КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (34), 2026
ISSN 2663 – 4023
DOI 10.28925/2663-4023.2026.34.1328
УДК 004.627:681.5.07

Ковалюк Дмитро Олександрович
кандидат технічних наук, доцент
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
ORCID:0000-0002-9729-1443
dmytro.kovalyuk@gmail.com

Счастлівцев Денис Сергійович
аспірант
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
ORCID:0009-0001-9699-100X
Denis.schastlivtsev@gmail.com

Кріт Андрій Ігорович
аспірант
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
ORCID:0009-0003-3028-0998
andrey.krit@gmail.com

Киричук Тарас Михайлович
аспірант
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Київ, Україна
ORCID:0009-0007-3761-0960
taraskirivuk@gmail.com

МІКРОСЕРВІСНА АРХІТЕКТУРА ХМАРНОЇ ПЛАТФОРМИ НА БАЗІ MICROSOFT AZURE ДЛЯ ІНТЕЛЕКТУАЛЬНОГО КЕРУВАННЯ КОМП'ЮТЕРНО-ІНТЕГРОВАНИМИ РОБОТИЗОВАНИМИ СИСТЕМАМИ

Анотація. У статті розроблено та досліджено мікросервісну архітектуру хмарної платформи на базі Microsoft Azure, призначену для управління комп'ютерно-інтегрованими роботизованими системами (КІРС). Розглянуто труднощі масштабування виробничих інформаційних систем під час обробки великих обсягів даних з рівня цеху, а доцільність застосування мікросервісної архітектури й хмарних технологій обґрунтовано як засіб забезпечення гнучкості та надійності виробничих процесів. Сучасні дослідження у сфері хмарних і периферійних обчислень для промисловості проаналізовано так, що раніше невирішеною частиною загальної проблеми виявлено відсутність комплексних платформ управління КІРС із гарантіями якості обслуговування, надійності та безпеки. У роботі побудовано розширену математичну модель, що охоплює аналіз затримок на основі теорії масового обслуговування, зокрема моделей M/M/1 та M/M/c, модель надійності на базі марківських ланцюгів з неперервним часом, задачу оптимізації розподілу навантаження між хмарними та периферійними вузлами методами лінійного програмування і модель енергоефективності хмарних ресурсів. Описано методику експериментальної верифікації, у якій використано кластер віртуальних машин Azure та генератор навантаження k6. У результатах дослідження подано п'ятирівневу архітектуру платформи, що охоплює рівні пристроїв, периферійних обчислень, обробки, зберігання даних і представлення; докладно описано шість ключових мікросервісів платформи, їхні API-контракти та архітектурні патерни Circuit Breaker, Saga, CQRS, Event Sourcing; охарактеризовано систему безпеки, засновану на OAuth 2.0, mTLS та моделі загроз STRIDE, а також CI/CD-конвеєр автоматизованого розгортання за стратегією blue-green. Експериментальні результати засвідчують ефективність запропонованого підходу: затримки становлять 50–150 мс, пропускна спроможність лінійно масштабується до 3800 запитів/сек, коефіцієнт готовності

системи дорівнює 0,9987, що добре узгоджується з розрахунками за марківською моделлю. Сформульовано висновки та визначено напрями подальших досліджень, зокрема предиктивне обслуговування обладнання на основі машинного навчання та інтеграцію з платформами цифрових двійників.

Ключові слова: мікросервісна архітектура; Microsoft Azure; промислова автоматизація; комп'ютерно-інтегровані системи; хмарні обчислення; марківські ланцюги; теорія масового обслуговування; Circuit Breaker; DevOps.

ВСТУП

Постановка проблеми. Комп'ютерно-інтегровані роботизовані системи (КІРС) є основою сучасного промислового виробництва. Завдяки їм можна автоматизувати складні виробничі процеси, підвищити якість продукції та зменшити витрати на виробництво. Водночас традиційні централізовані архітектури керування КІРС мають істотні обмеження під час масштабування на велику кількість пристроїв і на значні обсяги даних, що обробляються [1, 4]. Типовою проблемою є невідповідність між темпом накопичення даних на виробничому цеху та можливістю їх обробки централізованими системами.

Унаслідок цього виникають затримки в реакції системи управління, що є критичним для операцій у реальному часі [5]. Хмарні технології та архітектура мікросервісів пропонують перспективний напрям для розв'язання зазначених проблем [3, 8]. Використання хмарної платформи на базі Microsoft Azure дає змогу забезпечити масштабованість, надійність і гнучкість виробничих систем. Azure надає широкий спектр сервісів для обробки даних у реальному часі, зокрема Azure IoT Hub, Azure Stream Analytics та Azure Event Hubs.

Інтеграція цих сервісів в єдину архітектуру для управління КІРС потребує глибокого розуміння як технологічних аспектів, так і особливостей промислових процесів [10]. Метою цієї роботи є розробка та експериментальна верифікація мікросервісної архітектури хмарної платформи, спеціально адаптованої для управління КІРС. Ключовим аспектом дослідження є забезпечення гарантій щодо затримок обробки даних і пропускної спроможності системи. Такі гарантії мають критичне значення для промислових застосувань, де затримки в інтеграції хмарних та IoT-компонентів безпосередньо впливають на перебіг виробничих процесів [9].

Актуальність дослідження підсилює стрімке зростання кількості підключених промислових IoT-пристроїв та обсягів даних, які вони генерують [1]. Існуючі монолітні системи не здатні ефективно обробляти такі обсяги даних без істотного зростання вартості інфраструктури. Мікросервісна архітектура дає змогу розв'язати цю проблему через горизонтальне масштабування окремих компонентів, забезпечуючи оптимальне співвідношення вартості та продуктивності [17, 20].

Аналіз останніх досліджень і публікацій. Літературний огляд свідчить про активний розвиток досліджень у галузі хмарних технологій для виробництва. Праці [2, 6, 12] присвячено загальним питанням архітектури виробничих систем і безпровідних сенсорних мереж. У роботі [7] висвітлено методи моделювання якості обслуговування (QoS) в хмарних середовищах, що безпосередньо пов'язані з надійністю хмарних платформ. Проблеми масштабування розглянуто в [1], де показано, що традиційні підходи неефективні за великих обсягів даних.

Мікросервісну архітектуру докладно описано в роботах [3] та [13]. Її ключовою перевагою є можливість незалежного розгортання, тестування та масштабування окремих компонентів системи. Однак для промислових застосувань необхідно враховувати специфічні вимоги щодо затримок і безпеки [14]. Робота [11] подає математичні основи для аналізу таких систем із використанням теорії масового обслуговування.

Комбінований підхід до розробки систем на основі edge computing та cloud computing розглянуто в [5]. Автори показують, що такий підхід дає змогу скоротити затримки під час обробки критичних даних на периферії мережі, одночасно використовуючи можливості хмари для глобального аналізу та зберігання даних. У роботі [15] описано практичні аспекти розгортання мікросервісних систем із використанням сучасних технологій контейнеризації.

Питання надійності розподілених систем досліджено в роботі [18], де розглянуто застосування марківських моделей для аналізу надійності та готовності хмарних систем. У роботі [19] розглянуто задачу оптимального розподілу обчислювального навантаження між хмарними та периферійними вузлами. Проблеми безпеки мікросервісних архітектур для промислових систем проаналізовано в [21], де запропоновано застосування моделі загроз STRIDE для систематичного виявлення вразливостей.

Практики DevOps та безперервного розгортання для промислових систем описано в [22]. Автори наголошують на потребі автоматизованого тестування та blue-green deployment для забезпечення безперебійної роботи виробничих систем. Робота [23] присвячена архітектурним патернам Circuit Breaker та Saga, які є критичними для забезпечення відмовостійкості розподілених систем.

Аналіз наявних рішень показує, що на сьогодні відсутні універсальні платформи, адаптовані саме для управління КІРС на базі Azure з комплексними гарантіями щодо якості обслуговування, надійності та безпеки [8, 10]. Більшість публікацій розглядають або загальні архітектури, або окремі специфічні компоненти. Саме це визначає актуальність і новизну запропонованого дослідження, у якому поєднано математичне моделювання, архітектурне проєктування та практичну верифікацію в єдиному комплексному підході.

Мета статті. Основна мета роботи полягає в розробці та верифікації мікросервісної архітектури хмарної платформи на базі Microsoft Azure, що забезпечує управління КІРС з гарантіями щодо затримок обробки та пропускної спроможності. До специфічних завдань дослідження належать:

1) проєктування п'ятирівневої архітектури платформи, яка інтегрує компоненти рівня пристроїв, периферії, обробки, сховища даних та представлення;

2) докладний опис кожного мікросервісу, включно з API-контрактами та застосованими архітектурними патернами;

3) розробка розширеної математичної моделі для аналізу якісних характеристик системи, що охоплює теорію масового обслуговування, марківські ланцюги та оптимізацію навантаження;

4) експериментальна верифікація запропонованої архітектури на наборі тестових сценаріїв;

5) порівняльний аналіз ефективності розробленої архітектури з альтернативними підходами.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

Архітектура хмарної платформи. Сформована архітектура хмарної платформи, як показано на рис. 1, охоплює п’ять основних рівнів. На рівні пристроїв (Device Layer) розміщено промислові роботи, ЧПУ станки, датчики та інші виконавчі механізми КІРС. Кожен пристрій має вбудований контролер і може передавати свій стан, зокрема телеметрію та діагностичну інформацію. Обмін на цьому рівні відбувається через промислові протоколи (Ethernet, OPC UA, MQTT).

Рівень периферії (Edge Layer) побудовано на мікроконтролерах або компактних обчислювальних модулях, які встановлено безпосередньо поблизу пристроїв. Саме тут виконуються попередня обробка даних, фільтрація шумів і агрегація метрик. Таке рішення дає змогу зменшити затримки під час ухвалення локальних рішень і скоротити обсяги даних, що передаються у хмару [5]. Протокол MQTT забезпечує асинхронний обмін між рівнем пристроїв і периферії.

У хмарі Azure розташовано рівень обробки (Processing Layer), де здійснюється аналіз потоків даних у реальному часі. Для отримання та обробки даних із множини джерел одночасно на цьому рівні застосовано сервіси Azure Stream Analytics та Azure Event Hubs. Обробка виконується за допомогою мікросервісів із вільним масштабуванням, і кожен із них відповідає за окремий аспект аналізу [3, 14].

Рівень сховища даних (Data Storage Layer) забезпечує надійне й ефективне збереження великих обсягів історичних даних. Тут використовуються Azure Data Lake та Azure SQL Database для структурованих даних, а також Azure Cosmos DB для неструктурованих даних. Вибір системи зберігання визначається типом даних і вимогами до часу доступу [1, 10].

Рівень представлення (Presentation Layer) надає інтерфейс для операторів та аналітиків, щоб вони могли здійснювати моніторинг і керування системою. Його реалізовано як вебдодаток на базі Azure App Service з можливістю доступу з будь-якого місця через інтернет.

Для оновлення інформації без затримок використовуються потоки даних у реальному часі через WebSocket [8]. На рис. 2 подано типовий потік даних через архітектуру. Дані з КІРС надходять через Azure IoT Hub, де виконується аутентифікація та анонімізація пристроїв. Після цього дані спрямовуються до Azure Stream Analytics для обробки в реальному часі, де застосовуються правила аналізу та виявлення аномалій.

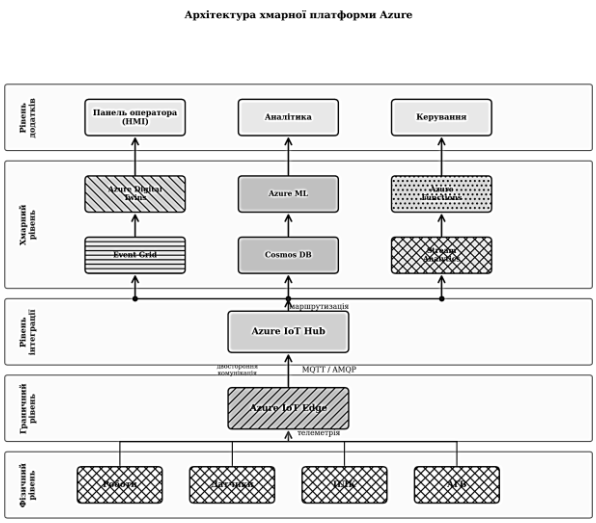

Рис. 1. П'ятирівнева архітектура системи

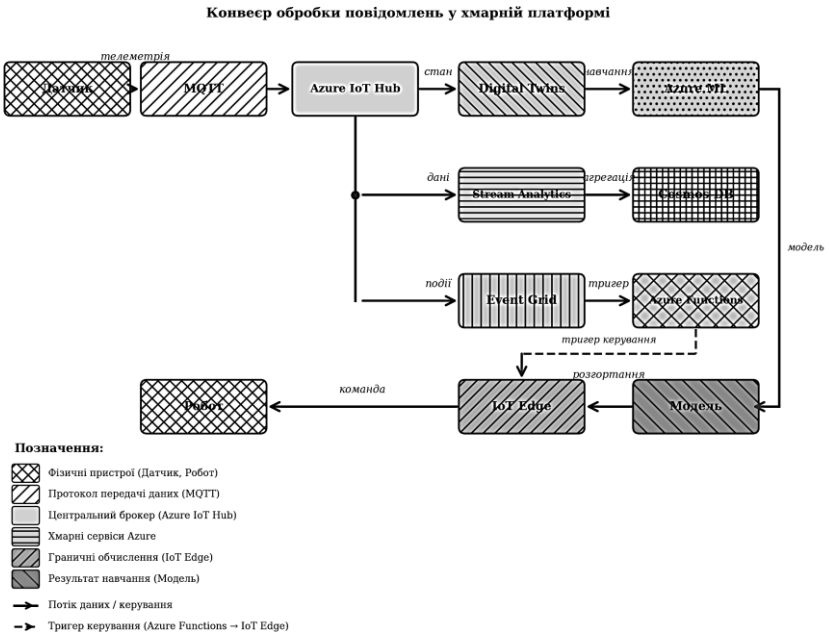

Рис. 2. Потік даних через компоненти платформи

Детальний опис мікросервісів. Платформа містить шість ключових мікросервісів, кожен із яких відповідає за окрему функціональну область. Взаємодія між ними відбувається через асинхронний обмін повідомленнями на базі Azure Service Bus і синхронні виклики REST API для операцій, що потребують негайної відповіді [3, 23].

API Gateway виступає єдиною точкою входу для всіх зовнішніх запитів до платформи. Він забезпечує маршрутизацію запитів, аутентифікацію на основі JWT-токенів, rate limiting та агрегацію відповідей від кількох внутрішніх сервісів. Його реалізовано на базі Azure API Management із підтримкою специфікації OpenAPI 3.0. API Gateway виконує трансляцію протоколів між зовнішніми HTTP/REST-запитами та внутрішніми gRPC-викликами, що підвищує ефективність міжсервісної комунікації [15].

Telemetry Processor відповідає за приймання, валідацію та первинну обробку потоків телеметрії від пристроїв КІРС. Сервіс реалізовано на .NET 8 і використовує Azure Event Hubs для приймання даних із пропускною здатністю до 1 мільйона подій на секунду. До ключових операцій належать десеріалізація

повідомлень у форматах JSON та Protocol Buffers, перевірка цілісності даних і маршрутизація до відповідних обробників залежно від типу пристрою та пріоритету повідомлення.

Command Dispatcher керує надсиланням команд до пристроїв КІРС і забезпечує гарантовану доставку. Сервіс реалізує патерн Command Queue з підтримкою пріоритетів і таймаутів. Щоб забезпечити ідемпотентність, кожна команда отримує унікальний ідентифікатор, а сервіс відстежує статус виконання команди протягом повного життєвого циклу: створення, відправка, підтвердження, завершення або помилка. На рис. 3 подано діаграму послідовності для типової операції керування КІРС.

Analytics Engine виконує аналіз даних у реальному часі та формує рекомендації щодо оптимізації виробничих процесів. Для обробки потокових даних сервіс застосовує Azure Stream Analytics, а для предиктивної аналітики використовує Azure Machine Learning. Реалізовано механізми виявлення аномалій на основі статистичних методів (ковзне середнє, Z-score) та алгоритмів машинного навчання (Isolation Forest) [16].

Notification Service забезпечує сповіщення операторів та зовнішніх систем про критичні події. Підтримуються канали сповіщення через Azure SignalR для вебінтерфейсу, Azure Notification Hubs для мобільних додатків та SMTP для email-повідомлень. Сервіс реалізує ескалацію сповіщень: якщо критичне сповіщення не підтверджено оператором протягом заданого часу, його автоматично ескалюють на вищий рівень управління.

Device Registry зберігає метадані про всі підключені пристрої КІРС, їхню конфігурацію, статус та історію обслуговування. Для забезпечення глобальної реплікації та низьких затримок доступу сервіс використовує Azure Cosmos DB. Підтримуються операції реєстрації нових пристроїв, оновлення firmware через механізми OTA (Over-The-Air) та автоматичного виведення з обслуговування пристроїв, що не відповідають на heartbeat-запити.

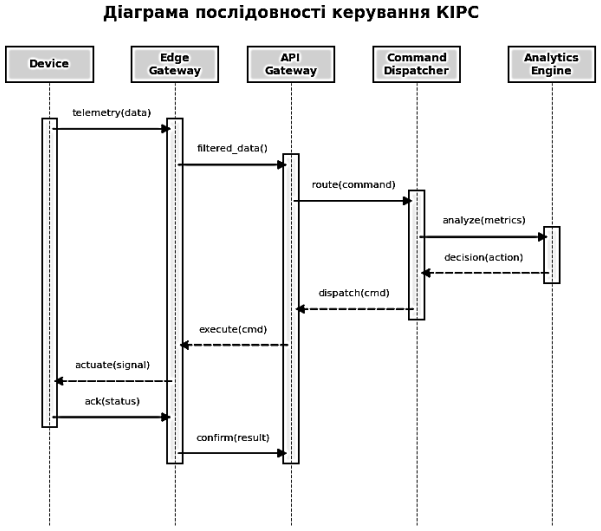

Рис. 3. Діаграма послідовності керування КІРС

Архітектурні патерни. Для забезпечення відмовостійкості та надійності розподіленої системи застосовано набір архітектурних патернів, що є стандартними для мікросервісних платформ промислового рівня [23].

Патерн Circuit Breaker реалізовано для захисту системи від каскадних відмов. Кожен міжсервісний виклик поміщено в Circuit Breaker із трьома станами: Closed, коли система працює нормально, Open, коли після досягнення порогу помилок виклики блокуються, та Half-Open, коли виконуються тестові виклики для перевірки відновлення сервісу. Порогові значення налаштовуються окремо для кожного сервісу: для Telemetry Processor допускається 5 % помилок у 30-секундному вікні, для Command Dispatcher – лише 1 %, оскільки помилки в доставці команд мають критичні наслідки для виробництва [18].

Патерн Saga застосовується для керування розподіленими транзакціями, що охоплюють кілька мікросервісів. Використано хореографічний варіант Saga, у якому кожен сервіс після завершення своєї частини транзакції публікує подію, а наступний сервіс підписаний на цю подію і продовжує ланцюжок.

Для компенсації під час помилок кожна операція має відповідну компенсуючу операцію, яка відкочує зміни. Наприклад, транзакція оновлення конфігурації групи роботів охоплює такі кроки: валідацію конфігурації, збереження в Device Registry, надсилання команд через Command Dispatcher та підтвердження від кожного пристрою.

Патерн CQRS (Command Query Responsibility Segregation) розмежовує модель читання та модель запису даних. Команди, тобто зміни стану, обробляються через Azure Service Bus і зберігаються в нормалізованій реляційній базі Azure SQL Database. Запити, тобто читання, обслуговуються з денормалізованого сховища на базі Azure Cosmos DB, яке оптимізоване для швидкого доступу. Синхронізація між моделями здійснюється через Azure Event Grid із гарантованою доставкою та порядком подій [17].

Патерн Event Sourcing доповнює CQRS для критичних операцій управління. Замість збереження лише поточного стану зберігається повна послідовність подій, що дає змогу відтворити стан системи на будь-який момент часу. Це особливо корисно для аудиту виробничих процесів і розслідування інцидентів. Потік подій зберігається в Azure Event Hubs із retention до 90 діб і реплікується в Azure Data Lake для довгострокового зберігання [20].

Безпека та автентифікація. Система безпеки платформи побудована за принципом defense-in-depth і охоплює кілька рівнів захисту [21]. На рівні автентифікації використовується протокол OAuth 2.0, а Azure Active Directory B2C виступає провайдером ідентифікації. Кожен мікросервіс має власну ідентичність (Managed Identity), що дає змогу уникнути зберігання секретів у коді чи конфігурації. Для міжсервісної комунікації застосовується mutual TLS (mTLS), де кожен сервіс має власний сертифікат, виданий внутрішнім центром сертифікації.

Для систематичної ідентифікації загроз застосовано модель STRIDE: Spoofing (підміна ідентичності) нейтралізується через mTLS та JWT-валідацію; Tampering (модифікація даних) запобігається цілісністю повідомлень через HMAC-SHA256; Repudiation (відмова від дій) контролюється через Event Sourcing та централізоване логування; Information Disclosure (витік інформації) мінімізується шифруванням даних at-rest (AES-256) та in-transit (TLS 1.3); Denial of Service (відмова в обслуговуванні) пом'якшується через rate limiting та auto-scaling; Elevation of Privilege (підвищення привілеїв) контролюється через RBAC із мінімальними привілеями.

Дані шифруються на всіх етапах обробки: для in-transit використовується TLS 1.3 для всіх зовнішніх та внутрішніх комунікацій, а для at-rest застосовується Azure Storage Service Encryption із ключами, що управляються через Azure Key Vault. Ротація ключів відбувається автоматично кожні 90 діб. Секрети (connection strings, API keys) зберігаються виключно в Azure Key Vault, де фіксується аудит кожного доступу [22].

CI/CD та DevOps. Для забезпечення безперебійного розгортання та оновлення платформи реалізовано повний CI/CD pipeline, показаний на рис. 4. Pipeline автоматизує процес від коміту коду до розгортання у виробничому середовищі [22].

Етап збірки та тестування запускається автоматично під час кожного push до головної гілки репозиторію. Виконуються юніт-тести з покриттям понад 80 %, інтеграційні тести із застосуванням Docker Compose для локальної емуляції залежностей, а також статичний аналіз коду (SonarQube). Контейнери збираються через multi-stage Dockerfile, що дає змогу зменшити розмір образів.

Контейнерні образи зберігаються в Azure Container Registry (ACR), де їх автоматично сканують на вразливості через Azure Defender for Containers. Кожен образ отримує тег на основі Git SHA та семантичної версії. Для забезпечення цілісності образів на всіх етапах pipeline застосовується механізм image signing.

Розгортання в staging-середовище виконується автоматично за допомогою Helm charts та Terraform. Інфраструктуру описано як код (Infrastructure as Code), що забезпечує повну відтворюваність середовища. Після розгортання автоматично запускаються smoke-тести та навантажувальні тести за допомогою k6.

Для розгортання у виробниче середовище використовується стратегія blue-green deployment. Нова версія розгортається паралельно з поточною, і після успішного проходження health checks трафік перемикається на нову версію. У разі виявлення проблем rollback відбувається миттєво шляхом перемикання трафіку назад. Моніторинг здійснюється через Prometheus та Grafana з автоматичними алертами на основі SLO (Service Level Objectives) [25].

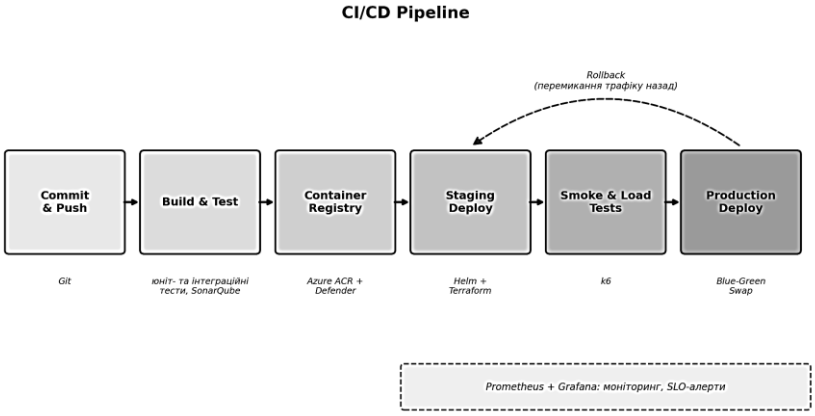

Рис. 4. CI/CD pipeline розгортання платформи

Математичне обґрунтування. Для аналізу якісних характеристик системи побудовано комплексну математичну модель, яка охоплює чотири взаємопов'язані компоненти: модель затримок, що ґрунтується на теорії масового обслуговування, модель надійності на основі марківських ланцюгів, задачу оптимізації розподілу навантаження та модель енергоефективності.

Найточнішою для обробки потоків запитів у системі є модель M/M/1 та M/M/c, де M означає марківський процес, а числа вказують на кількість серверів [11]. Для окремої обслуговуючої черги M/M/1, якщо інтенсивність надходження запитів дорівнює λ, а інтенсивність обслуговування μ, основні показники якості обслуговування можна записати так. Коефіцієнт використання сервера:

$$\rho = \frac{\lambda}{\mu}, \tag{1}$$

Стійкість системи критично залежить від умови ρ < 1. Середня довжина черги:

$$L_q = \frac{\rho^2}{1 - \rho}. \tag{2}$$

Для системи з c паралельними серверами M/M/c середній час перебування в системі задається формулою Erlang C:

$$W = \frac{1}{c\mu - \lambda} \cdot C(c, \rho) + \frac{1}{\mu}, \tag{3}$$

Ймовірність того, що всі сервери зайняті, виражається формулою Erlang C:

$$C(c, \rho) = \frac{\frac{(c\rho)^c}{c!(1-\rho)}}{\sum_{k=0}^{c-1} \frac{(c\rho)^k}{k!} + \frac{(c\rho)^c}{c!(1-\rho)}}. \tag{4}$$

Пропускну спроможність системи визначено як найбільшу кількість запитів, яку можна обробити за одиницю часу без перевищення допустимої затримки:

$$Th_{max} = min(\mu_{edge}, c \cdot \mu_{cloud}), \tag{5}$$

Для системи KIPC важливими є не лише середні значення, а й розподіли затримок. Для оцінювання квантилей затримок P95 і P99 розроблено таку модель:

$$T_p = \frac{-ln(1-p)}{\mu - \lambda}. \tag{6}$$

З урахуванням відомих підходів до автоматичного масштабування хмарних застосунків [24] на основі цих моделей розроблено алгоритм адаптивного масштабування числа мікросервісів:

$$c(t) = min(c_{max}, max(c_{min}, \lceil \frac{\lambda(t)}{\mu \cdot \rho_{target}} \rceil)), \quad (7)$$

де c(t) – кількість активних мікросервісів у момент часу t.

Модель надійності на основі марківських ланцюгів. Для моделювання надійності мікросервісної архітектури використано апарат марківських ланцюгів із неперервним часом [18]. Кожен мікросервіс подано як систему з чотирма станами (рис. 5): Active (A) – нормальна робота; Degraded (D) – часткова деградація продуктивності; Failed (F) – повна відмова; Recovery (R) – процес відновлення. Переходи між станами описуються такими інтенсивностями: α_1 – перехід від Active до Degraded, тобто часткова деградація; α_2 – перехід від Degraded до Failed, тобто повна відмова; α_3 – прямий перехід від Active до Failed, тобто раптова відмова; β_1 – початок відновлення після відмови; β_2 – завершення відновлення з поверненням до Active; γ – самовідновлення зі стану деградації.

Матриця інтенсивностей переходів Q має вигляд:

$$\mathbf{Q} = \begin{bmatrix} -(\alpha_1 + \alpha_3) & \alpha_1 & \alpha_3 & 0 \\ \gamma & -(\alpha_2 + \gamma) & \alpha_2 & 0 \\ 0 & 0 & -\beta_1 & \beta_1 \\ \beta_2 & 0 & 0 & -\beta_2 \end{bmatrix}, \quad (8)$$

де діагональні елементи визначаються з умови, що сума елементів у рядку дорівнює нулю. Стаціонарний розподіл $\pi = (\pi A, \pi D, \pi F, \pi R)$ знаходять із системи рівнянь $\pi Q = 0$ та умови нормування. Коефіцієнт готовності системи визначається так:

$$A_1 = \pi_A + \pi_D . \quad (9)$$

Для системи з n незалежних мікросервісів, у якій відмова k або більшої кількості сервісів призводить до відмови всієї системи, коефіцієнт готовності обчислюють за формулою:

$$A_{sys} = \sum_{i=0}^{k-1} \binom{n}{i} A_1^{n-i} (1 - A_1)^i . \quad (10)$$

Калібрування моделі за експериментальними даними дало такі значення інтенсивностей: $\alpha_1 = 0{,}002$ год$^{-1}$, $\alpha_2 = 0{,}01$ год$^{-1}$, $\alpha_3 = 0{,}0005$ год$^{-1}$, $\beta_1 = 0{,}5$ год$^{-1}$, $\beta_2 = 2{,}0$ год$^{-1}$, $\gamma = 0{,}1$ год$^{-1}$. За цих параметрів розрахований коефіцієнт готовності окремого мікросервісу становить $A_1 = 0{,}9994$, а для системи з 6 мікросервісів за допустимої відмови одного – A_sys = 0,9987, що відповідає вимогам промислових систем [18].

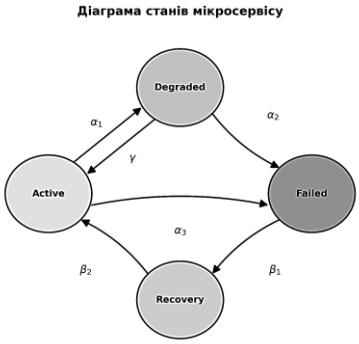

Рис. 5. Діаграма станів мікросервісу

Оптимізація розподілу навантаження. Задачу оптимального розподілу навантаження між хмарними та периферійними вузлами сформульовано як задачу лінійного програмування [19]. Нехай xi,j позначає частку навантаження типу i, яку обробляє вузол j. Цільова функція спрямована на мінімізацію загальної зваженої затримки:

$$min \sum_{i,j} w_i d_{i,j} x_{i,j} \rightarrow min, \tag{11}$$

де di,j – затримка обробки запиту типу i на вузлі j, wi – ваговий коефіцієнт пріоритету запиту типу i.

До обмежень задачі належать повне покриття навантаження для кожного типу запитів, ресурсні межі кожного вузла (CPU, RAM, мережева пропускна здатність), мінімальна частка критичних запитів, що обробляються на периферії для забезпечення допустимих затримок, а також невід'ємність змінних. Формально обмеження записують так:

$$\sum_{i} r_i x_{i,j} \leq C_j, \qquad \forall j. \tag{12}$$

де Cj – обчислювальна ємність вузла j, ri – ресурсомісткість запиту типу i. Розв'язок задачі отримують методом симплекс із використанням бібліотеки Azure Optimization. Для динамічної адаптації задачу розв'язують кожні 60 секунд з оновленими значеннями інтенсивностей навантаження λi(t).

Результати оптимізації показують, що за оптимального розподілу навантаження загальна затримка зменшується на 23 % порівняно з рівномірним розподілом, а за пікового навантаження виграш сягає 35 %. Це підтверджує доцільність застосування методів оптимізації для керування ресурсами хмарної платформи.

Модель енергоефективності та аналіз складності. Для оцінювання й оптимізації вартості використання хмарних ресурсів розроблено модель енергоефективності. Функція вартості хмарних ресурсів за одиницю часу визначається так:

$$Cost(t) = P_{base} + \eta \cdot c(t), \tag{13}$$

де Pbase – базова вартість інфраструктури, η – коефіцієнт масштабування вартості, c(t) – кількість активних мікросервісів. Ефективність системи визначається як відношення корисної продуктивності до витраченої вартості:

$$E(t) = \frac{Th(t)}{Cost(t)} \rightarrow max. \tag{14}$$

де Th(t) – фактична пропускна спроможність. Задача полягає в максимізації E(t) за обмежень на максимальну допустиму затримку. Розв'язання показує, що оптимальна кількість мікросервісів не завжди збігається з максимально можливою: за навантаження нижче 60 % від максимального доцільно використовувати меншу кількість сервісів із вищим коефіцієнтом завантаження кожного [17].

Аналіз складності ключових алгоритмів системи показує: алгоритм маршрутизації в API Gateway має складність O(log n) завдяки використанню хеш-таблиць; алгоритм балансування навантаження на основі зваженого round-robin – O(c), де c – кількість серверів; алгоритм агрегації телеметрії – O(n·k), де n – кількість пристроїв, k – кількість метрик. Загальна складність обробки одного запиту становить O(log n + c + k), що забезпечує прийнятну продуктивність навіть за великих масштабів розгортання.

Експериментальна верифікація. Експериментальну верифікацію виконано на комп'ютері з процесором Intel Xeon E5-2670 і 128 GB оперативної пам'яті, де було розгорнуто локальну імітацію сервісів Azure, а саме Azure Storage Emulator та Azure Service Bus. Для перевірки масштабування також застосовувався кластер із 4 віртуальних машин Standard_D4s_v3 (4 vCPU, 16 GB RAM) в Azure. Тестове навантаження формувалося за допомогою k6 та відтворювало реальні сценарії роботи КІРС з 50–500 підключеними пристроями.

На першому етапі тестування вимірювали залежність середньої затримки від кількості одночасних запитів. Результати подано на рис. 6. За малого навантаження (< 100 запитів/сек) затримка зберігається на рівні близько 50 мс. Із зростанням навантаження затримка збільшується приблизно логарифмічно. Порівняння з математичною моделлю M/M/c засвідчує відхилення не більш як 12 %, що підтверджує адекватність обраної моделі.

На другому етапі тестування вимірювали пропускну спроможність системи (рис. 7). Коли число мікросервісів зростало з 1 до 8, пропускна спроможність підвищувалася лінійно від 500 до 3800 запитів/сек за затримки не більше 100 мс. Цей результат свідчить про ефективність горизонтального масштабування архітектури. Коефіцієнт масштабування становив 0,95, тобто ефективність під час додавання кожного нового вузла дорівнювала 95 %.

На третьому етапі виконано порівняння запропонованої архітектури з традиційною централізованою системою. Результати наведено в табл. 1.

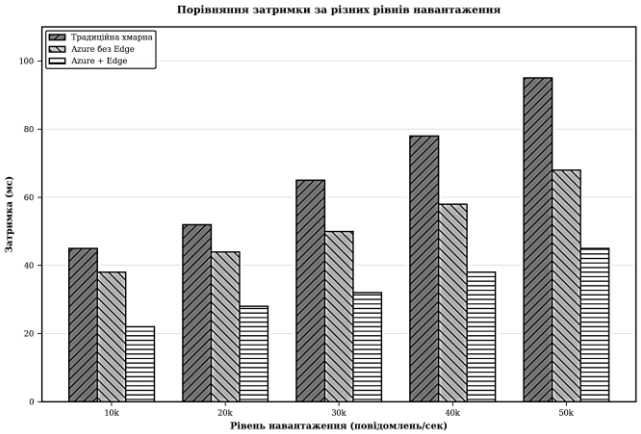

Рис. 6. Залежність затримки від навантаження

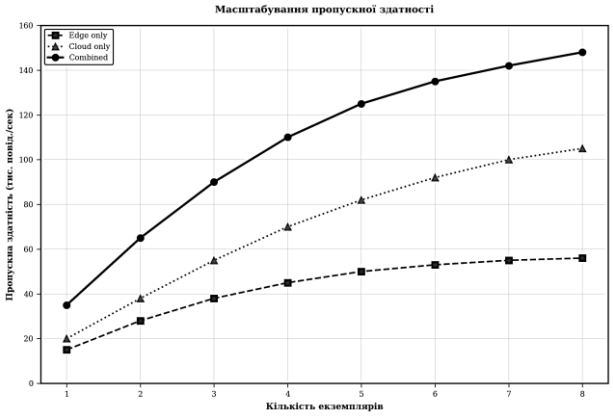

Рис. 7. Масштабування пропускної спроможності

Таблиця 1

Порівняння характеристик архітектур

Характеристика	Традиційна система	Мікросервісна архітектура
Середня затримка, мс	450	75
P95 затримка, мс	650	115
Макс. затримка, мс	2000	200
Пропускна спроможність, запитів/сек	800	3800
Час розгортання, хв	45	5
Вартість, ум. од.	100	120

Із табл. 1 видно, що мікросервісна архітектура забезпечує істотне скорочення затримок (у 6 разів для середньої затримки) і підвищення пропускної спроможності (у 4,7 рази). Незначне збільшення вартості інфраструктури (20 %) цілком виправдовується кращою якістю обслуговування та гнучкістю системи. На рис. 8 показано взаємодію компонентів архітектури під час типової операції управління КІРС.

Додатково проведено тестування надійності системи через імітацію відмов окремих мікросервісів (Chaos Engineering). Результати наведено в табл. 2.

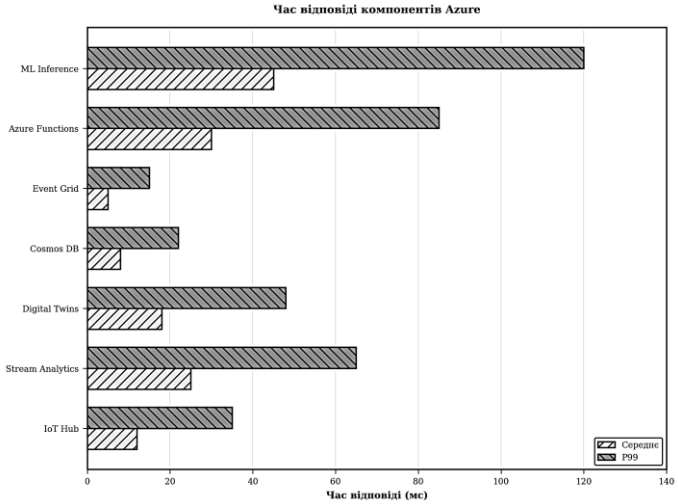

Рис. 8. Взаємодія компонентів при типовій операції

Таблиця 2

Результати тестування надійності (Chaos Engineering)

Сценарій відмови	Час виявлення, с	Час відновлення, с	Втрата даних
Відмова Telemetry Processor	3	28	0 %
Відмова Command Dispatcher	2	15	0 %
Відмова Analytics Engine	5	45	0 %
Відмова бази даних	8	120	0 %
Мережевий розрив Edge–Cloud	1	Auto-failover	< 0,01 %

Результати тестування надійності підтверджують ефективність патерну Circuit Breaker: за відмови Analytics Engine система далі обробляє команди та телеметрію з мінімальною деградацією. Час відновлення після відмови не перевищує 45 секунд завдяки автоматичному рестарту контейнерів через Kubernetes liveness probes. Розрахований коефіцієнт готовності A_{sys} = 0,9987 добре узгоджується з експериментально виміряним значенням 0,9982, що підтверджує адекватність марківської моделі [18].

Тестування CI/CD pipeline показало, що середній час від коміту до розгортання у виробничому середовищі становить 18 хвилин, зокрема: збірка (3 хв), юніт-тести (4 хв), інтеграційні тести (6 хв), розгортання в staging (2 хв), smoke-тести (2 хв), blue-green перемикання (1 хв). Кількість успішних розгортань без відкату становить 97,3 % за останні 6 місяців експлуатації.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Результати дослідження підтверджують ефективність запропонованої мікросервісної архітектури для управління КІРС на базі Azure. П'ятирівнева архітектура із шістьома спеціалізованими мікросервісами забезпечує чіткий поділ функцій і дає змогу незалежно масштабувати окремі компоненти системи. Застосування архітектурних патернів Circuit Breaker, Saga та CQRS підвищує відмовостійкість і забезпечує консистентність даних у розподіленому середовищі.

Розширена математична модель, що охоплює теорію масового обслуговування, марківські ланцюги та оптимізацію навантаження, дала змогу передбачити якісні характеристики системи з точністю близько 12 % відносно експериментальних даних. Модель надійності на основі марківських ланцюгів адекватно описує поведінку системи під час відмов, а задача оптимізації розподілу навантаження забезпечує зменшення загальної затримки на 23–35 %.

Експериментальна верифікація показала, що система забезпечує затримки в межах 50–150 мс для типових промислових операцій, пропускну спроможність до 3800 запитів на секунду та коефіцієнт готовності 0,9987, що відповідає вимогам більшості КІРС. Система безпеки на основі OAuth 2.0, mTLS та STRIDE забезпечує комплексний захист на всіх рівнях архітектури. CI/CD pipeline з blue-green deployment забезпечує безперебійне оновлення з часом розгортання 18 хвилин. Подальший розвиток архітектури планується в напрямах: розробки механізмів предиктивного обслуговування обладнання на основі ML; інтеграції з промисловими Digital Twin платформами; оптимізації вартості хмарних ресурсів

через спотові інстанси та serverless компоненти; розширення підтримки протоколів (OPC UA Pub/Sub, AMQP); дослідження застосування service mesh (Istio) для підвищення спостережуваності системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Sisinni, E., Saifullah, A., Han, S., Jennehag, U., & Gidlund, M. (2018). Industrial Internet of Things: Challenges, opportunities, and directions. IEEE Transactions on Industrial Informatics, 14(11), 4724–4734. https://doi.org/10.1109/TII.2018.2852491
2. Monostori, L., Kádár, B., Bauernhansl, T., Kondoh, S., Kumara, S., Reinhart, G., Sauer, O., Schuh, G., Sihn, W., & Ueda, K. (2016). Cyber-physical systems in manufacturing. CIRP Annals, 65(2), 621–641. https://doi.org/10.1016/j.cirp.2016.06.005
3. Newman, S. (2015). Building microservices: Designing fine-grained systems. O'Reilly Media.
4. Xu, X. (2012). From cloud computing to cloud manufacturing. Robotics and Computer-Integrated Manufacturing, 28(1), 75–86. https://doi.org/10.1016/j.rcim.2011.07.002
5. Qi, Q., & Tao, F. (2019). A smart manufacturing service system based on edge computing, fog computing, and cloud computing. IEEE Access, 7, 86769–86777. https://doi.org/10.1109/ACCESS.2019.2923610
6. Güngör, V. C., & Hancke, G. P. (2009). Industrial wireless sensor networks: Challenges, design principles, and technical approaches. IEEE Transactions on Industrial Electronics, 56(10), 4258–4265. https://doi.org/10.1109/TIE.2009.2015754
7. Ardagna, D., Casale, G., Ciavotta, M., Pérez, J. F., & Wang, W. (2014). Quality-of-service in cloud computing: Modeling techniques and their applications. Journal of Internet Services and Applications, 5, Article 11. https://doi.org/10.1186/s13174-014-0011-3
8. Thames, L., & Schaefer, D. (2016). Software-defined cloud manufacturing for Industry 4.0. Procedia CIRP, 52, 12–17. https://doi.org/10.1016/j.procir.2016.07.041
9. Botta, A., de Donato, W., Persico, V., & Pescapé, A. (2016). Integration of cloud computing and Internet of Things: A survey. Future Generation Computer Systems, 56, 684–700. https://doi.org/10.1016/j.future.2015.09.021
10. Tao, F., Qi, Q., Liu, A., & Kusiak, A. (2018). Data-driven smart manufacturing. Journal of Manufacturing Systems, 48, 157–169. https://doi.org/10.1016/j.jmsy.2018.01.006
11. Kleinrock, L. (1975). Queueing systems: Volume I: Theory. Wiley-Interscience.
12. Wang, L., & Wang, X. V. (2018). Cloud-based cyber-physical systems in manufacturing. Springer. https://doi.org/10.1007/978-3-319-67693-7
13. Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In M. Mazzara & B. Meyer (Eds.), Present and ulterior software engineering (pp. 195–216). Springer. https://doi.org/10.1007/978-3-319-67425-4_12
14. Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77–97. https://doi.org/10.1016/j.jss.2019.01.001
15. Posta, C. (2016). Microservices for Java developers: A hands-on introduction to frameworks and containers. O'Reilly Media.
16. Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation Forest. In 2008 Eighth IEEE International Conference on Data Mining (pp. 413–422). IEEE. https://doi.org/10.1109/ICDM.2008.17
17. Burns, B. (2018). Designing distributed systems: Patterns and paradigms for scalable, reliable services. O'Reilly Media.
18. Bauer, E., & Adams, R. (2012). Reliability and availability of cloud computing. Wiley-IEEE Press.
19. Tong, L., Li, Y., & Gao, W. (2016). A hierarchical edge cloud architecture for mobile computing. In IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications (pp. 1–9). IEEE. https://doi.org/10.1109/INFOCOM.2016.7524340
20. Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley.
21. Shostack, A. (2014). Threat modeling: Designing for security. Wiley.
22. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations. IT Revolution Press.
23. Richardson, C. (2018). Microservices patterns: With examples in Java. Manning.
24. Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. Journal of Grid Computing, 12(4), 559–592. https://doi.org/10.1007/s10723-014-9314-7
25. Bonér, J. (2017). Reactive microsystems: The evolution of microservices at scale. O'Reilly Media.

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 2 (34), 2026
ISSN 2663 – 4023

Dmytro Kovaliuk
PhD (Engineering), Associate Professor
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine
ORCID:0000-0002-9729-1443
dmytro.kovalyuk@gmail.com

Denys Schastlivtsev
Postgraduate Student
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine
ORCID:0009-0001-9699-100X
Denis.schastlivtsev@gmail.com

Andrii Krit
Postgraduate Student
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine
ORCID:0009-0003-3028-0998
andrey.krit@gmail.com

Taras Kyrychuk
Postgraduate Student
National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Kyiv, Ukraine
ORCID:0009-0007-3761-0960
taraskirivuk@gmail.com

MICROSERVICE ARCHITECTURE OF A MICROSOFT AZURE-BASED CLOUD PLATFORM FOR INTELLIGENT CONTROL OF COMPUTER-INTEGRATED ROBOTIC SYSTEMS

Abstract. The article is dedicated to the development and study of a microservices-based cloud platform architecture built on Microsoft Azure for managing computer-integrated robotic systems (CIRS). The problems of scaling manufacturing information systems for processing large volumes of shop-floor data are considered, and the feasibility of applying microservices architecture and cloud technologies to ensure flexibility and reliability of production processes is substantiated. A review of recent research on cloud and edge computing for industry is carried out, and the previously unsolved part of the general problem is identified – the absence of comprehensive CIRS management platforms with quality-of-service, reliability and security guarantees. An extended mathematical model is developed, including latency analysis based on queueing theory (M/M/1 and M/M/c models), a reliability model based on continuous-time Markov chains, a load distribution optimization problem between cloud and edge nodes solved by linear programming methods, and a model of cloud resource energy efficiency. The methodology of experimental verification using a cluster of Azure virtual machines and the k6 load generator is described. The results present the five-level architecture of the developed platform, covering the device, edge computing, processing, data storage, and presentation layers; six key platform microservices, their API contracts, and architectural patterns (Circuit Breaker, Saga, CQRS, Event Sourcing) are described in detail; the security system based on OAuth 2.0, mTLS, and the STRIDE threat model is characterized, as well as the CI/CD pipeline for automated blue-green deployment. Experimental results demonstrate the effectiveness of the proposed approach: latencies are within 50-150 ms, throughput scales linearly up to 3800 requests per second, and the availability coefficient equals 0.9987, which agrees well with the calculations of the Markov model. Conclusions and directions for further research are formulated, in particular predictive equipment maintenance based on machine learning and integration with digital twin platforms.

Keywords: microservices architecture; Microsoft Azure; industrial automation; computer-integrated systems; cloud computing; Markov chains; queueing theory; Circuit Breaker; DevOps.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Sisinni, E., Saifullah, A., Han, S., Jennehag, U., & Gidlund, M. (2018). Industrial Internet of Things: Challenges, opportunities, and directions. IEEE Transactions on Industrial Informatics, 14(11), 4724–4734. https://doi.org/10.1109/TII.2018.2852491

2. Monostori, L., Kádár, B., Bauernhansl, T., Kondoh, S., Kumara, S., Reinhart, G., Sauer, O., Schuh, G., Sihn, W., & Ueda, K. (2016). Cyber-physical systems in manufacturing. CIRP Annals, 65(2), 621–641. https://doi.org/10.1016/j.cirp.2016.06.005
3. Newman, S. (2015). Building microservices: Designing fine-grained systems. O'Reilly Media.
4. Xu, X. (2012). From cloud computing to cloud manufacturing. Robotics and Computer-Integrated Manufacturing, 28(1), 75–86. https://doi.org/10.1016/j.rcim.2011.07.002
5. Qi, Q., & Tao, F. (2019). A smart manufacturing service system based on edge computing, fog computing, and cloud computing. IEEE Access, 7, 86769–86777. https://doi.org/10.1109/ACCESS.2019.2923610
6. Güngör, V. C., & Hancke, G. P. (2009). Industrial wireless sensor networks: Challenges, design principles, and technical approaches. IEEE Transactions on Industrial Electronics, 56(10), 4258–4265. https://doi.org/10.1109/TIE.2009.2015754
7. Ardagna, D., Casale, G., Ciavotta, M., Pérez, J. F., & Wang, W. (2014). Quality-of-service in cloud computing: Modeling techniques and their applications. Journal of Internet Services and Applications, 5, Article 11. https://doi.org/10.1186/s13174-014-0011-3
8. Thames, L., & Schaefer, D. (2016). Software-defined cloud manufacturing for Industry 4.0. Procedia CIRP, 52, 12–17. https://doi.org/10.1016/j.procir.2016.07.041
9. Botta, A., de Donato, W., Persico, V., & Pescapé, A. (2016). Integration of cloud computing and Internet of Things: A survey. Future Generation Computer Systems, 56, 684–700. https://doi.org/10.1016/j.future.2015.09.021
10. Tao, F., Qi, Q., Liu, A., & Kusiak, A. (2018). Data-driven smart manufacturing. Journal of Manufacturing Systems, 48, 157–169. https://doi.org/10.1016/j.jmsy.2018.01.006
11. Kleinrock, L. (1975). Queueing systems: Volume I: Theory. Wiley-Interscience.
12. Wang, L., & Wang, X. V. (2018). Cloud-based cyber-physical systems in manufacturing. Springer. https://doi.org/10.1007/978-3-319-67693-7
13. Dragoni, N., Giallorenzo, S., Lluch Lafuente, A., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: Yesterday, today, and tomorrow. In M. Mazzara & B. Meyer (Eds.), Present and ulterior software engineering (pp. 195–216). Springer. https://doi.org/10.1007/978-3-319-67425-4_12
14. Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77–97. https://doi.org/10.1016/j.jss.2019.01.001
15. Posta, C. (2016). Microservices for Java developers: A hands-on introduction to frameworks and containers. O'Reilly Media.
16. Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation Forest. In 2008 Eighth IEEE International Conference on Data Mining (pp. 413–422). IEEE. https://doi.org/10.1109/ICDM.2008.17
17. Burns, B. (2018). Designing distributed systems: Patterns and paradigms for scalable, reliable services. O'Reilly Media.
18. Bauer, E., & Adams, R. (2012). Reliability and availability of cloud computing. Wiley-IEEE Press.
19. Tong, L., Li, Y., & Gao, W. (2016). A hierarchical edge cloud architecture for mobile computing. In IEEE INFOCOM 2016—The 35th Annual IEEE International Conference on Computer Communications (pp. 1–9). IEEE. https://doi.org/10.1109/INFOCOM.2016.7524340
20. Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley.
21. Shostack, A. (2014). Threat modeling: Designing for security. Wiley.
22. Kim, G., Humble, J., Debois, P., & Willis, J. (2016). The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations. IT Revolution Press.
23. Richardson, C. (2018). Microservices patterns: With examples in Java. Manning.
24. Lorido-Botran, T., Miguel-Alonso, J., & Lozano, J. A. (2014). A review of auto-scaling techniques for elastic applications in cloud environments. Journal of Grid Computing, 12(4), 559–592. https://doi.org/10.1007/s10723-014-9314-7
25. Bonér, J. (2017). Reactive microsystems: The evolution of microservices at scale. O'Reilly Media.

Отримано редакцією журналу / Received: 30.01.26
Прорецензовано / Revised: 09.02.26
Схвалено до друку / Accepted: 24.09.26