | Збір і візуалізація телеметрії |
| OTA-оновлення | Balena, Mender, AWS IoT Device Management | Віддалене оновлення firmware |
| Безпека | HashiCorp Vault, OpenSSL | Зберігання секретів, підпис firmware |
| Оркестрація | Kubernetes, K3s | Керування контейнерами в edge-середовищах |

Переваги використання DevOps у IoT включають в себе безперервність обслуговування, яка реалізується через оновлення та відбуваються без зупинки системи. Окрім цього, прискорюється процес розгортання шляхом зменшення часу між розробкою й доставкою нової версії ПЗ для кінцевих пристроїв.

Інтеграція DevOps у IoT забезпечує автоматичний контроль якості, зокрема, CI-процеси дозволяють тестувати оновлення до розгортання, а також прозорість і керованість, що полягає у здатності моніторингу на наданні інформації про стан усіх пристроїв у реальному часі.

Важливою перевагою DevOps у IoT є масштабованість. DevOps легко адаптується до збільшення кількості пристроїв без додаткових ресурсів.

DevOps-методологія у поєднанні з технологіями IoT створює нову парадигму керування розподіленими системами. Вона забезпечує безперервне оновлення, моніторинг і прогнозування стану пристроїв, що є ключовими умовами їхньої надійної експлуатації. Основними напрямами адаптації DevOps до IoT залишаються безпека, автономність edge-вузлів і аналітика телеметрії в реальному часі.

## 2.3. Методи і засоби збору, зберігання та обробки телеметричних даних

Ефективне функціонування IoT-систем значною мірою залежить від того, наскільки якісно організовано процес збору, зберігання та обробки телеметричних даних.

Ці дані є основою для моніторингу стану пристроїв, прогнозування відмов і реалізації процесів автоматичного обслуговування.

У системах DevOps-типу цей процес повинен бути безперервним, масштабованим і надійним, адже він забезпечує аналітику в реальному часі та зворотний зв’язок для CI/CD-процесів.

Збір даних у системах Інтернету речей здійснюється на рівні пристроїв (сенсорів, контролерів, мікрокомп’ютерів) і включає декілька етапів:

– первинна обробка на пристрої – вимірювання параметрів середовища, попередня фільтрація шумів, усереднення результатів;

– пакетування даних – формування повідомлень, що містять часову мітку, ідентифікатор пристрою, значення параметрів і службову інформацію;

– передача даних – за допомогою легковагових протоколів, таких як MQTT, CoAP, AMQP або HTTP REST API.

На рис. 2.3 представлено один з варіантів організації IoT інфраструктури. Найбільш популярним у сучасних IoT-рішеннях є протокол MQTT (Message Queuing Telemetry Transport), який реалізує модель «публікація–підписка». Він характеризується низькими вимогами до пропускної здатності мережі, що робить його ідеальним для пристроїв з обмеженими ресурсами.

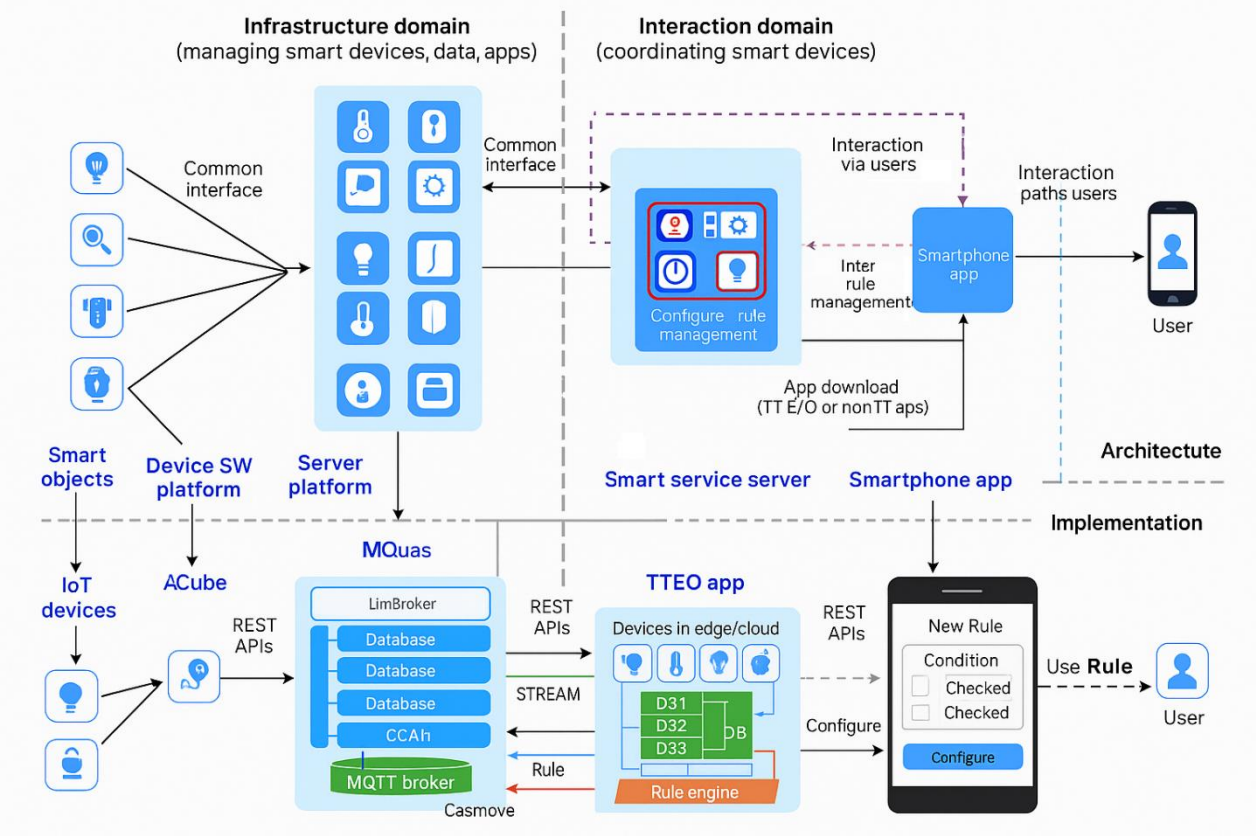


Рис. 2.3. Варіант організації IoT-інфраструктури для збору даних

У системах із підвищеними вимогами до безпеки використовується MQTT over TLS, що забезпечує шифрування переданих даних і аутентифікацію клієнтів.

Порівняння найбільш широко використовуваних протоколів передачі даних при організації IoT-інфраструктури наведено у табл. 2.4.

*Таблиця 2.4*

**Порівняння протоколів передачі даних в IoT**

| Протокол | Тип взаємодії | Основні переваги | Обмеження | Типові застосування |
|---|---|---|---|---|
| MQTT | Публікація–підписка | Низьке навантаження, QoS, простота | Не підходить для великих файлів | Сенсори, телеметрія |
| CoAP | Клієнт–сервер | Легкий, UDP, REST-подібний | Обмежена безпека | Побутові пристрої |
| HTTP REST | Клієнт–сервер | Універсальність, сумісність | Високе навантаження | Хмарні API |
| AMQP | Публікація–черга | Гарантована доставка | Висока складність | Банківські й промислові системи |

Після збору дані надходять до шлюзів або хмарних платформ, де проходять процес агрегації, буферизації та збереження.

Для цього використовуються дві основні категорії сховищ:

– Time-series databases (TSDB) – бази даних часових рядів (InfluxDB, TimescaleDB, Prometheus TSDB), які оптимізовані для зберігання показників, що надходять у вигляді потоку з часовими мітками.

– Data Lake / Data Warehouse – централізовані сховища великих обсягів даних (AWS S3, Hadoop HDFS, Snowflake), які дозволяють виконувати аналітику, машинне навчання та архівування історичних даних.

Для проміжного кешування телеметрії часто застосовуються брокери повідомлень, такі як RabbitMQ, Kafka або Mosquitto, які забезпечують черги обміну між пристроями та аналітичними сервісами. Це дає можливість зберігати дані навіть у разі тимчасової втрати зв’язку.

Схема загального процесу зображена на рис. 2.4.

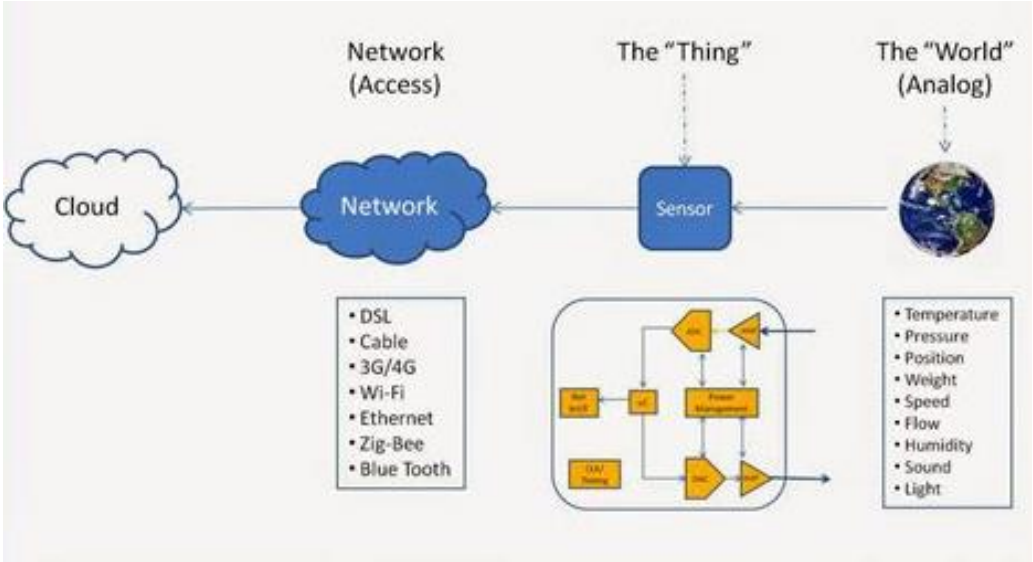


Рис. 2.4. Схема загального процесу збору і передачі показників з IoT-пристроїв

Процес обробка телеметрії включає фільтрацію та нормалізацію даних, що передбачає усунення аномалій, приведення одиниць вимірювання до єдиної нормалізованої шкали. Наступна фаза процесу обробки телеметрії преедбачає агрегацію даних, тобто об'єднання даних за часовими вікнами.

Аналітика в реальному часі забезпечується шляхом застосування потокових обчислень за допомогою фреймворків Apache Kafka Streams, Apache Flink, Spark Streaming. Прогнозування стану пристроїв виконується на основі алгоритмів ARIMA, LSTM, Random Forest, Gradient Boosting.

Візуалізацію показників стану IoT пристроїв можна виконати за допомогою Grafana, Kibana, Power BI, які інтегруються в DevOps-пайплайн.

У межах DevOps така обробка є частиною процесу Continuous Monitoring (CM), що забезпечує зворотний зв'язок до CI/CD і дає змогу автоматично коригувати конфігурацію пристроїв.

Для підвищення ефективності систем обробки телеметрії пропонується застосувати такі методи як:

– edge-аналітика – часткова обробка даних безпосередньо на пристрої або шлюзі для зменшення трафіку;

– data compression & sampling – використання алгоритмів стиснення та вибірки даних для економії пропускної здатності;

– batch and stream processing – поєднання періодичної та безперервної обробки залежно від задачі;

– anomaly detection pipelines – автоматичне виявлення відхилень у потоках телеметрії для запобігання відмовам.

Основними проблемами організації збору та обробки телеметрії є надмірний обсяг даних, що потребує ефективної фільтрації, забезпечення цілісності даних під час передачі, узгодження форматів даних від різнорідних сенсорів, вибір оптимальної архітектури для зберігання та потреба у безпеці ы контролі доступу до потоків даних.

У перспективі очікується активний розвиток інтелектуальних брокерів повідомлень, які зможуть здійснювати попередню аналітику та автоматичне виявлення аномалій на рівні мережевого вузла.

Також важливим напрямом є інтеграція DevOps-процесів із DataOps, що забезпечить повний цикл керування даними від їх збору до автоматизованої аналітики та оновлень системи.

## 2.4. Математичне забезпечення процесу моніторингу, оновлення і прогнозування стану IoT-пристроїв

Формалізацію процесу моніторингу, оновлення і прогнозування стану IoT-пристроїв пропонується виконати на основі математичного представлення DevOps-платформи для IoT. Таку платформу можна представити у вигляді множини пристроїв:

$$D = \{d_1, d_2, \dots, d_N\}, \tag{2.1}$$

де $D$ – множина усіх IoT-пристроїв;

$d_i$ – елемент множини $D$ (конкретний IoT-пристрій), $i = 1 \dots N$, $N$ – кількість пристроїв.

Кожен пристрій описується вектором поточних технічних параметрів:

$$x_i = [T_i(t), C_i(t), R_i(t), S_i(t)] \quad (2.2)$$

де $T_i(t)$ – температура процесора IoT-пристрою в момент часу $t$;

$C_i(t)$ – використання (завантаженість) процесора в момент часу $t$;

$R_i(t)$ – завантаженість оперативної пам'яті в момент часу $t$;

$S_i(t)$ – якість зв'язку в момент часу $t$.

Параметр $T_i(t)$ характеризує температуру центрального процесора або мікроконтролера IoT-пристрою і є одним із ключових індикаторів його технічного стану. Перевищення допустимих температурних меж може призводити до зниження продуктивності, нестабільної роботи або фізичної деградації електронних компонентів.

Для більшості вбудованих платформ, зокрема, ESP32, STM32, Raspberry Pi температура процесора залежить від інтенсивності обчислювального навантаження, частоти роботи процесора і тривалості активних сесій передавання даних. Окрім цього, на цей показник впливають зовнішні фактори, наприклад, температура навколишнього середовища та ефективність тепловідведення.

У межах IoT-систем параметр $T_i(t)$ використовується для виявлення аномальних режимів роботи (перегрів, теплові піки), оцінювання впливу OTA-оновлень на апаратне навантаження та формування прогнозу деградації пристрою. Якщо спостерігається тренд зростання температури у часі, особливо у поєднанні з іншими негативними показниками, то це може свідчити про поступову втрату стабільності функціонування пристрою.

Параметр $C_i(t)$ відображає поточний рівень завантаження центрального процесора, зазвичай у відсотках від максимальної обчислювальної потужності. Для IoT-пристроїв цей показник є критичним через обмежені апаратні ресурси.

Високі значення $C_i(t)$ можуть бути наслідком надмірної частоти опитування сенсорів, неефективної реалізації алгоритмів обробки даних, некоректної роботи програмного забезпечення або побічних ефектів після оновлення прошивки.

Моніторинг використання ЦП дозволяє оцінювати ефективність виконання програмних компонентів, виявляти перевантаження та програмні збої, а також

приймати рішення щодо оптимізації або оновлення прошивки пристроїв з обмеженими ресурсами.

Параметр $R_i(t)$ характеризує обсяг доступної або використаної оперативної пам'яті пристрою. Для вбудованих систем цей ресурс є одним із найбільш обмежених. Зменшення доступної пам'яті може бути спричинене витоками пам'яті, накопиченням тимчасових даних, некоректною обробкою мережевих пакетів або збоєм програмних модулів.

Аналіз динаміки $R_i(t)$ дозволяє виявляти потенційні програмні помилки, прогнозувати збої ще до їх виникнення, оцінювати стабільність роботи після OTA-оновлень і підвищувати надійність системи в цілому.

Параметр $S_i(t)$ відображає якість комунікаційного каналу, наприклад, RSSI для Wi-Fi, рівень сигналу для LoRaWAN або NB-IoT або затримки передачі та втрати пакетів.

Якість зв'язку безпосередньо впливає на стабільність передачі телеметрії, час доставки OTA-оновлень, частоту повторних передавань та енергоспоживання пристрою.

Низькі значення $S_i(t)$ можуть призводити до зростання навантаження на процесор і збільшення температури, що робить цей параметр важливим у комплексному аналізі стану пристрою.

Інтегральний показник стану

$$Q_i(t) = \sum_{k=1}^{4} \omega_k \cdot \frac{x_{i,k}(t)}{x_{k,max}} \tag{2.3}$$

де $\omega_k$ – експериментально визначені коефіцієнти важливості ($\sum \omega_k = 1$);

$x_{i,k}(t)$ – значення $k$-го показника $i$-го IoT-пристрою;

$x_{k,max}$ – максимальне значення $k$-го показника для групи однотипних пристроїв.

Якщо значення інтегрального показника стану пристрою $Q_i(t)$ більше за порогове мінімально допустиме значення значення $Q_{th}$, то пристрій перебуває у

нормі. В іншому випадку фіксується аномалія або формується потреба в оновленні прошивки.

Для прогнозування стану IoT-пристроїв використовується проста рекурентна модель типу ковзного середнього:

$$\hat{Q}_i(t+1) = \alpha Q_i(t) + (1-\alpha) Q_i(t-1), \quad (2.4)$$

де $\alpha$ – коефіцієнт згладжування, $\alpha \in [0,1]$.

У випадку, якщо $|Q_i(t) - \hat{Q}_i(t)| > \varepsilon$, то система формує попередження про можливу деградацію.

Кожен з IoT-пристроїв має пріоритет оновлення прошивки:

$$P_i = \beta_1 \cdot (1 - Q_i) + \beta_2 \cdot \Delta T_i + \beta_2 \cdot A_i, \quad (2.5)$$

де $T_i$ – час від останнього оновлення пристрою;

$A_i$ – кількість зареєстрованих помилок для і-го пристрою;

$\beta_j$ – вагові коефіцієнти.

Пристрої з найбільшими $P_i$ потрапляють у чергу оновлення DevOps-пайплайну (CI/CD).

Для комплексної перевірки ефективності розробленої DevOps-орієнтованої системи моніторингу, оновлення та прогнозування стану IoT-пристроїв використовується набір кількісних показників, які дозволяють оцінити надійність, оперативність і точність роботи системи в цілому. У роботі пропонується використати три метрики:

– середня доступність IoT-пристрою;
– середній час оновлення пристрою;
– точність прогнозування стану IoT-пристрою.

Однією з ключових характеристик надійності IoT-інфраструктури є доступність, яка відображає частку часу, протягом якого пристрій або система

перебуває у працездатному стані та здатна виконувати свої функції. Середня доступність обчислюється за формулою:

$$A = \frac{1}{N}\sum_{i=1}^{N}\frac{t_{\text{роб}}(i)}{t_{\text{заг}}(i)} \quad (2.6)$$

де $N$ – кількість IoT-пристроїв у системі;

$t_{\text{роб}}(i)$ – сумарний час коректної роботи і-го пристрою протягом періоду спостереження;

$t_{\text{заг}}(i)$ – загальна тривалість періоду спостереження для і-го пристрою.

Даний показник набуває значень у межах [0;1]. Якщо значення $A$ ближче до 1 означає високу стабільність та безперервність роботи системи, а відповідно зменшення значення цього показника свідчить про часті збої, перезавантаження або тривалі простої пристроїв.

У контексті DevOps-підходу для IoT-систем показник доступності дозволяє оцінити вплив OTA-оновлень на стабільність пристроїв, визначити ефективність механізмів автоматичного відновлення та порівнювати різні конфігурації або версії прошивок.

Оперативність розгортання оновлень є критичною характеристикою для DevOps-орієнтованих IoT-систем, особливо в умовах великої кількості розподілених вузлів і обмежених мережевих ресурсів.

Середній час оновлення визначається як:

$$T_{upd} = \frac{1}{N}\sum_{i=1}^{N}(t_{\text{кін}}(i) - t_{\text{поч}}(i)), \quad (2.7)$$

де $t_{\text{поч}}(i)$ – момент ініціації оновлення програмного забезпечення для iii-го пристрою;

$t_{\text{кін}}(i)$ – момент завершення встановлення та підтвердження коректного запуску оновленої прошивки;

Показник середнього часу оновлення дозволяє оцінити ефективність OTA-механізмів доставки, вплив якості мережевого з'єднання на процес оновлення та масштабованість системи при одночасному оновленні великої кількості пристроїв.

Зменшення значення $T_{upd}$ свідчить про оптимальну організацію DevOps-процесу, наявність edge-кешування та ефективне використання мережевих ресурсів.

Для оцінювання ефективності модуля прогнозування стану IoT-пристроїв використовується показник точності класифікації, який відображає частку правильно ідентифікованих станів пристроїв.

Точність прогнозу обчислюється за формулою:

$$Acc = \frac{TP+TN}{TP+TN+FP+FN}, \quad (2.8)$$

де $TP$ – кількість випадків, коли деградація або аномальний стан пристроїв були правильно виявлені;

$TN$ – кількість випадків правильно визначеного нормального стану пристроїв;

$FP$ – хибні спрацювання системи;

$FN$ – пропущені випадки деградації.

Даний показник характеризує загальну якість роботи алгоритму прогнозування, однак у контексті експлуатації IoT-систем особливу увагу слід приділяти мінімізації значення $FN$, оскільки пропущені збої можуть призводити до відмов обладнання або втрати даних.

Точність прогнозування забезпечує можливість порівняння різних алгоритмів оцінювання стану, аналізувати ефективність інтеграції прогнозних моделей у DevOps-процес та визначати доцільність автоматичного запуску профілактичних оновлень.

Сукупний аналіз показників доступності, часу оновлення та точності прогнозування дозволяє отримати інтегральну оцінку ефективності

запропонованої системи. Високі значення доступності та точності при мінімальному часі оновлення свідчать про доцільність використання запропонованих методів і DevOps-засобів для обслуговування IoT-інфраструктури.

## 2.5. Висновки до розділу

1. Запропоновано та обґрунтовано багаторівневу архітектуру DevOps-орієнтованої IoT-системи, яка поєднує рівень IoT-пристроїв, edge-обчислень і хмарних сервісів, що дало змогу забезпечити безперервний збір телеметрії, масштабовану обробку даних, інтеграцію механізмів прогнозування стану та реалізацію безпечних OTA-оновлень у розподіленому середовищі.

2. Запропоновано математичну модель оцінювання технічного стану IoT-пристроїв, яка базується на векторі ключових параметрів (температура процесора, завантаженість CPU та пам’яті, якість зв’язку) та інтегральному показнику стану, що дало змогу формалізувати поняття працездатності IoT-пристроїв та забезпечити кількісну оцінку їх працездатності.

3. Розроблено метод визначення пріоритетів OTA-оновлення та прогнозування деградації, який поєднує поточний стан пристрою, час від останнього оновлення та кількість зафіксованих помилок, що дає змогу автоматизувати формування черги оновлень у DevOps-пайплайні та зменшити ризик відмов критичних вузлів.

4. Запропоновано систему метрик для оцінювання ефективності роботи платформи, що включає середню доступність, середній час оновлення та точність прогнозування стану. Комплексне використання цих показників дозволяє здійснювати об’єктивну оцінку надійності, оперативності та інтелектуальності запропонованої системи.

# РОЗДІЛ 3
# ПРОГРАМНА РЕАЛІЗАЦІЯ ТА НАЛАШТУВАННЯ IOT-ІНФРАСТРУКТУРИ ТА DEVOPS ЗАСОБІВ

## 3.1. Реалізація апаратної частини IoT-вузла

Апаратна частина IoT-вузла є базовим елементом усієї DevOps-орієнтованої системи моніторингу, оновлення та прогнозування технічного стану пристроїв. Від правильного вибору апаратної платформи, сенсорів і способів збору телеметрії залежить достовірність даних, стабільність роботи вузла, можливість віддаленого оновлення прошивки та ефективність подальшої аналітики. У межах даної роботи апаратну реалізацію IoT-вузла спроєктовано з урахуванням вимог до масштабованості, енергоефективності, безпеки та сумісності з DevOps-інфраструктурою.

У якості базової апаратної платформи для реалізації IoT-вузла обрано мікроконтролерну платформу ESP32, яка широко використовується у промислових та дослідницьких IoT-рішеннях. Такий вибір зумовлений поєднанням достатньої обчислювальної потужності, вбудованих засобів бездротового зв'язку та підтримки сучасних механізмів безпеки (рис. 3.1).

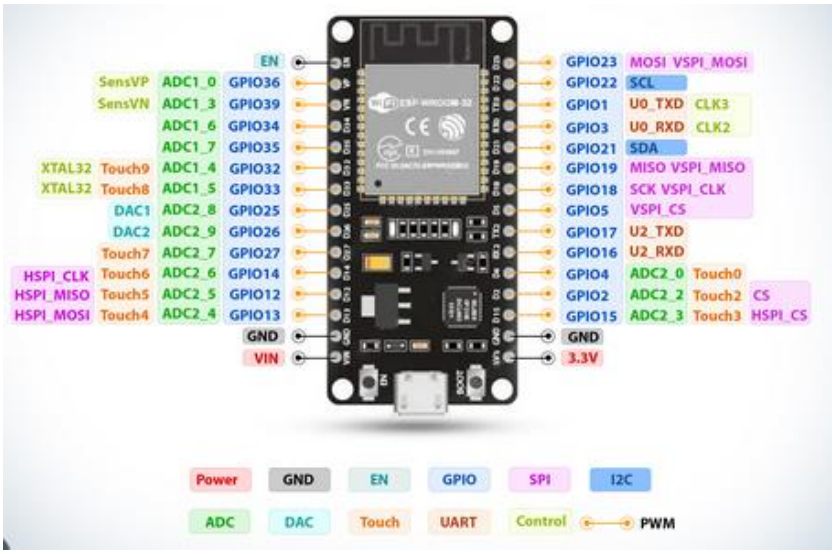


Рис. 3.1. Мікроконтролер ESP32

ESP32 є двоядерним мікроконтролером з архітектурою Xtensa або RISC-V (залежно від модифікації), який підтримує Wi-Fi та Bluetooth, має апаратні модулі шифрування, достатній обсяг оперативної пам’яті та флеш-пам’яті для зберігання прошивки й OTA-образів. Платформа підтримує апаратне розділення пам’яті для OTA-оновлень (OTA0/OTA1), що дозволяє реалізувати безпечне оновлення прошивки з можливістю автоматичного відкату у разі помилки.

Для порівняння, альтернативні платформи, такі як STM32, орієнтовані переважно на низькорівневі вбудовані системи та часто потребують зовнішніх мережевих модулів, що ускладнює реалізацію OTA-оновлень. Платформи класу Raspberry Pi мають значно вищу обчислювальну потужність, однак характеризуються підвищеним енергоспоживанням і зазвичай застосовуються на рівні edge-шлюзів, а не кінцевих IoT-вузлів. Таким чином, ESP32 є оптимальним компромісом між функціональністю, енергоспоживанням і вартістю.

Для оцінювання технічного стану IoT-вузла використовується сукупність апаратних і програмних джерел телеметрії. Основними параметрами, які реєструються та передаються у систему моніторингу, є температура мікроконтролера, параметри живлення, стан обчислювальних ресурсів та характеристики мережевого з’єднання.

Температура мікроконтролера є одним із ключових показників працездатності пристрою. Перегрів може призводити до зниження стабільності, або фізичної деградації компонентів. У ESP32 температура зчитується за допомогою вбудованих датчиків або непрямо оцінюється через внутрішні регістри, що дозволяє відслідковувати теплові режими у реальному часі.

Параметри живлення включають напругу живлення, рівень заряду акумулятора у разі автономного живлення та індикатори споживання енергії. Ці дані дозволяють виявляти проблеми з джерелами живлення, деградацію акумуляторів або перевантаження енергетичних ланцюгів.

Стан обчислювальних ресурсів визначається показниками завантаження процесора, використання оперативної пам’яті та кількістю помилок виконання. Для ESP32 ці параметри доступні через системні API та використовуються для

оцінювання ефективності роботи прошивки, а також для виявлення витоків пам’яті або нестабільних алгоритмів.

Якість мережевого з’єднання характеризується такими параметрами, як RSSI, затримки передачі повідомлень MQTT, кількість повторних підключень та втрати пакетів. Ці показники мають суттєвий вплив на процес OTA-оновлень і стабільність передачі телеметрії.

Збір телеметричних даних на IoT-вузлі організовано у вигляді періодичного опитування апаратних і програмних сенсорів з подальшою агрегацією результатів у структуровані повідомлення. Первинна обробка даних виконується безпосередньо на вузлі з метою зменшення обсягу переданої інформації та зниження навантаження на мережу.

До первинної обробки належать фільтрація аномальних значень, усереднення показників за часові вікна та формування компактних телеметричних пакетів. Для передачі даних використовується протокол MQTT, який добре пристосований до IoT-середовищ завдяки низьким накладним витратам і підтримці асинхронної передачі повідомлень.

Важливим аспектом є синхронізація збору телеметрії з процесами OTA-оновлення. Під час оновлення прошивки частота збору даних може зменшуватися, а після завершення – збільшуватися для контролю стабільності роботи. Такий підхід дозволяє оперативно виявляти негативні наслідки оновлення та ініціювати механізми rollback.

Однією з ключових особливостей IoT-вузлів є обмежені апаратні ресурси, зокрема обчислювальна потужність, обсяг пам’яті та доступна енергія. ESP32, як типова мікроконтролерна платформа, працює в умовах жорстких обмежень, що накладає вимоги на оптимізацію програмного забезпечення та режимів роботи.

Енергоспоживання IoT-вузла залежить від режимів роботи процесора, активності бездротового модуля та частоти збору телеметрії. Для зменшення споживання енергії застосовуються режими сну, динамічне керування частотою процесора та оптимізація мережевої активності. Такі заходи дозволяють суттєво продовжити час автономної роботи пристроїв.

Обмеження оперативної пам’яті вимагають раціонального використання буферів, мінімізації динамічного виділення пам’яті та контролю витоків. Це особливо важливо у контексті OTA-оновлень, де необхідно тимчасово зберігати образи прошивки та забезпечувати коректний перехід між версіями.

Таким чином, апаратна реалізація IoT-вузла у даній роботі побудована з урахуванням вимог до надійності, енергоефективності та сумісності з DevOps-орієнтованою інфраструктурою. Обрана платформа, сенсори та механізми збору телеметрії створюють основу для подальшої реалізації процесів моніторингу, прогнозування стану та автоматизованого оновлення програмного забезпечення, що детально розглядається у наступних підрозділах.

## 3.2. Програмна реалізація IoT-пристроїв та пристроїв edge-рівня

Програмне забезпечення IoT-вузла на базі ESP32 реалізовано у вигляді багатомодульної прошивки, яка складається з незалежних функціональних компонентів, що взаємодіють між собою через внутрішні черги повідомлень і спільні структури даних. Такий підхід дозволяє ізолювати логіку збору телеметрії від мережевої взаємодії та OTA-оновлень, що підвищує стабільність роботи пристрою.

Структурно прошивка містить такі основні модулі:

– модуль ініціалізації апаратних ресурсів;
– модуль збору телеметрії;
– модуль мережевої взаємодії (MQTT/HTTP);
– модуль OTA-оновлення;
– модуль самодіагностики та контролю помилок.

На рисунку 3.11 наведено структурну схему програмної прошивки IoT-вузла. Прошивка реалізована у вигляді набору взаємопов’язаних функціональних модулів, кожен з яких відповідає за окремий етап роботи пристрою.


Рис. 3.2. Структурна схема прошивки ESP32

Модуль ініціалізації апаратних ресурсів, який виконує початкове налаштування мікроконтролера, периферійних інтерфейсів, сенсорів та мережевих параметрів. Даний модуль забезпечує перевірку готовності апаратної частини до подальшої роботи та формує базову конфігурацію системи.

Наступним є модуль збору телеметрії, який відповідає за періодичне зчитування апаратних і програмних параметрів IoT-вузла, зокрема температури мікроконтролера, стану пам’яті, завантаження процесора та параметрів мережевого з’єднання. Зібрані дані передаються для подальшої обробки та передавання.

Збір телеметрії реалізовано у вигляді окремого FreeRTOS-завдання, яке з фіксованим інтервалом:

– зчитує технічні параметри IoT-вузла;
– формує структуру телеметрії;
– передає дані у модуль мережевої взаємодії (MQTT).

Структура телеметричних даних показана на рис. 3.3.

```c
typedef struct {
    float cpu_temperature;
    uint32_t free_heap;
    int8_t wifi_rssi;
    uint32_t uptime;
} telemetry_t;
```

Рис. 3.3. Структура телеметричних даних

Отримання значення температури мікроконтролера забезпечує програмний код, наведений на рис. 3.4 та вбудований сенсор ESP32.

```c
#include "driver/temperature_sensor.h"

static float get_cpu_temperature(void) {
    temperature_sensor_handle_t temp_handle;
    temperature_sensor_config_t temp_config =
        TEMPERATURE_SENSOR_CONFIG_DEFAULT(10, 80);

    temperature_sensor_install(&temp_config, &temp_handle);
    temperature_sensor_enable(temp_handle);

    float temperature = 0;
    temperature_sensor_get_celsius(temp_handle, &temperature);

    temperature_sensor_disable(temp_handle);
    temperature_sensor_uninstall(temp_handle);

    return temperature;
}
```

Рис. 3.4. Зчитування температури мікроконтролера

Для того, щоб отримати значення стану пам’яті мікроконтролера реалізовано програмний код, який показаний на рис. 3.5.

```c
#include "esp_system.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

static uint32_t get_free_heap(void) {
    return esp_get_free_heap_size();
}

static uint32_t get_uptime(void) {
    return xTaskGetTickCount() / configTICK_RATE_HZ;
}
```

Рис. 3.5. Отримання стану пам’яті мікрконтролера

Доступність та якість зв’язку реалізується шляхом отримання RSSI Wi-Fi як показано на рис. 3.6.

```c
#include "esp_wifi.h"

static int8_t get_wifi_rssi(void) {
    wifi_ap_record_t ap_info;
    if (esp_wifi_sta_get_ap_info(&ap_info) == ESP_OK) {
        return ap_info.rssi;
    }
    return -127; // сигнал відсутній
}
```

Рис. 3.6. Отримання RSSI Wi-Fi

Програмно FreeRTOS-завдання збору телеметрії реалізується так, як проілюстровано на рис. 3.7, а його запуск продемонстровано на рис. 3.8.

```c
static void telemetry_task(void *pvParameters) {
    telemetry_t telemetry;

    while (1) {
        telemetry.cpu_temperature = get_cpu_temperature();
        telemetry.free_heap = get_free_heap();
        telemetry.wifi_rssi = get_wifi_rssi();
        telemetry.uptime = get_uptime();

        mqtt_publish_telemetry(&telemetry);

        vTaskDelay(pdMS_TO_TICKS(5000));
    }
}
```

Рис. 3.7. Програмна реалізація FreeRTOS-завдання

```c
xTaskCreate(
    telemetry_task,
    "telemetry_task",
    4096,
    NULL,
    5,
    NULL
);
```

Рис. 3.7. Запуск FreeRTOS-завдання

Модуль мережевої взаємодії є одним із ключових компонентів програмної архітектури IoT-вузла, оскільки забезпечує обмін даними між пристроєм, edge-шлюзом та хмарною платформою. Основним завданням даного модуля є надійне та енергоефективне передавання телеметричних даних, а також підтримка керувальних і сервісних операцій, зокрема отримання конфігурацій та OTA-оновлень.

У запропонованій системі використовується гібридний підхід, який поєднує протоколи MQTT та HTTP, що дозволяє оптимально розподілити навантаження і забезпечити стабільну роботу в умовах нестабільних мереж.

Протокол MQTT обрано як основний механізм передавання телеметричних даних з IoT-вузлів, оскільки він має мінімальні накладні витрати, підтримує асинхронну модель обміну, добре працює у мережах з високими затримками і дозволяє масштабувати кількість пристроїв. Для уніфікації обміну даними використовується ієрархічна структура, яка показана на рис. 3.8.

```
iot/{device_id}/telemetry
iot/{device_id}/status
iot/{device_id}/logs
```

Рис. 3.8. Уніфіковане представлення даних протоколу MQTT

Програмно ініціалізація MQTT-клієнта виконується за допомогою програмного коду, представленого на рис. 3.9.

```
#include "mqtt_client.h"

static esp_mqtt_client_handle_t mqtt_client;

static void mqtt_init(void) {
    esp_mqtt_client_config_t mqtt_cfg = {
        .uri = "mqtt://edge-gateway.local",
        .client_id = "esp32_node_01",
        .keepalive = 60,
    };

    mqtt_client = esp_mqtt_client_init(&mqtt_cfg);
    esp_mqtt_client_start(mqtt_client);
}
```

Рис. 3.9. Ініціалізація MQTT-клієнта

Публікацію телеметрії через протокол MQTT забезпечує фрагмент коду, показаний на рис. 3.10.

При цьому дані передаються у форматі JSON, а QoS = 1 гарантує доставку повідомлення.

```c
static void mqtt_publish_telemetry(const telemetry_t *data) {
    char payload[256];

    snprintf(payload, sizeof(payload),
        "{ \"temp\": %.2f, \"heap\": %lu, \"rssi\": %d, \"uptime\": %lu }",
        data->cpu_temperature,
        data->free_heap,
        data->wifi_rssi,
        data->uptime
    );

    esp_mqtt_client_publish(
        mqtt_client,
        "iot/esp32_node_01/telemetry",
        payload,
        0,
        1,
        0
    );
}
```

Рис. 3.10. Публікація телеметрії через MQTT

Протокол HTTP застосовується для операцій, які потребують передачі більших обсягів даних та гарантованої доставки. Окрім цього, даний протокол забезпечує простоту інтеграції з веб-сервісами.

Основні сценарії використання HTTP стосуються:

– перевірка доступності OTA-оновлень,
– отримання конфігураційних параметрів
– передавання логів у випадку помилок.

Для підвищення надійності мережевої взаємодії реалізовано повторні спроби підключення при втраті мережі, буферизацію телеметрії у RAM при недоступності брокера і контроль тайм-аутів та збоїв з’єднання. Захист передавання даних забезпечується TLS 1.3 для HTTP-з’єднань, автентифікацією клієнтів MQTT і перевіркою сертифікатів сервера.

Для перевірки оновлень використовується запит, показаний на рис. 3.11.

```
#include "esp_http_client.h"

static void check_firmware_update(void) {
    esp_http_client_config_t config = {
        .url = "https://ota-server.local/check",
        .method = HTTP_METHOD_GET,
    };

    esp_http_client_handle_t client = esp_http_client_init(&config);
    esp_http_client_perform(client);
    esp_http_client_cleanup(client);
}
```

Рис. 3.11. Приклад HTTP-запиту

Для підвищення надійності мережевої взаємодії реалізовано повторні спроби підключення при втраті мережі, буферизацію телеметрії у RAM при недоступності брокера і контроль тайм-аутів та збоїв з'єднання. Захист передавання даних забезпечується TLS 1.3 для HTTP-з'єднань, автентифікацією клієнтів MQTT і перевіркою сертифікатів сервера.

Окремо виділено модуль OTA-оновлення, який відповідає за безпечне завантаження, перевірку та встановлення нових версій прошивки. Модуль підтримує механізми верифікації цілісності, підпису прошивки та автоматичного відкочування (rollback) у разі помилки оновлення.

OTA-оновлення реалізовано з використанням двосекційної схеми пам'яті (OTA partitioning), яка підтримується платформою ESP32. У флеш-пам'яті пристрою зберігаються дві незалежні області для прошивок (OTA_0 та OTA_1), що дозволяє безпечно завантажувати нову версію прошивки, не перериваючи виконання поточної.

Алгоритм OTA-оновлення включає такі етапи:

– перевірка доступності нової версії прошивки;

– завантаження OTA-образу;
– перевірка цілісності та підпису;
– запис прошивки у неактивний розділ пам’яті;
– перехід на нову версію після перезавантаження;
– верифікація коректності запуску;
– підтвердження або rollback.

OTA-оновлення може ініціюватися двома способами: отриманням керувальної команди через MQTT та періодичною перевіркою доступності оновлень через HTTP(S). Завантаження прошивки через HTTPS виконується за допомогою фрагменту коду, показаного на рис. 3.12.

```c
#include "esp_https_ota.h"

void ota_start(const char *firmware_url) {
    esp_http_client_config_t http_config = {
        .url = firmware_url,
        .timeout_ms = 5000,
    };

    esp_https_ota_config_t ota_config = {
        .http_config = &http_config,
    };

    if (esp_https_ota(&ota_config) == ESP_OK) {
        esp_restart();
    }
}
```

Рис. 3.12. Завантаження OTA-оновлення через HTTPS

Перед встановленням нової прошивки виконується перевірка контрольної суми та цифрового підпису (рис. 3.13.). Це дозволяє захистити систему від пошкоджених або несанкціонованих образів.

Після першого запуску нової прошивки система переходить у умовний режим підтвердження. Якщо протягом заданого інтервалу пристрій:

– коректно передає телеметрію;
– не перезавантажується;
– не фіксує критичних помилок,
– оновлення вважається успішним.

```
bool verify_firmware_crc(uint32_t expected_crc) {
    uint32_t actual_crc = calculate_crc();
    return actual_crc == expected_crc;
}
```

Рис. 3.13. Перевірка цілісності прошивки

OTA-модуль інтегрований у DevOps-конвеєр, у якому прошивка збирається у CI-середовищі та формується підписаний OTA-артефакт. Далі артефакт публікується у OTA-менеджері і вже після цього edge-шлюз або IoT-вузол отримує оновлення автоматично. Це дозволяє забезпечити безперервне оновлення прошивки без ручного втручання та з контролем результатів.

Модуль OTA-оновлення безпосередньо пов’язаний з оцінюванням технічного стану пристрою. Після кожного оновлення аналізуються параметри температури мікроконтролера, стабільності живлення, частоти перезавантажень і затримки передачі телеметрії. Це дозволяє виявляти негативні наслідки оновлень та використовувати ці дані у модулі прогнозування деградації.

Завершальним компонентом є модуль самодіагностики та контролю помилок, який здійснює постійний моніторинг стану програмних і апаратних ресурсів, фіксує збої та формує діагностичні повідомлення. Даний модуль забезпечує підвищення надійності роботи IoT-вузла та своєчасне виявлення деградації його стану.

Основні функції модуля самодіагностики полягають у виявленні деградації ресурсу та фіксації подій збоїв, зокрема: перезавантаження, WDT, помилки

мережі/OTA. На нього також покладено відповідальність за формування діагностичних подій та їх відправлення на edge/хмару. Окрім цього, він ініціює захисні дії щодо зменшення частоти телеметрії, перезапуску MQTT, OTA rollback (за потреби). Модуль самодіагностики складається з правил, які задають порогові значення та умови, а також він містить реєстратор подій та оцінювач стану працездатності пристроїв. Структура події програмно задається так, як показано на рис. 3.14.

```c
typedef enum {
    EVT_OK = 0,
    EVT_TEMP_HIGH,
    EVT_HEAP_LOW,
    EVT_WIFI_WEAK,
    EVT_MQTT_DISCONNECT,
    EVT_OTA_FAIL,
    EVT_REBOOT
} diag_event_type_t;

typedef struct {
    diag_event_type_t type;
    uint32_t ts;          // uptime або Unix time
    int32_t value;        // вимір/код помилки
} diag_event_t;
```

Рис. 3.14. Програмно визначена подія

Виявлення причин перезавантаження IoT-пристрою реалізує фрагмент програмного коду, наведений на рис. 3.15.

```c
#include "esp_system.h"
#include "esp_log.h"

static void report_reset_reason(void) {
    esp_reset_reason_t reason = esp_reset_reason();
    // Причина reset: POWERON_RESET, RTCWDT_RTC_RESET, TG0WDT_SYS_RESET тощо
    // Тут можна сформувати подію EVT_REBOOT + reason
}
```

Рис. 3.15. Виявлення причин перезавантаження пристрою

Приклад визначення правил самодіагностики проілюстровано на рис. 3.16.

```
#define TEMP_WARN_C      70.0f
#define HEAP_CRIT_BYTES  30000
#define RSSI_WARN_DBM    -80
```

Рис. 3.16. Визначення правил самодіагностики

Таким чином, наведені структурна схема та відповідні програмні реалізації реалізують модульну архітектуру прошивки IoT-пристрою, що забезпечує гнучкість, масштабованість та можливість інтеграції з DevOps-орієнтованою інфраструктурою моніторингу й оновлення.

## 3.3. Реалізація DevOps-процесу для безперервного оновлення IoT-вузлів

Класичний DevOps включає два ключові цикли: безперервна інтеграція (Continuous Integration) та безперервне оновлення та розгортання (Continuous Delivery / Deployment).

Перший цикл орієнтований на забезпечення безперервної інтеграції коду, формування збірок firmware та виконання перевірок. Другий цикл передбачає автоматичне розгортання оновлень, тестування і доставку на пристрої.

У випадку IoT-інфраструктури цей процес значно ускладнюється та розширюється. Це обумовлено великою кількістю вузлів із різними прошивками, нестабільним функціонуванням мережі та можливими втратами зв’язку. Окрім цього потрібно враховувати обмеження обчислювальних ресурсів, потребу у диференційних OTA-оновленнях. Також важливими факторами при організації DevOps-процесу є необхідність забезпечення rollback у разі невдалого оновлення та підвищені вимоги до кібербезпеки, зокрема, підпис firmware, TLS 1.3, перевірка CRC.

Тому DevOps-процес запропоновано формувати у вигляді розширеного OTA-конвеєру, що включає:

– розробку firmware;
– побудову CI-конвеєра;
– формування OTA-пакету;
– доставку firmware до OTA-менеджера;
– поширення через edge-шлюзи;
– перевірку успішності оновлення;
– збір телеметрії після оновлення;
– автоматичний rollback при помилці.

У системі моніторингу та прогнозування стану IoT-пристроїв, DevOps-процес запропоновано організувати на базі таких компонентів:

1. Репозиторій вихідного коду – GitHub (для CI/CD), приватні GitLab-Runner для компіляції firmware.

2. CI-середовище для збирання прошивки – GitHub Actions (для ESP32), GitLab CI для Linux-додатків edge-шлюзу у випадку Raspberry Pi.

3. Система контейнеризації – Docker, Docker Compose для edge-сервісів на шлюзі.

4. OTA-менеджер – використовується один із промислових інструментів Mender.io, оскільки він підтримує підпис оновлень, скасування оновлень, delta-оновлення та режими поступового розгортання.

5. Edge-шлюз – виконує буферизацію OTA-оновлень, перенаправлення firmware на вузли з нестабільною мережею, локальну перевірку CRC та локальний моніторинг.

6. IoT-вузли – ESP32 як основний клас вузлів, підтримка OTA через HTTPS або Mender Client Agent та передача телеметрії через MQTT.

7. Моніторинг та аналітика – Prometheus, Node-Exporter на edge-шлюзі, Grafana для створеня дашбордів, InfluxDB/TimescaleDB для часових рядів, модуль прогнозування на базі LSTM/RandomForest.

Алгоритм процесу безперервної інтеграції при зборі даних щодо прошивок для IoT-вузлів передбачає виконання семи кроків. Перший крок передбачає

завантаження програмного коду, який супроводжується виконанням операції push у гілку main при спрацюванні робочого процесу (workflow), показаного на рис. 3.17.

```yaml
on:
  push:
    branches: [ "main" ]
```

Рис. 3.17. Завантаження програмного коду для IoT-вузлів

Другий крок виконує компіляцію коду прошивки. Для ESP32 компіляція має вигляд, як показано на рис. 3.18.

```yaml
- name: Compile firmware
  run: idf.py build
```

Рис. 3.18. Компіляція прошивки для ESP32

Наступний крок передбачає запуск unit-тестів. При цьому тестуються модулі сенсорів та логіка передачі MQTT. Далі формуються OTA-артефакти, як показано на рис. 3.19.

```yaml
- name: Create OTA bundle
  run: python tools/create_ota_bundle.py
```

Рис. 3.19. Формування OTA-артефактів

На п’ятому кроці потрібно сформувати підпис прошивки за допомогою приватного ключа (рис. 3.20).

```
openssl dgst -sha256 -sign private.pem firmware.bin > firmware.sig
```

Рис. 3.20. Створення приватного підпису прошивки

На завершальному етапі процесу безперервної інтеграції відбувається завантаження прошивки в OTA-менеджер, як показано на рис. 3.21.

```
curl -X POST https://mender-server/api/v1/deployments \
  -F artifact=@firmware.mender \
  -H "Authorization: Bearer $TOKEN"
```

Рис. 3.21. Завантаження прошивки в OTA-менеджер

## 3.4. Організація доставки OTA-оновлення через edge-шлюз

Доставку OTA-оновлення запропоновано покласти на менеджер Mender. Спочатку даний менеджер приймає прошивку, зберігаючи артефакт у репозиторії оновлень. Edge-шлюз перевіряє доступність оновлень шляхом регулярного виконання команди, показаної на рис. 3.22.

```
mender check-update
```

Рис. 3.22. Перевірка оновлень edge-шлюзом

У результаті виконання команди щодо доступності оновлень формується список IoT-вузлів, які можуть отримати нові версії прошивок. Передача оновлень відбувається через мережу за допомогою протоколу HTTPS або MQTT-file-transfer у випадку нестабільності мережі.

Наступний крок передбачає встановлення оновлення на вузлі. Для прикладу ESP32, завантажує нову прошивку в область OTA1/OTA2 та виконує перехід, як показано на рис. 3.23.

```
esp_ota_set_boot_partition(ota_partition);
esp_restart();
```

Рис. 3.23. Встановлення прошивки на ESP32

Верифікація коректності відбувається одразу після першого завантаження. При цьому IoT-вузол надсилає повідомлення, як показано на рис. 3.24.

```
status: "update_ok"
firmware_version: "1.4.2"
checksum_crc32: ...
```

Рис. 3.24. Приклад повідомлення щодо верифікації прошивки

У випадку, якщо результат негативний, то виконується відкочування (rollback). Rollback відбувається автоматично у випадках невідповідності CRC, наявності системної помилки під час завантаження, відсутності телеметрії після оновлення або при перевищенні часу першого запуску. Mender підтримує passive failover, тобто якщо система не підтвердила оновлення протягом 20 хв, повертається попередня версія прошивки.

Для оцінювання стану системи використовується стек: Prometheus та Node Exporter. Вони забезпечують збір телеметрії edge-шлюзу та контроль процесора, пам’яті, мережі. Інший стек: MQTT, Telegraf та InfluxDB відповідають за надходження та моніторинг телеметрії, працездатності ESP32 та кінцевих вузлів, а також за обробку пакетів з пристроїв.

Для візуалізації результатів моніторингу призначена Grafana, яка формує три дашборди:

– “Status of devices” – відображення стану пристроїв;
– “Firmware rollout” – скасовані оновлення прошивок;

– “Failure & rollback statistics” – статистичні дані щодо успішності оновлення та збоїв.

Після встановлення оновлень, система аналізує часові ряди телеметрії, зокрема:

– температура мікроконтролера;
– частота перезавантажень пристроїв;
– RSSI;
– затримки MQTT;
– споживання енергії.

Для прогнозування стану пристроїв застосовуються такі алгоритми, як LSTM, RandomForest при класифікації стану IoT-вузлів “нормальний/деградація”, ARIMA використовується для прогнозування трендів температури та навантаження. Модель працює у хмарі, але edge може виконувати базову діагностику.

У результаті розроблено повноцінний DevOps-процес, що забезпечує весь життєвий цикл оновлення прошивки IoT-вузлів, починаючи від створення firmware до встановлення та контролю результатів.

Система підтримує OTA-оновлення, контрольні механізми безпеки, rollback-логіку, edge-буферизацію даних та централізований моніторинг.

Запропонована архітектура поєднує CI/CD, хмарні сервіси, edge-обчислення і машинне навчання, що забезпечує надійне та масштабоване оновлення IoT-інфраструктури.

## 3.5. Висновки до розділу

1. Запропоновано реалізацію апаратної частини IoT-вузла на базі мікроконтролера ESP32, що забезпечує оптимальне поєднання обчислювальної потужності, енергоефективності та підтримки сучасних мережевих і безпекових механізмів, що дало змогу сформувати надійне джерело телеметричних даних для подальшого моніторингу та оцінювання технічного стану пристроїв.

2. Спроєктовано програмну структуру прошивки IoT-пристрою, яка включає модулі ініціалізації, збору телеметрії, мережевої взаємодії, OTA-оновлення та самодіагностики, що дозволяє ізолювати функціональні компоненти, підвищити стабільність роботи вузлів і забезпечити можливість безпечного віддаленого оновлення програмного забезпечення.

3. Розроблено механізм збору та передавання телеметрії з використанням протоколів MQTT та HTTP, що забезпечило енергоефективний, масштабований та надійний обмін даними між IoT-вузлами, edge-шлюзом і хмарною платформою.

4. Запропоновано DevOps-орієнтований процес безперервного оновлення IoT-вузлів, який включає CI/CD-конвеєр, формування підписок OTA-артефактів, доставку оновлень через edge-шлюзи та автоматичні механізми їх скасування, що дало змогу зменшити ризики відмов та забезпечити контроль якості оновлень у масштабованій IoT-інфраструктурі.

5. Інтегровано модуль самодіагностики та контролю помилок на основі аналізу температури, стану пам’яті, мережевих параметрів і подій збоїв, що дає можливість оперативно реагувати на деградацію пристрою та формувати вхідну інформацію для модулів прогнозування технічного стану.

# РОЗДІЛ 4
# ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

## 4.1. Охорона праці

Метою кваліфікаційної роботи магістра є дослідження та розроблення методів і DevOps-засобів моніторингу, оновлення та прогнозування стану IoT-пристроїв. Реалізація поставлених у роботі завдань передбачає активне використання комп'ютерної техніки, серверного обладнання, одноплатних комп'ютерів, IoT-вузлів та мережевих пристроїв, що зумовлює необхідність дотримання вимог охорони праці та техніки безпеки під час роботи з електронно-обчислювальними засобами.

Для забезпечення ефективної та безпечної діяльності фахівців, які займаються розробленням, тестуванням і експлуатацією DevOps-орієнтованих IoT-систем, необхідно організувати належні умови праці. При цьому відповідальність за дотримання нормативно-правових актів з охорони праці покладається як на інженерів і розробників, так і на керівників відповідних підрозділів [18].

Робочі місця фахівців, що здійснюють моніторинг і керування IoT-пристроями, повинні відповідати вимогам НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». Дотримання зазначених вимог є особливо важливим з огляду на тривалу роботу з моніторами, засобами налагодження програмного забезпечення, середовищами керування DevOps-конвеєрами та аналітичними панелями моніторингу.

Приміщення, у яких розміщуються робочі місця розробників і адміністраторів IoT-систем, за винятком серверних приміщень, повинні бути обладнані системами автоматичної пожежної сигналізації відповідно до вимог чинних державних будівельних норм і переліку об'єктів, що підлягають обладнанню автоматичними установками пожежогасіння [21].

Робочі приміщення мають бути оснащені первинними засобами пожежогасіння (вогнегасниками), кількість і тип яких визначаються відповідно до

Типових норм належності вогнегасників, затверджених наказами уповноважених органів. Проходи до засобів пожежогасіння повинні бути вільними та не захараценими обладнанням.

Приміщення, у яких експлуатується серверне обладнання, edge-шлюзи та мережеві компоненти IoT-інфраструктури, повинні бути оснащені системами автоматичної пожежної сигналізації та засобами пожежогасіння відповідно до вимог ДБН В.1.1-7-2016, ДСТУ Б.В.1.1-36:2016, НАПБ А.01.001-2014, а також технічної документації виробників обладнання.

Електроживлення комп’ютерної та мережевої техніки має здійснюватися через окрему групову трипровідну електромережу з прокладанням фазового, нульового робочого та нульового захисного провідників. Нульовий захисний провідник використовується виключно для заземлення електроприймачів і не може використовуватися як нульовий робочий провідник. Забороняється підключення нульового робочого та нульового захисного провідників до одного контактного затискача [19].

Площа поперечного перерізу нульового робочого та нульового захисного провідників повинна бути не меншою за площу перерізу фазового провідника та відповідати вимогам ДСТУ Б В.2.5-82:2016 з урахуванням номінального навантаження, температурного режиму та умов експлуатації [20].

У приміщеннях, де одночасно експлуатується понад п’ять комп’ютерів або серверних пристроїв, має бути встановлений аварійний резервний вимикач, що забезпечує повне відключення електроживлення, за винятком систем освітлення.

Комп’ютерна техніка, сервери, edge-шлюзи та периферійні пристрої повинні підключатися до електромережі лише через справні штепсельні з’єднання та розетки заводського виготовлення, які мають спеціальні контакти для підключення нульового захисного провідника. Конструкція таких з’єднань повинна забезпечувати першочергове підключення заземлення.

Не допускається експлуатація комп’ютерної техніки та IoT-обладнання з використанням двопровідних мереж або не сертифікованих перехідних пристроїв. Штепсельні з’єднання для різних рівнів напруги повинні мати чіткі конструктивні

та візуальні відмінності.

Важливим чинником охорони праці є забезпечення належного природного та штучного освітлення робочих місць відповідно до вимог ДБН В.2.5-28:2018 «Природне і штучне освітлення». Робочі місця повинні бути організовані з урахуванням ергономічних вимог, що забезпечують комфортну та безпечну роботу з комп'ютерною технікою.

Відстань від екрана монітора до очей оператора повинна відповідати вимогам НПАОП 0.00-7.15-18, а розміщення допоміжних пристроїв, зокрема принтерів і мережевого обладнання, має забезпечувати зручність доступу та мінімізацію фізичного навантаження.

Таким чином, у результаті аналізу вимог з охорони праці та електробезпеки визначено основні умови організації безпечних робочих місць для фахівців, які займаються розробленням і експлуатацією DevOps-орієнтованих систем моніторингу, оновлення та прогнозування стану IoT-пристроїв. Дотримання зазначених вимог забезпечує ефективну, надійну та безпечну професійну діяльність.

## 4.2. Забезпечення безпеки життєдіяльності населення в умовах надзвичайних ситуацій природного походження

Надзвичайні ситуації природного походження становлять одну з найбільш небезпечних загроз для життя і здоров'я населення, об'єктів інфраструктури та навколишнього природного середовища [22]. До таких ситуацій належать землетруси, повені, зсуви, селі, урагани, буревії, сильні снігопади, посухи, лісові пожежі, епідемії та інші природні явища, які можуть мати масштабні соціальні, економічні та екологічні наслідки. Забезпечення безпеки життєдіяльності населення в умовах таких ситуацій є важливою державною та суспільною задачею, що потребує системного підходу.

Безпека життєдіяльності в умовах надзвичайних ситуацій природного походження передбачає комплекс організаційних, інженерно-технічних, санітарно-

гігієнічних, інформаційних і правових заходів, спрямованих на запобігання виникненню загроз або мінімізацію їх наслідків. Основною метою цих заходів є збереження життя і здоров'я людей, забезпечення стійкого функціонування об'єктів життєзабезпечення та зменшення матеріальних втрат.

Одним із ключових елементів системи безпеки є попередження надзвичайних ситуацій. Воно базується на моніторингу природних процесів, прогнозуванні можливих небезпечних явищ та оцінюванні ризиків. Використання метеорологічних, гідрологічних, сейсмічних і геоінформаційних систем дозволяє завчасно виявляти потенційні загрози та інформувати населення і відповідні служби. Своєчасне прогнозування сприяє прийняттю превентивних рішень, таких як евакуація, укріплення захисних споруд або обмеження доступу до небезпечних зон [22].

Важливе значення має підготовка населення до дій у надзвичайних ситуаціях. Вона включає інформування громадян про можливі природні загрози, правила поведінки у разі їх виникнення, а також навчання навичкам надання першої медичної допомоги. Рівень обізнаності населення значною мірою визначає масштаби людських втрат, оскільки правильні та своєчасні дії дозволяють зменшити ризик травмування або загибелі.

У разі виникнення надзвичайної ситуації природного походження ключову роль відіграє система оповіщення та інформування населення. Вона повинна забезпечувати швидке, достовірне та зрозуміле донесення інформації про характер загрози, її можливі наслідки та необхідні дії. Оповіщення здійснюється з використанням різних каналів зв'язку, включаючи сирени, засоби масової інформації, мобільні повідомлення та інтернет-ресурси. Надійність і доступність цієї системи є критичним фактором ефективного реагування [22].

Забезпечення безпеки життєдіяльності також передбачає організацію захисних заходів, серед яких важливе місце займає евакуація населення з небезпечних територій. Евакуаційні заходи повинні бути заздалегідь сплановані, включати визначення маршрутів, місць тимчасового розміщення та забезпечення людей необхідними ресурсами. Особлива увага приділяється вразливим категоріям

населення, зокрема дітям, людям похилого віку та особам з обмеженими можливостями.

Суттєвим аспектом безпеки є інженерно-технічний захист територій і об’єктів інфраструктури. Він включає будівництво дамб, протизсувних споруд, укріплення берегів, створення протипожежних бар’єрів, а також дотримання будівельних норм і правил у зонах підвищеного ризику. Надійність таких заходів значною мірою визначає масштаби руйнувань у разі природних катастроф [24].

Не менш важливим є медичне та санітарно-епідеміологічне забезпечення населення. Після природних катастроф часто виникає загроза поширення інфекційних захворювань, нестачі чистої води та продовольства. Тому система охорони здоров’я повинна бути готовою до оперативного надання медичної допомоги, організації протиепідемічних заходів та психологічної підтримки постраждалих.

Окрему роль у забезпеченні безпеки життєдіяльності відіграє координація дій органів державної влади, місцевого самоврядування та аварійно-рятувальних служб. Чітке розмежування повноважень, наявність планів реагування та злагоджена взаємодія дозволяють ефективно управляти процесами ліквідації наслідків надзвичайних ситуацій. Важливим елементом є також міжнародне співробітництво у сфері запобігання та реагування на природні катастрофи [22].

Таким чином, забезпечення безпеки життєдіяльності населення в умовах надзвичайних ситуацій природного походження є комплексною задачею, що потребує поєднання превентивних, організаційних, технічних та соціальних заходів. Ефективна реалізація цих заходів дозволяє зменшити ризики для життя і здоров’я людей, підвищити стійкість суспільства до природних загроз та забезпечити швидке відновлення нормальних умов життєдіяльності після надзвичайних ситуацій.

## 4.3. Методи захисту від дії ЕМІ, що базуються на врахуванні його можливого негативного впливу

ЕМІ здатний викликати потужні імпульси струмів і напруг у проводах і кабелях повітряних і підземних ліній зв'язку, сигналізацій, управління, електропередачі, в антенах радіостанцій [23].

Вплив ЕМІ може призвести до згорання чутливих електронних і електричних елементів, пов'язаних з великими антенами чи відкритими проводами, а також до серйозних порушень в цифрових і контрольних пристроях, зазвичай без необоротних змін.

Для найбільш важливих пристроїв треба застосовувати заходи захисту та підвищувати їх стійкість до ЕМІ. Ступінь ушкодження залежить в основному від амплітуди наведеного імпульсу напруги чи струму і електричної міцності обладнання. Особливо схильна до впливу ЕМІ радіоелектронна апаратура, виконана на напівпровідникових та інтегральних системах, працюючих на малих струмах і напругах і, чутливих до впливу зовнішніх електричних і магнітних полів.

ЕМІ пробиває ізоляцію, випалює елементи електросхем радіоапаратури, викликає коротке замикання в радіопристроях, іонізацію діелектриків, спотворює або повністю стирає магнітний запис, позбавляє пам'яті ЕОМ [23].

Інженерно-технічні заходи мають забезпечити підвищену стійкість виробничих споруд, технологічних ліній, устаткування, комунікацій об'єкта до впливу уражаючих факторів під час надзвичайних ситуацій.

При проведенні цих заходів необхідно враховувати конкретні умови підприємства. Проте є загальні організаційні інженерно-технічні заходи, які мають проводитись на всіх об'єктах.

Захистити цінне і унікальне устаткування можна завдяки проведенню інженерно-технічних заходів, щоб зменшити небезпеку пошкодження і руйнування цінного й унікального устаткування, комп’ютерної техніки, станків з програмним керуванням, шліфувальних, токарних, розточних, зубофрезерних, пресових станків, автоматичних конвеєрних ліній та іншого устаткування. Варіантами такого

захисту є розміщення зазначеного устаткування в заглиблених приміщеннях, а також використання спеціальних захисних пристосувань, закріплення станків на фундаментах, застосування контрфорсів для підвищення стійкості проти перекидання обладнання [23].

Створення резерву енергетичних потужностей за рахунок автономних пересувних електростанцій, а також місцевих джерел електроенергії. Підготовка автономних електростанцій до роботи за спеціальним режимом (графіком) для забезпечення технологічних процесів виробництва, для яких неможливі тривалі перерви в електропостачанні. З метою попередження аварій на електричних мережах необхідно установити автоматичну систему відключення при виникненні перенапруги. Повітряні лінії електропостачання замінити на підземно-кабельні. Створення необхідних запасів (резервів) паливно-мастильних матеріалів та інших видів палива й організація їх безпечного зберігання. Щоб не допустити зупинки підприємства через дефіцит палива, необхідно підготуватись для роботи на різних видах палива: нафта, вугілля, газ [23].

В загальному, методи і засоби захисту від електромагнітних полів можна умовно розділити на інженерно-технічні, організаційні та лікувально-профілактичні. Згідно зі встановленою процедурою, захист людини від такого небезпечного впливу повинен здійснюється такими способами:

– зменшення випромінювання від джерела;
– екранування джерела випромінювання та робочого місця;
– встановлення санітарно-захисної зони;

Залежно від джерел випромінювання та того, де саме вони знаходяться – ззовні чи в середині приміщення, можна оптимально підібрати захист від ЕМП. Якщо джерело забруднення електромагнітним випромінюванням знаходиться ззовні приміщення (трансформатора підстанція чи електрощитова, вишка стільникового зв’язку, тощо), надійним засобом захисту від електромагнітного випромінювання у цьому випадку є спеціально розроблена для захисту від ЕМП екрануюча сітка HEG10 фірми «Gigahertz Solutions». Цю сітку кріплять до стіни за допомогою спеціального клею, що також має екрануючі властивості. Сітка HEG10

забезпечує захист від ЕМП високих частот та від малих електричних полів, сітка потребує заземлення. Якщо ж джерело ЕМП знаходиться в середині приміщення, оптимальним засобом захисту від електромагнітного випромінювання є екрануюча фарба. Надійний захист від електромагнітних полів забезпечує фарба для екранування ЕМП виробництва «Gigahertz Solutions». Вона не токсична, відтак, придатна до використання як в середині приміщення, так і ззовні [24].

Окрім функції захисту від електромагнітного випромінювання, ці засоби можна також використовувати і для захисту даних з комп'ютерних мереж та комп'ютерів. Окрім того, захист офісу від ЕМП ззовні дасть можливість заблокувати зовнішні мережі WIFI, та захистити від проникнення ззовні власну мережу WIFI, що актуально для офісів, розміщених у великих офісних центрах.

У результаті проведеного аналізу щодо застосування основних способів та засобів в ході проведення невідкладних аварійно-рятувальних робіт на промисловому підприємстві встановлено послідовність дій, які необхідно виконати для збереження життя і здоров'я працівників підприємства, а також необхідні для цього матеріально-технічні засоби. Також проведено аналіз дії електромагнітного імпульсу при ядерному вибухові на людину та господарську діяльність і визначено потенційні шляхи мінімізації його негативного впливу та збереження життєдіяльності підприємств.

# ВИСНОВКИ

Основні наукові та практичні результати полягають в наступному.

1. Проведено аналітичний огляд сучасних підходів до моніторингу, обслуговування та прогнозування стану IoT-пристроїв, що дозволило встановити обмеження традиційних методів та обґрунтувати доцільність переходу до прогнозного обслуговування на основі аналізу телеметричних даних.

2. Проаналізовано архітектури побудови IoT-систем із використанням edge-, fog- та cloud-рівнів, а також моделі потокової обробки даних, що дало змогу визначити ключові вимоги до масштабованості, затримок передачі, обробки великих обсягів даних і безперервного моніторингу технічного стану пристроїв.

3. Досліджено особливості застосування DevOps-підходів у середовищах Інтернету речей, включаючи CI/CD-процеси, OTA-оновлення та зворотну телеметрію, що дозволило обґрунтувати необхідність автоматизації життєвого циклу IoT-пристроїв з урахуванням обмежених ресурсів і вимог до безпеки.

4. Проаналізовано хмарні моделі надання сервісів IaaS, PaaS та SaaS у контексті DevOps-орієнтованих IoT-систем, що дало змогу визначити їх переваги та обмеження при організації обробки телеметрії, моніторингу, аналітики та оновлення програмного забезпечення.

5. Встановлено, що комплексне поєднання хмарних сервісів, DevOps-інструментів і методів аналізу даних є найбільш перспективним підходом, який забезпечує автоматизований моніторинг, прогнозування технічного стану та безпечне оновлення IoT-пристроїв.

6. Запропоновано та обґрунтовано багаторівневу архітектуру DevOps-орієнтованої IoT-системи, яка поєднує рівень IoT-пристроїв, edge-обчислень і хмарних сервісів, що дало змогу забезпечити безперервний збір телеметрії, масштабовану обробку даних, інтеграцію механізмів прогнозування стану та реалізацію безпечних OTA-оновлень у розподіленому середовищі.

7. Запропоновано математичну модель оцінювання технічного стану IoT-пристроїв, яка базується на векторі ключових параметрів (температура процесора,

завантаженість CPU та пам’яті, якість зв’язку) та інтегральному показнику стану, що дало змогу формалізувати поняття працездатності IoT-пристроїв та забезпечити кількісну оцінку їх працездатності.

8. Розроблено метод визначення пріоритетів OTA-оновлення та прогнозування деградації, який поєднує поточний стан пристрою, час від останнього оновлення та кількість зафіксованих помилок, дає змогу автоматизувати формування черги оновлень у DevOps-пайплайні та зменшити ризик відмов критичних вузлів.

9. Запропоновано систему метрик для оцінювання ефективності роботи платформи, що включає середню доступність, середній час оновлення та точність прогнозування стану. Комплексне використання цих показників дозволяє здійснювати об’єктивну оцінку надійності, оперативності та інтелектуальності запропонованої системи.

10. Запропоновано реалізацію апаратної частини IoT-вузла на базі мікроконтролера ESP32, що забезпечує оптимальне поєднання обчислювальної потужності, енергоефективності та підтримки сучасних мережевих і безпекових механізмів, що дало змогу сформувати надійне джерело телеметричних даних для подальшого моніторингу та оцінювання технічного стану пристроїв.

11. Розроблено механізм збору та передавання телеметрії з використанням протоколів MQTT та HTTP, що забезпечило енергоефективний, масштабований та надійний обмін даними між IoT-вузлами, edge-шлюзом і хмарною платформою.

12. Запропоновано DevOps-орієнтований процес безперервного оновлення IoT-вузлів, який включає CI/CD-конвеєр, формування підписок OTA-артефактів, доставку оновлень через edge-шлюзи та автоматичні механізми їх скасування, щодало змогу зменшити ризики відмов та забезпечити контроль якості оновлень у масштабованій IoT-інфраструктурі.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Луцків А. М., Лупенко С. А., Пасічник В. В. Паралельні та розподілені обчислення : навч. посіб. / А. М. Луцків, С. А. Лупенко, В. В. Пасічник. – Львів : ПП «Магнолія 2006», 2024. 565 с.

2. Паламар М.І., Стрембіцький М.О., Паламар А.М. Проектування комп'ютеризованих вимірювальних систем і комплексів. Навчальний посібник. Тернопіль: ТНТУ. 2019. 150 с.

3. Оконський М.В., Лупенко С.А., Паламар А.М. «Комп'ютерна система для моніторингу метеорологічних параметрів на основі IoT». Матеріали X Міжнар. наук.-техн. конференції «Актуальні задачі сучасних технологій». Тернопіль : ТНТУ, 2021. С. 112.

4. Elkateb F., Khan A., Ahmed M.N., et al. Machine learning and IoT – Based predictive maintenance: A comprehensive study. Computers & Electrical Engineering. 2024. Vol. 105, p.108512. DOI: 10.1016/j.compeleceng.2024.108512. URL: https://www.sciencedirect.com/science/article/pii/S1110016823011572 (дата звернення: 18.11.2025 р.).

5. What is a Kubernetes cluster? URL: https://www.vmware.com/topics/glossary/content/kubernetes-cluster.html (дата звернення 21.11.2025 р).

6. What is Kubernetes infrastructure? URL: https://www.vmware.com/topics/glossary/content/kubernetes-infrastructure.html (дата звернення 23.11.2025 р).

7. Kubernetes Clusters: Everything You Need To Know. URL: https://www.containiq.com/post/kubernetes-cluster (дата звернення 23.11.2025 р).

8. Vault Documentation. URL: https://developer.hashicorp.com/vault/docs?host=www.vaultproject.io (дата звернення 24.11.2025 р).

9. How Ansible works. URL: https://www.ansible.com/overview/how-ansible-works (дата звернення 25.11.2025 р).

10. Red Hat Ansible Automation Platform. URL: https://www.redhat.com/en/technologies/management/ansible (дата звернення 25.11.2025 р).

11. OpenStack Services. URL: https://www.openstack.org/software/project-navigator/openstack-components#openstack-services (дата звернення 06.12.2025).

12. Sun Y., et al. «UAV and IoT-Based Systems for the Monitoring of Industrial Facilities and their Adjacent Territories». Sensors. 2022. Vol. 22(17), Article 6444. DOI:10.3390/s22176444. URL: https://www.mdpi.com/1424-8220/22/17/6444 (дата звернення: 09.12.2025).

13. Caldana V. M., Garrido da Silva F. D. «Internet of Things and Artificial Intelligence applied to predictive maintenance in Industry 4.0: A systematic literature review». Proceedings of the International Conference on Industrial Engineering and Operations Management (IEOM) 2021. URL: https://www.ieomsociety.org/brazil2020/papers/582.pdf (дата звернення: 10.12.2025).

14. Луцик Н. С., Луцків А. М., Осухівська Г. М., Тиш Є. В. Методичні рекомендації до виконання кваліфікаційної роботи магістра для студентів спеціальності 123 «Комп'ютерна інженерія» другого (магістерського) рівня вищої освіти усіх форм навчання. Тернопіль. ТНТУ. 2024. 44 с.

15. Луцків А.М., Комарницький В.В. DevOps-підхід до автоматизації CI/CD у розподілених IoT-системах. Матеріали XIV міжнародної науково - технічної конференції молодих учених і студентів «Актуальні задачі сучасних технологій» (11-12 грудня 2025 р.) Тернопільського національного технічного університету імені Івана Пулюя. Тернопіль: ТНТУ. 2025. С. 296.

16. Луцків А.М., Комарницький В.В. Проєктування системи моніторингу та віддаленого оновлення IoT-пристроїв. Матеріали XIII науково-технічної конференції Тернопільського національного технічного університету імені Івана Пулюя «Інформаційні моделі, системи та технології» (17-18 грудня 2025 року). Тернопіль: ТНТУ. 2025. С. 131.

17. Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks. Vol. 61. N. 1. January 2015. pp. 85–117.

18. НПАОП 0.00-7.15-18 «Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями». Київ. 2018.

19. ДБН В.2.5-28-2018 «Природне і штучне освітлення». Київ : Мінрегіон України. 2018.

20. ДБН В.1.1-7-2016 «Пожежна безпека об’єктів будівництва». Київ : Мінрегіон України. 2016.

21. Бедрій Я. Основи охорони праці користувачів персональних комп’ютерів: навчальний посібник для студентів ВНЗ та інженерів-практиків. Навчальна книга-Богдан. 2014. 144 с.

22. Стручок В.С. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «Безпека в надзвичайних ситуаціях». Тернопіль: ФОП Паляниця В. А.156 с.

23. Стручок В.С. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА. ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА»» / Тернопіль: ФОП Паляниця В. А. 156 с.

24. Желібо Е.Н. Безпека життєдіяльності: Навчальний посібник/ За редакцією Е.П. Желібо, В.М. Львів: «Новий світ - 2000», 2011. 320с.

Додаток А

Тези конференцій

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**Тернопільський національний технічний університет імені Івана Пулюя**
**Маріборский університет (Словенія)**
**Технічний університет у Кошице (Словаччина)**
**Вільнюський технічний університет ім. Гедимінаса (Литва)**
**Краківський економічний університет (Польща)**
**Вроцлавський економічний університет (Польща)**
**Університет «Опольська Політехніка» (Польща)**
**Національний університет «Полтавська політехніка імені Юрія Кондратюка»**
**Вінницький національний аграрний університет**
**Львівський національний університет ім. І. Франка**
**Головне управління Пенсійного фонду в Тернопільській області**
**Наукове товариство ім. Шевченка**
**Тернопільський обласний комунальний інститут післядипломної педагогічної освіти**
**Сумський державний педагогічний університет**
**Запорізький національний університет**

# АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

## Збірник

тез доповідей

**XIV Міжнародної науково-технічної конференції молодих учених та студентів**

11-12 грудня 2025 року

**УКРАЇНА**
**ТЕРНОПІЛЬ – 2025**

УДК 004.75
**А.М. Луцків канд. техн. наук, доцент, В.В. Комарницький**
Тернопільський національний технічний університет імені Івана Пулюя

## DEVOPS-ПІДХІД ДО АВТОМАТИЗАЦІЇ CI/CD У РОЗПОДІЛЕНИХ IOT-СИСТЕМАХ

**A.M. Lutskiv PhD., Assoc. Prof., V.V. Komarnytskyi**
## DEVOPS APPROACH TO CI/CD AUTOMATION IN DISTRIBUTED IOT SYSTEMS

У сучасній комп'ютерній інженерії DevOps виступає основним підходом до інтеграції розроблення, тестування, розгортання та підтримки програмних систем. Його головна мета полягає у забезпеченні безперервного життєвий циклу програмного продукту з високим рівнем автоматизації та контролю якості. Для IoT-систем, які поєднують тисячі або навіть мільйони розподілених пристроїв, DevOps набуває особливого значення. Тут необхідно не лише підтримувати актуальність програмного забезпечення на різних вузлах, але й гарантувати стабільність роботи та безпеку оновлень у реальному часі. Класичний DevOps включає низку етапів, які утворюють циклічний процес CI/CD. Структуру цього процесу з використанням сучасних підходів проілюстровано на рис. 1.


Рис. 1. Узагальнена схема DevOps-процесу для IoT-систем на основі сучасних підходів

Основними складовими DevOps-процесу є:

– Continuous Integration (CI) – безперервне об'єднання коду, перевірка сумісності нових змін і автоматичне тестування;

– Continuous Deployment (CD) – автоматизоване розгортання оновлень у робочому середовищі;

– Monitoring – постійне відстеження стану системи та пристроїв після оновлення;

– Feedback – зворотний зв'язок, який дозволяє швидко реагувати на помилки й удосконалювати систему.

Для традиційних веб- або серверних застосунків CI/CD-процес реалізується у хмарі або у локальному дата-центрі. В IoT-середовищі ситуація ускладнюється тим, що розгортання відбувається не на одному сервері, а на великій кількості фізичних пристроїв, часто з обмеженими ресурсами, низькою пропускною здатністю каналів зв'язку та нестабільною мережею.

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**
**ІМЕНІ ІВАНА ПУЛЮЯ**

**М А Т Е Р І А Л И**

**XIII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ**

# «ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

**17-18 грудня 2025 року**

**ТЕРНОПІЛЬ**
**2025**

**УДК 004.75**

**А. Луцків к. т. н, доц.; В. Комарницький**
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

## ПРОЄКТУВАННЯ СИСТЕМИ МОНІТОРИНГУ ТА ВІДДАЛЕНОГО ОНОВЛЕННЯ IOT-ПРИСТРОЇВ

**A. Lutskiv PhD., Assoc. Prof.; V. Komarnytskyi**

## DESIGN OF A MONITORING AND REMOTE UPDATE SYSTEM FOR IOT DEVICES

Проєктування системи моніторингу, прогнозування та віддаленого оновлення стану IoT-пристроїв потребує формування цілісної архітектури, що поєднує апаратну, програмну та мережеву складові. Вона повинна забезпечувати безперервний збір телеметрії, обробку даних у режимі реального часу, можливість виконання OTA-оновлень та зручне масштабування. Окрім цього, система моніторингу повинна забезпечувати такі критерії якості як висока надійність і відмовостійкість, а також безпека комунікацій.

Будь-яка система моніторингу стану IoT-пристроїв у DevOps-парадигмі складається щонайменше з трьох функціональних рівнів. Перший рівень – це рівень пристроїв (Device Layer), який включає сенсори, мікроконтролери, мікрокомп'ютери та виконавчі механізми. На цьому рівні формуються пакети телеметричних даних та виконується початкова обробка.

Наступний рівень – периферійний рівень. Він реалізує агрегацію даних, буферизацію, попередню аналітику, розподілення навантаження та обробку подій, що важливо для зниження трафіку та затримок.

Третій рівень пов'язаний з хмарними сервісами. Він забезпечує зберігання великих обсягів даних, містить набори інструментів для проведення глибокої аналітики, підтримує побудову моделей прогнозування, DevOps-процеси, керування оновленнями та візуалізацію. У роботі пропонується архітектура платформи моніторингу та прогнозування стану IoT пристроїв з використанням DevOps практик та інструментів, яка представлена на рис. 1.


**Рисунок 1.** Узагальнена багаторівнева архітектура DevOps-орієнтованої IoT-платформи

Пристроями, які безпосередньо виконують вимірювання показників певних об'єктів чи параметрів стану середовища є IoT-пристрої.